MODEL ENGINEERING COLLEGE

(Unit of I H R D)

THRIKKAKKARA, ERNAKULAM

LABORATORY RECORD

CSL332 – NETWORKING LAB

NAME: Sumegh S Pai

BRANCH: Computer Science and Engineering

SEMESTER: VI ROLL NO. 20CSB58

Certified that this is the Bonafide work done by

Sumegh S Pai

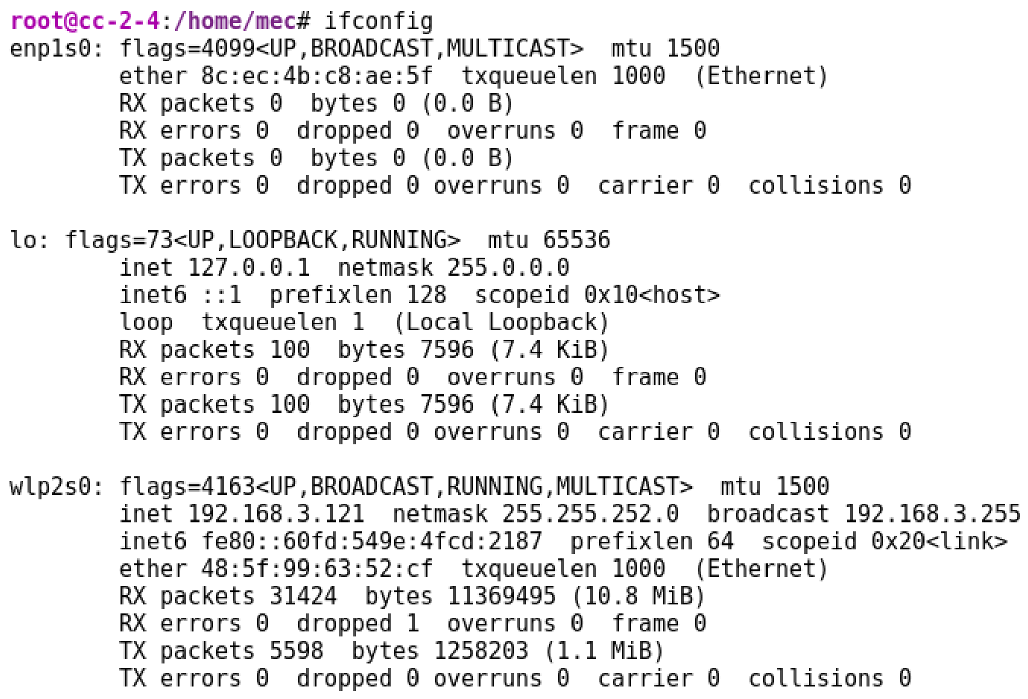
Staff- in Charge ***Head of the Department***

Register No. MDL20CS117 Thrikkakkara

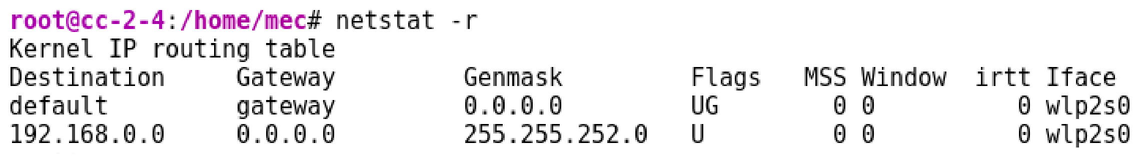
Year & Month...

Date:..

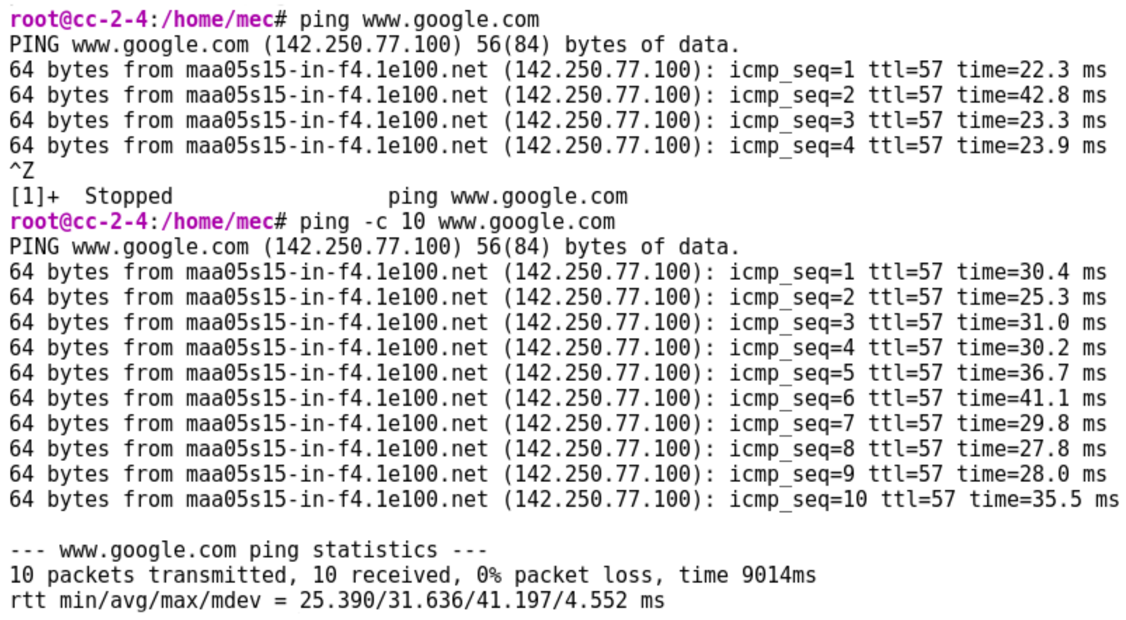
INDEX



```
root@cc-2-4:/home/mec# ifconfig
enp1s0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 8c:ec:4b:c8:ae:5f  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        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 1  (Local Loopback)
        RX packets 100  bytes 7596 (7.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 100  bytes 7596 (7.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.3.121  netmask 255.255.252.0  broadcast 192.168.3.255
        inet6 fe80::60fd:549e:4fcd:2187  prefixlen 64  scopeid 0x20<link>
        ether 48:5f:99:63:52:cf  txqueuelen 1000  (Ethernet)
        RX packets 31424  bytes 11369495 (10.8 MiB)
        RX errors 0  dropped 1  overruns 0  frame 0
        TX packets 5598  bytes 1258203 (1.1 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```



```
root@cc-2-4:/home/mec# netstat -r
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
default         gateway         0.0.0.0         UG        0 0          0 wlp2s0
192.168.0.0     0.0.0.0         255.255.252.0   U         0 0          0 wlp2s0
```



```
root@cc-2-4:/home/mec# ping www.google.com
PING www.google.com (142.250.77.100) 56(84) bytes of data.
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=1 ttl=57 time=22.3 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=2 ttl=57 time=42.8 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=3 ttl=57 time=23.3 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=4 ttl=57 time=23.9 ms
^Z
[1]+  Stopped                 ping www.google.com
root@cc-2-4:/home/mec# ping -c 10 www.google.com
PING www.google.com (142.250.77.100) 56(84) bytes of data.
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=1 ttl=57 time=30.4 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=2 ttl=57 time=25.3 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=3 ttl=57 time=31.0 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=4 ttl=57 time=30.2 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=5 ttl=57 time=36.7 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=6 ttl=57 time=41.1 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=7 ttl=57 time=29.8 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=8 ttl=57 time=27.8 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=9 ttl=57 time=28.0 ms
64 bytes from maa05s15-in-f4.1e100.net (142.250.77.100): icmp_seq=10 ttl=57 time=35.5 ms

--- www.google.com ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9014ms
rtt min/avg/max/mdev = 25.390/31.636/41.197/4.552 ms
```

Date : 28/02/2023

Experiment No : 1
Basic Networking Commands

AIM:

To familiarize with the basics of network configuration files and networking commands in Linux.

THEORY:

<u>Networking Commands</u>

Linux networking commands are used extensively to inspect, analyze, maintain, and troubleshoot the networks connected to the system.

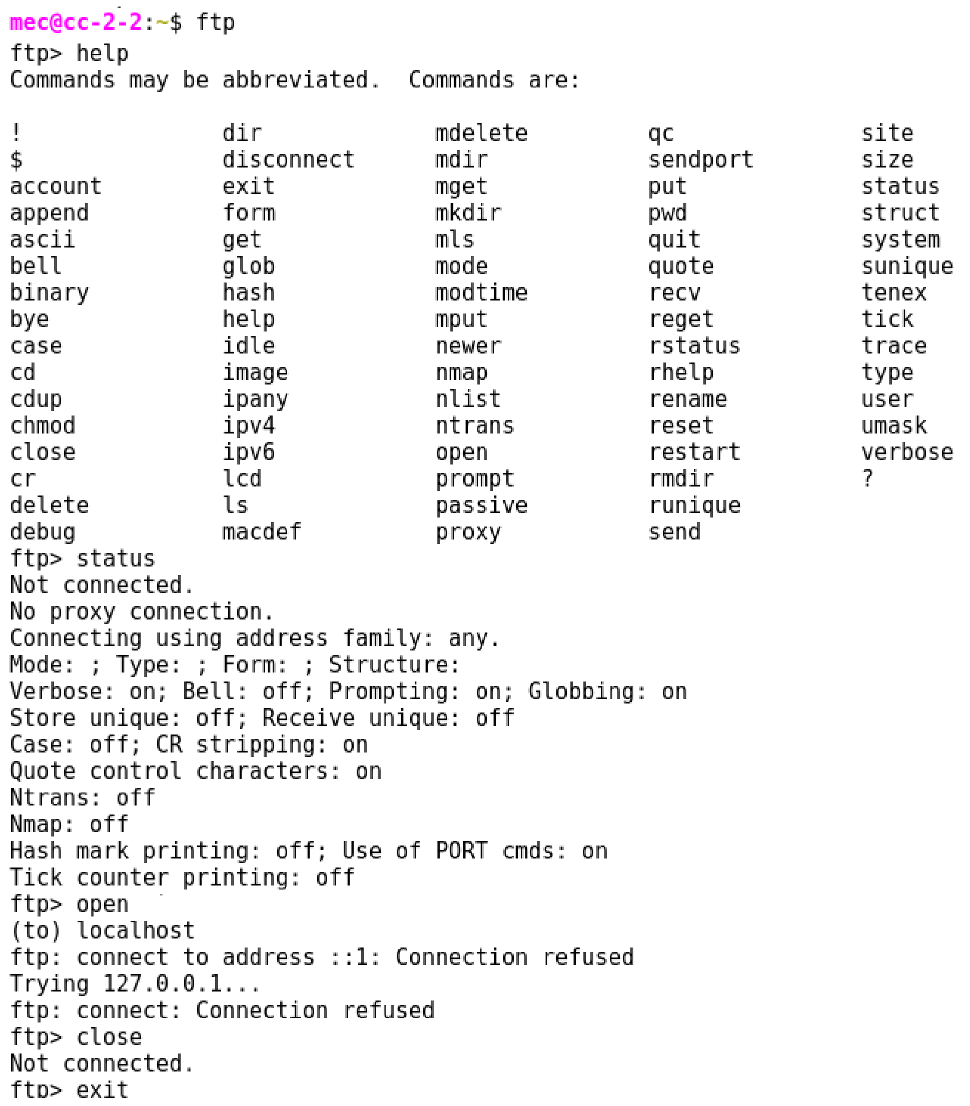
1. Ip : It is a handy tool for configuring the network interfaces for Linux administrators. It can be used to get the details of a specific interface.
   Syntax - ip a
   ip addr

2. traceroute : Linux traceroute command is a network troubleshooting utility that helps us determine the number of hops and packets traveling path required to reach a destination.It can display the routes, IP addresses, and hostnames of routers over a network.
   Syntax - traceroute <destination>

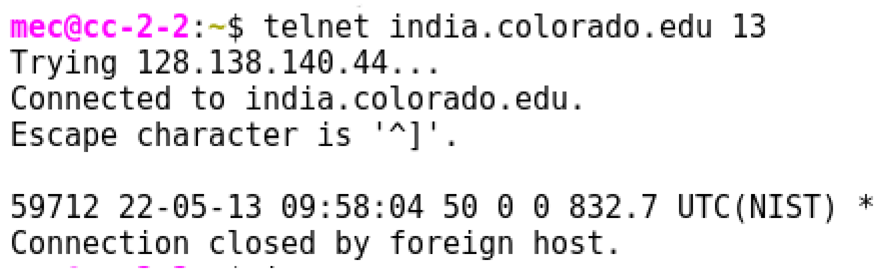
3. ping : It basically checks for the network connectivity between two nodes. ping stands for Packet INternet Groper.The ping command sends the ICMP echo request to check the network connectivity.
   Syntax - ping <destination>
   You can limit the number of packets by including "-c" in the ping command.
   Syntax - ping -c <number> <destination>

4. netstat : Linux netstat command stands for Network statistics. It displays information about different interface statistics, including open sockets, routing tables, and connection information.
   Syntax – netstat

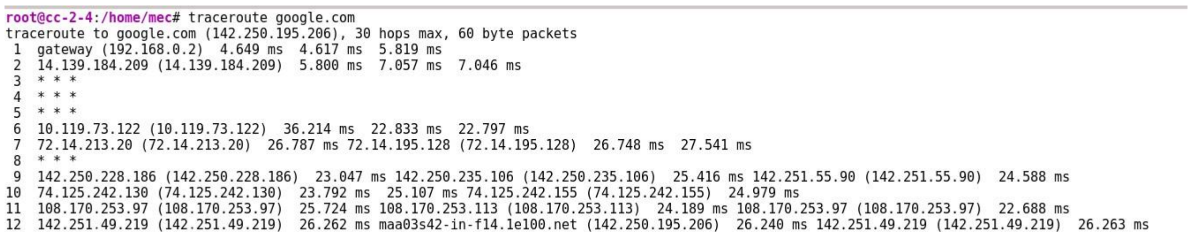
5. hostname : hostname command allows us to set and view the hostname of the system. A hostname is the name of any computer that is connected to a network that is uniquely identified over a network.
   Syntax – hostname



```
mec@cc-2-2:~$ ftp
ftp> help
Commands may be abbreviated.  Commands are:

!               dir             mdelete         qc              site
$               disconnect      mdir            sendport        size
account         exit            mget            put             status
append          form            mkdir           pwd             struct
ascii           get             mls             quit            system
bell            glob            mode            quote           sunique
binary          hash            modtime         recv            tenex
bye             help            mput            reget           tick
case            idle            newer           rstatus         trace
cd              image           nmap            rhelp           type
cdup            ipany           nlist           rename          user
chmod           ipv4            ntrans          reset           umask
close           ipv6            open            restart         verbose
cr              lcd             prompt          rmdir           ?
delete          ls              passive         runique
debug           macdef          proxy           send
ftp> status
Not connected.
No proxy connection.
Connecting using address family: any.
Mode: ; Type: ; Form: ; Structure:
Verbose: on; Bell: off; Prompting: on; Globbing: on
Store unique: off; Receive unique: off
Case: off; CR stripping: on
Quote control characters: on
Ntrans: off
Nmap: off
Hash mark printing: off; Use of PORT cmds: on
Tick counter printing: off
ftp> open
(to) localhost
ftp: connect to address ::1: Connection refused
Trying 127.0.0.1...
ftp: connect: Connection refused
ftp> close
Not connected.
ftp> exit
```



```
mec@cc-2-2:~$ telnet india.colorado.edu 13
Trying 128.138.140.44...
Connected to india.colorado.edu.
Escape character is '^]'.

59712 22-05-13 09:58:04 50 0 0 832.7 UTC(NIST) *
Connection closed by foreign host.
```



```
root@cc-2-4:/home/mec# traceroute google.com
traceroute to google.com (142.250.195.206), 30 hops max, 60 byte packets
 1  gateway (192.168.0.2)  4.649 ms  4.617 ms  5.819 ms
 2  14.139.184.209 (14.139.184.209)  5.800 ms  7.057 ms  7.046 ms
 3  * * *
 4  * * *
 5  * * *
 6  10.119.73.122 (10.119.73.122)  36.214 ms  22.833 ms  22.797 ms
 7  72.14.213.20 (72.14.213.20)  26.787 ms 72.14.195.128 (72.14.195.128)  26.748 ms  27.541 ms
 8  * * *
 9  142.250.228.186 (142.250.228.186)  23.047 ms 142.250.235.106 (142.250.235.106)  25.416 ms 142.251.55.90 (142.251.55.90)  24.588 ms
10  74.125.242.130 (74.125.242.130)  23.792 ms  25.107 ms 74.125.242.155 (74.125.242.155)  24.979 ms
11  108.170.253.97 (108.170.253.97)  25.724 ms 108.170.253.113 (108.170.253.113)  24.189 ms 108.170.253.97 (108.170.253.97)  22.688 ms
12  142.251.49.219 (142.251.49.219)  26.262 ms maa03s42-in-f14.1e100.net (142.250.195.206)  26.240 ms 142.251.49.219 (142.251.49.219)  26.263 ms
```

6. ifconfig : Linux ifconfig stands for interface configurator. It is one of the most basic commands used in network inspection. ifconfig is used to initialize an interface, configure it with an IP address, and enable or disable it. It is also used to display the route and the network interface.
   Syntax – ifconfig

7. arp : Linux arp command stands for Address Resolution Protocol. It is used to view and add content to the kernel's ARP table.All the systems maintain a table of IP addresses and their corresponding MAC addresses. This table is called the ARP Lookup table.
   Syntax – arp

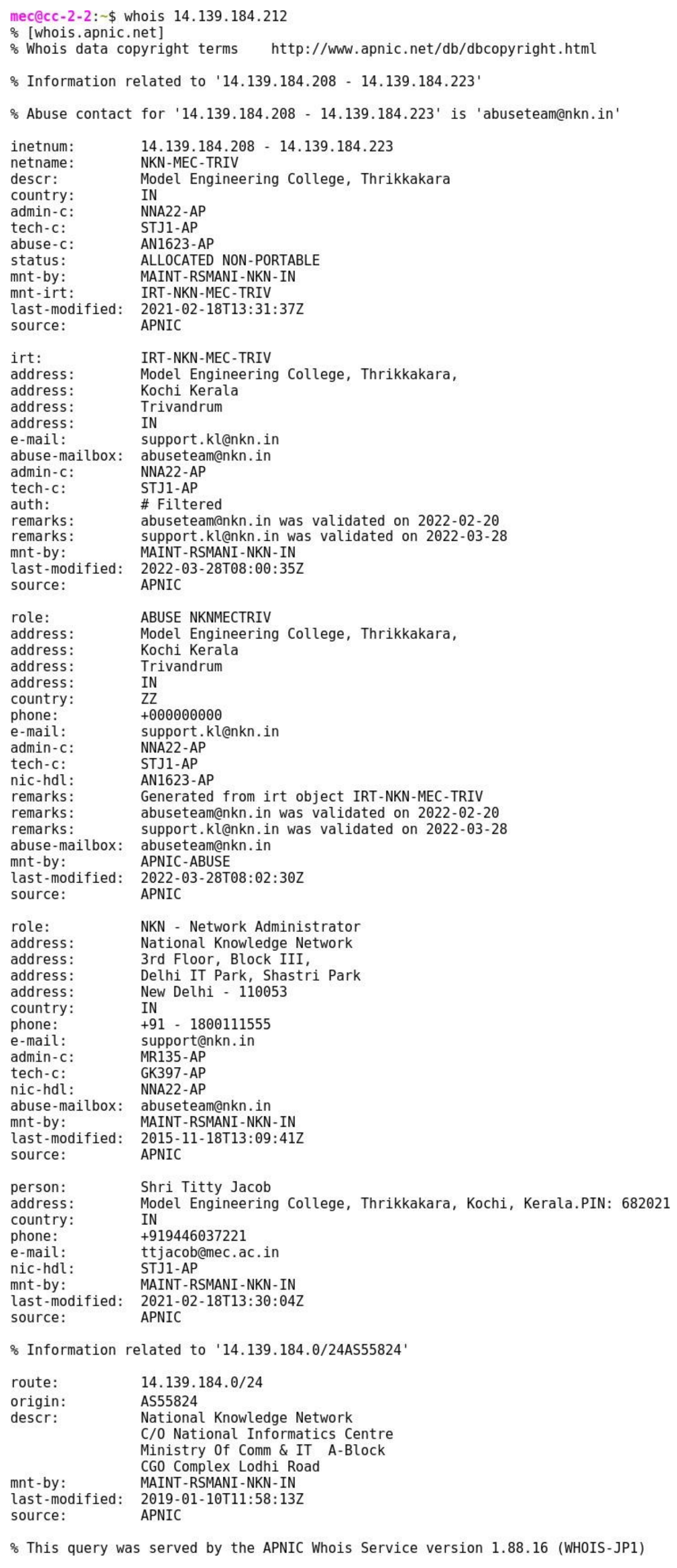
8. whois : Linux whois command is used to fetch all the information related to a website. You can get all the information about a website including the registration and the owner information.
   Syntax - whois <websiteName>

9. nslookup : nslookup (stands for "Name Server Lookup") is a useful command for getting information from the DNS server. It is a network administration tool for querying the Domain Name System (DNS) to obtain domain name or IP address mapping or any other specific DNS record. It is also used to troubleshoot DNS-related problems.
   Syntax - nslookup <domainName>

10. ftp : FTP (File Transfer Protocol) is a network protocol used for transferring files from one computer system to another. Even though the safety of FTP tends to spark a lot of discussion, it is still an effective method of transferring files within a secure network.
    Syntax - ftp [options] [IP address]

11. telnet : In Linux, the telnet command is used to create a remote connection with a system over a TCP/IP network. It allows us to administrate other systems by the terminal. We can run a program to conduct administration.
    Syntax - telnet hostname/IP address

12. finger : Finger command is a user information lookup command which gives details of all the users logged in. This tool is generally used by system administrators. It provides details like login name, user name, idle time, login time, and in some cases their email address even.
    Syntax - finger [option] [username]



```
mec@cc-2-2:~$ whois 14.139.184.212
% [whois.apnic.net]
% Whois data copyright terms    http://www.apnic.net/db/dbcopyright.html

% Information related to '14.139.184.208 - 14.139.184.223'

% Abuse contact for '14.139.184.208 - 14.139.184.223' is 'abuseteam@nkn.in'

inetnum:        14.139.184.208 - 14.139.184.223
netname:        NKN-MEC-TRIV
descr:          Model Engineering College, Thrikkakara
country:        IN
admin-c:        NNA22-AP
tech-c:         STJ1-AP
abuse-c:        AN1623-AP
status:         ALLOCATED NON-PORTABLE
mnt-by:         MAINT-RSMANI-NKN-IN
mnt-irt:        IRT-NKN-MEC-TRIV
last-modified:  2021-02-18T13:31:37Z
source:         APNIC

irt:            IRT-NKN-MEC-TRIV
address:        Model Engineering College, Thrikkakara,
address:        Kochi Kerala
address:        Trivandrum
address:        IN
e-mail:         support.kl@nkn.in
abuse-mailbox:  abuseteam@nkn.in
admin-c:        NNA22-AP
tech-c:         STJ1-AP
auth:           # Filtered
remarks:        abuseteam@nkn.in was validated on 2022-02-20
remarks:        support.kl@nkn.in was validated on 2022-03-28
mnt-by:         MAINT-RSMANI-NKN-IN
last-modified:  2022-03-28T08:00:35Z
source:         APNIC

role:           ABUSE NKNMECTRIV
address:        Model Engineering College, Thrikkakara,
address:        Kochi Kerala
address:        Trivandrum
address:        IN
country:        ZZ
phone:          +000000000
e-mail:         support.kl@nkn.in
admin-c:        NNA22-AP
tech-c:         STJ1-AP
nic-hdl:        AN1623-AP
remarks:        Generated from irt object IRT-NKN-MEC-TRIV
remarks:        abuseteam@nkn.in was validated on 2022-02-20
remarks:        support.kl@nkn.in was validated on 2022-03-28
abuse-mailbox:  abuseteam@nkn.in
mnt-by:         APNIC-ABUSE
last-modified:  2022-03-28T08:02:30Z
source:         APNIC

role:           NKN - Network Administrator
address:        National Knowledge Network
address:        3rd Floor, Block III,
address:        Delhi IT Park, Shastri Park
address:        New Delhi - 110053
country:        IN
phone:          +91 - 1800111555
e-mail:         support@nkn.in
admin-c:        MR135-AP
tech-c:         GK397-AP
nic-hdl:        NNA22-AP
abuse-mailbox:  abuseteam@nkn.in
mnt-by:         MAINT-RSMANI-NKN-IN
last-modified:  2015-11-18T13:09:41Z
source:         APNIC

person:         Shri Titty Jacob
address:        Model Engineering College, Thrikkakara, Kochi, Kerala.PIN: 682021
country:        IN
phone:          +919446037221
e-mail:         ttjacob@mec.ac.in
nic-hdl:        STJ1-AP
mnt-by:         MAINT-RSMANI-NKN-IN
last-modified:  2021-02-18T13:30:04Z
source:         APNIC

% Information related to '14.139.184.0/24AS55824'

route:          14.139.184.0/24
origin:         AS55824
descr:          National Knowledge Network
                C/O National Informatics Centre
                Ministry Of Comm & IT  A-Block
                CGO Complex Lodhi Road
mnt-by:         MAINT-RSMANI-NKN-IN
last-modified:  2019-01-10T11:58:13Z
source:         APNIC

% This query was served by the APNIC Whois Service version 1.88.16 (WHOIS-JP1)
```

Network Configuration Files

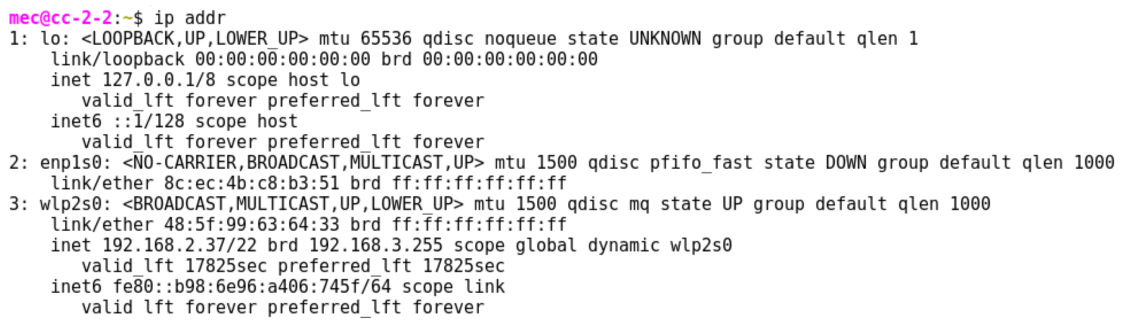
To store IP addresses and other related settings, Linux uses a separate configuration file for each network interface. All these configuration files are stored in the /etc/sysconfig/network-scripts directory.The important linux network configuration files are:

1. /etc/hosts : This file is used to map the hostname with IP address. Once hostname and IP address are mapped, hostname can be used to access the services available on the destination IP address. A hostname can be mapped with an IP address in two ways through the DNS server and through the /etc/hosts file.

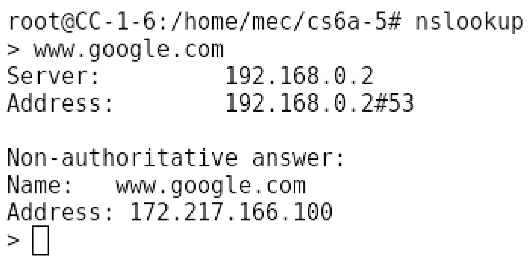
2. /etc/resolv.conf : The /etc/resolv.conf configuration file specifies the IP addresses of DNS servers and the search domain.

3. /etc/sysconfig/network : This file specifies routing and host information for all network interfaces.

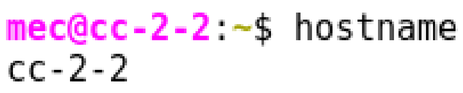
4. /etc/nsswitch.conf : The "/etc/nsswitch.conf" file contains your settings as to how various system lookups are carried out. One of the main functions of the "nsswitch.conf is to control how your network is resolved



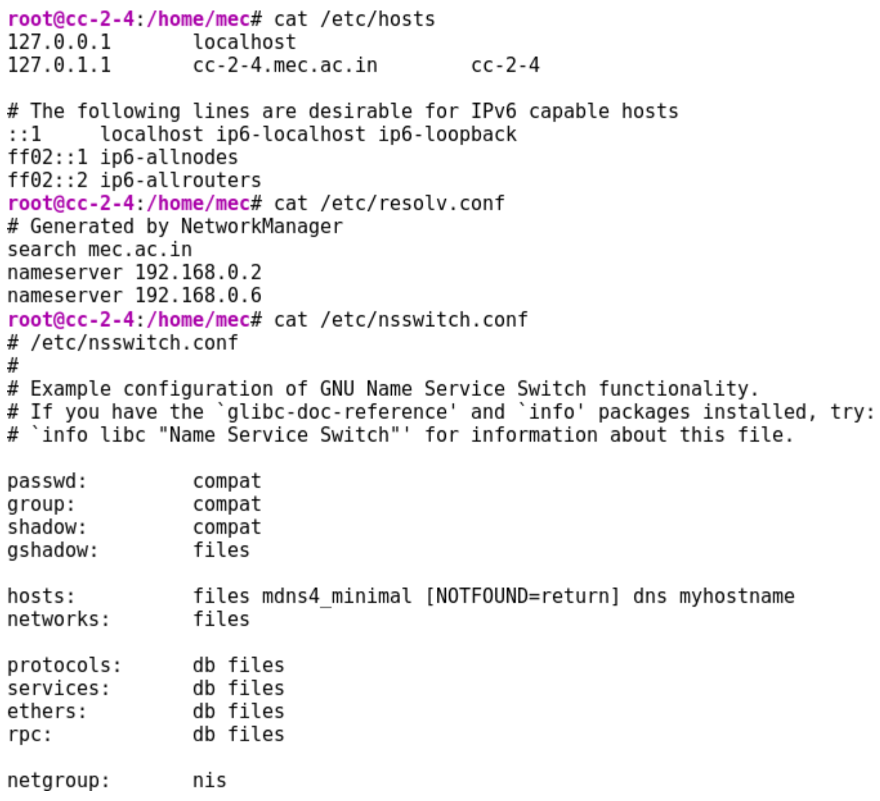
```
mec@cc-2-2:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp1s0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN group default qlen 1000
    link/ether 8c:ec:4b:c8:b3:51 brd ff:ff:ff:ff:ff:ff
3: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 48:5f:99:63:64:33 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.37/22 brd 192.168.3.255 scope global dynamic wlp2s0
       valid_lft 17825sec preferred_lft 17825sec
    inet6 fe80::b98:6e96:a406:745f/64 scope link
       valid_lft forever preferred_lft forever
```



```
root@CC-1-6:/home/mec/cs6a-5# nslookup
> www.google.com
Server:         192.168.0.2
Address:        192.168.0.2#53

Non-authoritative answer:
Name:   www.google.com
Address: 172.217.166.100
> 
```



```
mec@cc-2-2:~$ hostname
cc-2-2
```



```
root@cc-2-4:/home/mec# cat /etc/hosts
127.0.0.1       localhost
127.0.1.1       cc-2-4.mec.ac.in        cc-2-4

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
root@cc-2-4:/home/mec# cat /etc/resolv.conf
# Generated by NetworkManager
search mec.ac.in
nameserver 192.168.0.2
nameserver 192.168.0.6
root@cc-2-4:/home/mec# cat /etc/nsswitch.conf
# /etc/nsswitch.conf
#
# Example configuration of GNU Name Service Switch functionality.
# If you have the `glibc-doc-reference' and `info' packages installed, try:
# `info libc "Name Service Switch"' for information about this file.

passwd:         compat
group:          compat
shadow:         compat
gshadow:        files

hosts:          files mdns4_minimal [NOTFOUND=return] dns myhostname
networks:       files

protocols:      db files
services:       db files
ethers:         db files
rpc:            db files

netgroup:       nis
```

RESULT:

Basics of network configuration files and networking commands in Linux were understood.

Date : 28/02/2023

Experiment No : 2
Socket Programming

AIM :

To familiarize and understand the use and functioning of system calls used for network programming in Linux

THEORY :

1. socket() :

   The socket system call creates a new socket by assigning a new descriptor. Any subsequent system calls are identified with the created socket.
   Syntax - int socket(int domain, int type, int protocol);

2. bind() :

   The bind system call associates a local network transport address with a socket. For a client process, it is not mandatory to issue a bind call. It is often necessary for a server process to issue an explicit bind request before it can accept connections or start communication with clients.
   Syntax - int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

3. connect() :

   The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. The addrlen argument specifies the size of addr. The connect system call is normally called by the client process to connect to the server process.
   Syntax - int connect(int sockfd, const struct sockaddr *addr,socklen_t addrlen);

4. listen() :

   isten() marks the socket referred to by sockfd as a passive socket, that is, as a socket that will be used to accept incoming connection requests using accept().There is a limit on the number of connections that can be queued up, after which any further connection requests are ignored.
   Syntax - int listen(int sockfd, int backlog);

5. accept() :

   The accept system call is a blocking call that waits for incoming connections. Once a connection request is processed, a new socket descriptor is returned by accept. This new socket is connected to the client and the other sockets remain in LISTEN state to accept further connections.

Syntax - int accept(int sockfd, struct sockaddr *restrict addr, socklen_t *restrict addrlen);

6. send()/sendto() :

   These system calls are used to send messages or data.send() is used in connection oriented protocols while sendto() is used in connection-less protocols.

   Syntax - send(int sockfd, const void *buf, size_t len, int flags);
   sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen);

7. recv()/recvfrom() :

   These system calls are used to send messages or data.recv() is used in connection oriented protocols while recvfrom() is used in connection-less protocols.

   Syntax - recv(int sockfd, void *buf, size_t len, int flags);
   recvfrom(int sockfd, void *restrict buf, size_t len, int flags, struct sockaddr *restrict src_addr, socklen_t *restrict addrlen);

8. close() :

   Sockets need to be closed after they are not being used anymore. Its only argument is the socket file descriptor and it returns 0 once it's successfully closed.

   Syntax - int close(int fd);

RESULT:

The use of system calls used for networking programming in Linux was familiarised and understood.

PROGRAM

Server

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main()
{
	char buf[100];
	int k;
	socklen_t len;
	int sock_desc, temp_sock_desc;
	struct sockaddr_in server, client;

	sock_desc = socket(AF_INET, SOCK_STREAM, 0);
	if (sock_desc == -1)
		printf("Error in socketcreation");
	server.sin_family = AF_INET;
	server.sin_addr.s_addr = INADDR_ANY;
	server.sin_port  =  3003;
	client.sin_family = AF_INET;
	client.sin_addr.s_addr = INADDR_ANY;
	client.sin_port = 3003;

	k = bind(sock_desc, (struct sockaddr *)&server, sizeof(server));
	if (k == -1)
		printf("Error in binding");

	k = listen(sock_desc, 5);
	if (k == -1)
		printf("Error in listening");

	len = sizeof(client);
	temp_sock_desc = accept(sock_desc, (struct sockaddr *)&client,&len);
	if (temp_sock_desc == -1)
		printf("Error in temporary socket creation");

	k = recv(temp_sock_desc, buf, 100, 0);
	if (k == -1)
		printf("Error in receiving");

	printf("Message got from client: %s", buf);
	close(temp_sock_desc);
	return 0;
}
```

Date : 07/03/2023

Experiment No : 3
Transmission Control Protocol

AIM :

To implement client-server communication using socket programming and TCP as transport layer protocol

THEORY :

A TCP (transmission control protocol) is a connection-oriented communication. TCP is designed to send the data packets over the network. It ensures that data is delivered to the correct destination.TCP creates a connection between the source and destination node before transmitting the data and keeps the connection alive until the communication is active.

Server-client model is communication model for sharing the resource and provides the service to different machines. Server is the main system which provides the resources and different kind of services when client requests to use it.

Client
```
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
	char buf[100];
	int k;
	int sock_desc;
	struct sockaddr_in client;

	sock_desc = socket(AF_INET, SOCK_STREAM, 0);
	if (sock_desc == -1)
		printf("Error in socket creation!");

	client.sin_family = AF_INET;
	client.sin_addr.s_addr = INADDR_ANY;
	client.sin_port = 3003;

	k = connect(sock_desc, (struct sockaddr *)&client, sizeof(client));
	if (k == -1)
		printf("Error in connecting to server!");

	printf("\nEnter data to be send: ");
	fgets(buf, 100, stdin);

	k = send(sock_desc, buf, 100, 0);
	if (k == -1)
		printf("Error in sending!");

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

OUTPUT:

```
// Terminal 1
$ gcc -o server server.c
$ ./server
_
// Terminal 2
$ gcc -o client client.c
$ ./client
Enter data to be send: hi programmers!

// Terminal 1
$ gcc -o server server.c
$ ./server
Message got from client: hi programmers!
```

ALGORITHM:

Server :

1. Create a socket with type as SOCK_STREAM to create a tcp socket using socket() system call.
2. Bind the socket to a specific port using bind() system call.
3. Listen for new connections using the listen() system call.
4. Accept connection from client process into a temporary socket.
5. Read the message from the client using the recv() system call into a buffer.
6. Display the message and close the socket.

Client :

1. Create a socket with type as SOCK_STREAM to create a tcp socket using socket() system call.
2. Connect to the server using connect() system call.
3. Read the message to be sent from the user.
4. Send the message to the server using send() system call.
5. Close the socket.

RESULT:

The programs for client-server communication using socket programming for TCP was executed and output was verified successfully.

PROGRAM

Server

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
      struct sockaddr_in server, client;
      if (argc != 2)
            printf("Input format not correct!\n");

      int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
      if (sockfd == -1)
            printf("Error in socket creation!\n");
      server.sin_family = AF_INET;
      server.sin_addr.s_addr = INADDR_ANY;
      server.sin_port = htons(atoi(argv[1]));

      if (bind(sockfd, (struct sockaddr *)&server, sizeof(server)) < 0)
            printf("Error in bind()!\n");
      char buffer[100];
      socklen_t server_len = sizeof(server);
      printf("Server waiting...\n");

      if (recvfrom(sockfd, buffer, 100, 0, (struct sockaddr *)&server,
&server_len)< 0)
            printf("Error in receiving!\n");
      printf("Got a datagram: %s", buffer);
      return 0;
}
```

Date : 07/03/2023

Experiment No : 4
User Datagram Protocol

AIM :

To implement client-server communication using socket programming and UDP as transport layer protocol.

THEORY :

UDP is a connection-less protocol that, unlike TCP, does not require any handshaking prior to sending or receiving data, which simplifies its implementation. In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram. Similarly, the server need not accept a connection and just waits for datagrams to arrive.

Server-client model is communication model for sharing the resource and provides the service to different machines. Server is the main system which provides the resources and different kind of services when client requests to use it.

Client

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
      struct sockaddr_in server, client;
      if (argc != 3)
            printf("Input format not correct!\n");

      int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
      if (sockfd == -1)
            printf("Error in socket creation!\n");
      server.sin_family = AF_INET;
      server.sin_addr.s_addr = INADDR_ANY;
      server.sin_port = htons(atoi(argv[2]));



      char buffer[100];
      printf("Enter a message to sent to server: ");

      fgets(buffer, 100, stdin);
      if (sendto(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr *)&server,
sizeof(server)) < 0)
            printf("Error in sending!\n");
      return 0;
}
```

OUTPUT

```
// Terminal 1
$ gcc -o server server.c
$ ./server 2000
Server waiting...

// Terminal 2
$ gcc -o client client.c
$ ./client localhost 2000
Enter a message to sent to server: Morning!

// Terminal 1
$ gcc -o server server.c
$ ./server 2000
Server waiting...
Got a datagram: Morning!
```

ALGORITHM:
Server :

1. Create a socket with type as SOCK_DGRAM to create a udp socket using socket() system call.
2. Bind the socket to a specific port using bind() system call.
3. Using the recvfrom() system call message sent from the client process into a buffer.
4. Display the message and close the socket.

Client :

1. Create a socket with type as SOCK_DGRAM to create a udp socket using socket() system call.
2. Read the message to be sent from the user.
3. Send the message to the server using sendto() system call.
4. Close the socket.

RESULT:

The programs for client-server communication using socket programming for UDP was executed and output was verified successfully.

PROGRAM

Server

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <unistd.h>
#include <arpa/inet.h>
typedef struct packet{
      char data[1024];
}Packet;
typedef struct frame{
      int frame_kind; //ACK:0, SEQ:1 FIN:2
      int sq_no;
      int ack;
      Packet packet;
}Frame;
int main(int argc, char** argv){
      if (argc != 2){
            printf("Usage: %s <port>", argv[0]);
            exit(0);
      }
      int port = atoi(argv[1]);
      int sockfd;
      struct sockaddr_in serverAddr, newAddr;
      char buffer[1024];
      socklen_t addr_size;
      int  frame_id=0;
      Frame frame_recv;
      Frame frame_send;
      sockfd = socket(AF_INET, SOCK_DGRAM, 0);
      memset(&serverAddr, '\0', sizeof(serverAddr));
      serverAddr.sin_family = AF_INET;
      serverAddr.sin_port = htons(port);
      serverAddr.sin_addr.s_addr  =  inet_addr("127.0.0.1");
      bind(sockfd, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
      addr_size = sizeof(newAddr);
      while(1){
            int f_recv_size = recvfrom(sockfd, &frame_recv,sizeof(Frame),0,
(struct sockaddr*)&newAddr, &addr_size);
            if (f_recv_size > 0 && frame_recv.frame_kind == 1 &&
frame_recv.sq_no == frame_id){
                  printf("[+]Frame Received: %s\n",
                  frame_recv.packet.data);
                  frame_send.sq_no = 0;
                  frame_send.frame_kind = 0;
                  frame_send.ack = frame_recv.sq_no + 1;
                  sendto(sockfd, &frame_send, sizeof(frame_send), 0,(struct
sockaddr*)&newAddr, addr_size);
                  printf("[+]Ack Send\n");
            }else{
printf("[+]Frame Not Received\n"); }
frame_id++;
      }
}
```

Date : 23/03/2023

Experiment No : 5
Stop and Wait Protocol

AIM :

To write a program to simulate the stop and wait protocol.

THEORY :

Sliding window protocols are data link layer protocols for reliable and sequential delivery of data frames. In this protocol, multiple frames can be sent by a sender at a time before receiving an acknowledgment from the receiver. The term sliding window refers to the imaginary boxes to hold frames.

Stop-and-Wait is the simplest flow control method. In this, the sender will transmit one frame at a time to the receiver. The sender will stop and wait for the acknowledgement from the receiver. When the sender gets the acknowledgement (ACK), it will send the next data packet to the receiver and wait for the disclosure again, and this process will continue as long as the sender has the data to send. The sender and receiver window size is 1.

ALGORITHM:

Stop and Wait Server :

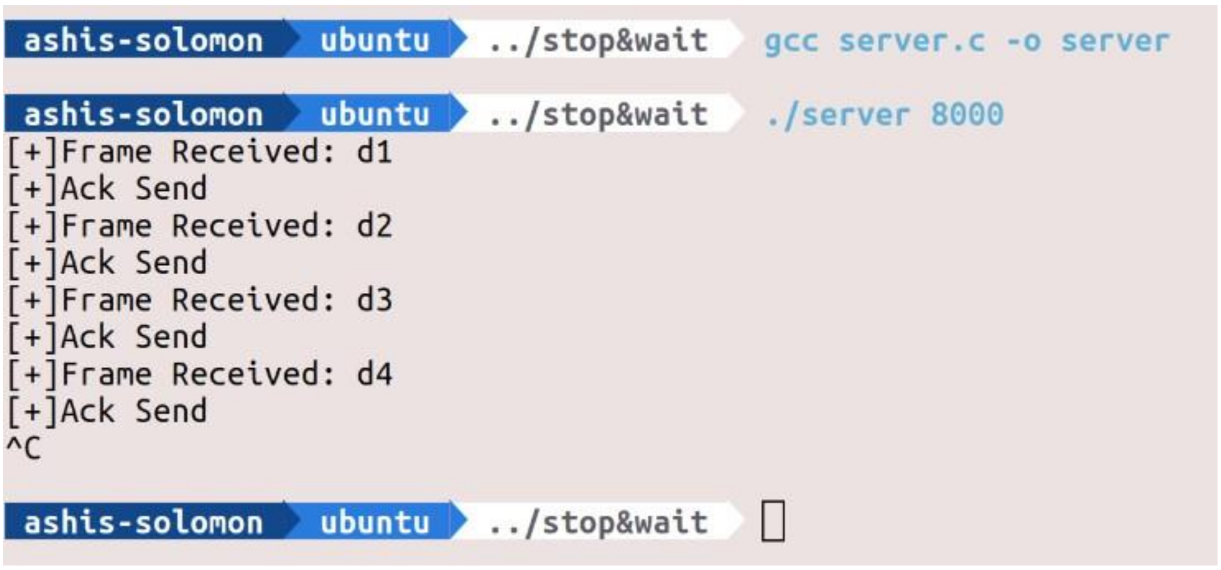
1. Create a socket using the socket() function.
2. Bind the socket to a specific port using the bind() function.
3. Create a loop to continuously receive frames from the client.
4. Receive the frame using the recvfrom() function.
5. If the received frame's frame_kind is 1 (data frame) and the sq_no matches the expected frame_id:
   a) Print the received data from the packet.
   b) Create an acknowledgment frame (frame_send) with frame_kind = 0 (acknowledgment frame), sq_no = 0, and ack = frame_recv.sq_no + 1.
   c) Send the acknowledgment frame to the client using the sendto() function.
6. If the received frame is not as expected (frame_kind is not 1 or sq_no doesn't match):
   a) Print a message indicating that the frame was not received properly.
7. Increment the frame_id to expect the next frame.
8. Repeat steps 4 to 7 for subsequent frames.
9. Close the socket using the close() function.

Client

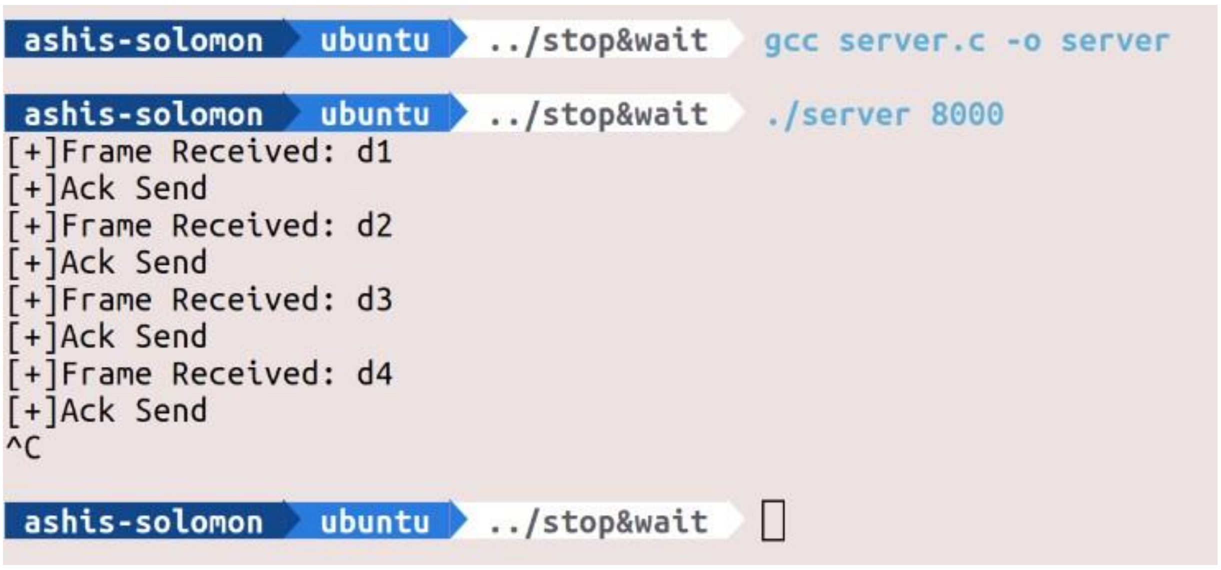
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
typedef struct packet{
	char data[1024];
}Packet;
typedef struct frame{
	int frame_kind; //ACK:0, SEQ:1 FIN:2
	int sq_no;
	int ack;
	Packet packet;
}Frame;

int main(int argc, char *argv[]){
	if (argc != 2){
		printf("Usage: %s <port>", argv[0]);
		exit(0);
	}
	int port = atoi(argv[1]);
	int sockfd;
	struct sockaddr_in serverAddr;
	char buffer[1024];
	socklen_t addr_size;
	int frame_id = 0;
	Frame frame_send;
	Frame  frame_recv;
	int ack_recv = 1;
	sockfd = socket(AF_INET, SOCK_DGRAM, 0);
	memset(&serverAddr, '\0', sizeof(serverAddr));
	serverAddr.sin_family = AF_INET;
	serverAddr.sin_port = htons(port);
	serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
	while(1){
		if(ack_recv == 1){
			frame_send.sq_no = frame_id;
			frame_send.frame_kind = 1;
			frame_send.ack = 0;
			printf("Enter Data: ");
			scanf("%s", buffer);
			strcpy(frame_send.packet.data,  buffer);
			sendto(sockfd, &frame_send, sizeof(Frame), 0, (struct
sockaddr*)&serverAddr, sizeof(serverAddr));
			printf("[+]Frame Send\n");
		}
	int addr_size = sizeof(serverAddr);
	int f_recv_size = recvfrom(sockfd, &frame_recv,
	sizeof(frame_recv), 0 ,(struct sockaddr*)&serverAddr, &addr_size);
	if( f_recv_size > 0 && frame_recv.sq_no == 0 && frame_recv.ack ==
frame_id+1){
			printf("[+]Ack Received\n");
			ack_recv = 1;
		}else{
			printf("[-]Ack Not Received\n");
			ack_recv = 0;
		}
	}
}
```

Stop and Wait Client **:**

1. Create a socket using the socket() function.
2. Initialize the server address and port.
3. Create a loop to continuously send frames to the server.
4. If the previous acknowledgment (ack_recv) is 1 (indicating the previous frame was successfully acknowledged):
   a) Create a data frame (frame_send) with frame_kind = 1, sq_no = frame_id, ack = 0.
   b) Prompt the user to enter the data to be sent and store it in the frame_send's packet data field.
   c) Send the data frame to the server using the sendto() function.
5. Receive the acknowledgment frame from the server using the recvfrom() function.
6. If the acknowledgment frame's sq_no is 0 and ack matches frame_id + 1:
   a) Print a message indicating that the acknowledgment was received.
   b) Set ack_recv to 1 to indicate successful acknowledgment.
7. If the acknowledgment frame is not as expected (sq_no is not 0 or ack doesn't match frame_id + 1):
   a) Print a message indicating that the acknowledgment was not received properly.
   b) Set ack_recv to 0 to indicate unsuccessful acknowledgment.
8. Increment the frame_id to prepare for the next frame.
9. Repeat steps 4 to 8 for subsequent frames.
10. Close the socket using the close() function.



```
ashis-solomon  ubuntu  ../stop&wait  gcc server.c -o server

ashis-solomon  ubuntu  ../stop&wait  ./server 8000
[+]Frame Received: d1
[+]Ack Send
[+]Frame Received: d2
[+]Ack Send
[+]Frame Received: d3
[+]Ack Send
[+]Frame Received: d4
[+]Ack Send
^C

ashis-solomon  ubuntu  ../stop&wait
```



```
ashis-solomon  ubuntu  ../stop&wait  gcc server.c -o server

ashis-solomon  ubuntu  ../stop&wait  ./server 8000
[+]Frame Received: d1
[+]Ack Send
[+]Frame Received: d2
[+]Ack Send
[+]Frame Received: d3
[+]Ack Send
[+]Frame Received: d4
[+]Ack Send
^C

ashis-solomon  ubuntu  ../stop&wait
```

RESULT:

The program for Stop and Wait protocol was executed and output was verified successfully.

PROGRAM

Server

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<sys/time.h>
#include<netinet/in.h>
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<fcntl.h>
void rsendd(int ch,int c_sock){
    char buff2[60];
    bzero(buff2,sizeof(buff2));
    strcpy(buff2,"reserver message :");
    buff2[strlen(buff2)]=(ch)+'0';
    buff2[strlen(buff2)]='\0';
    printf("Resending Message to client :%s \n",buff2);
    write(c_sock, buff2, sizeof(buff2));
    usleep(1000);
}
int main() {
    int s_sock, c_sock;
    s_sock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server, other;
    memset(&server, 0, sizeof(server));
    memset(&other, 0, sizeof(other));
    server.sin_family = AF_INET;
    server.sin_port = htons(9009);
    server.sin_addr.s_addr = INADDR_ANY;
    socklen_t add;
    f(bind(s_sock,(struct sockaddr*)&server, sizeof(server)) ==
-1){
        printf("Binding failed\n");
        return 0; }
    printf("\tServer Up\n Selective repeat scheme\n\n");
    listen(s_sock, 10);
    add = sizeof(other);
    c_sock = accept(s_sock, (struct sockaddr*)&other, &add);
    time_t t1,t2;
    char msg[50]="server message :";
    char buff[50];
    int flag=0;
    fd_set set1,set2,set3;
    struct timeval timeout1,timeout2,timeout3;
    int rv1,rv2,rv3;
    int tot=0;
    int ok[20];
    memset(ok,0,sizeof(ok));
    while(tot<9){
        int toti=tot;
        for(int j=(0+toti);j<(3+toti);j++){
            bzero(buff,sizeof(buff));
            char buff2[60];
            bzero(buff2,sizeof(buff2));
            strcpy(buff2,"server message :");
            buff2[strlen(buff2)]=(j)+'0';
            buff2[strlen(buff2)]='\0';
            printf("Message sent to client :%s \n",buff2,tot,j);
            write(c_sock, buff2, sizeof(buff2));
            usleep(1000);
        }
        for(int k=0+toti;k<(toti+3);k++){
            qq: FD_ZERO(&set1);
            FD_SET(c_sock, &set1);
            timeout1.tv_sec = 2;
            timeout1.tv_usec = 0;
            rv1 = select(c_sock + 1, &set1, NULL, NULL,
&timeout1);
            if(rv1 == -1)
                perror("select error ");
            else if(rv1 == 0){
                printf("Timeout for message :%d \n",k);
                rsendd(k,c_sock);
                goto qq;} // a timeout occured
            else{
                read(c_sock, buff, sizeof(buff));
                printf("Message from Client: %s\n", buff);
                if(buff[0]=='n'){
                    printf(" corrupt message awk (msg %d)\
n",buff[strlen(buff)-1]-'0');
                    rsendd((buff[strlen(buff)-1]-'0'),c_sock);
                    goto qq;}
                else
                    tot++;}
        }}
        close(c_sock);
        close(s_sock);
        return 0;
}
```

Client

```
#include<time.h>
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<sys/time.h>
#include<sys/wait.h>
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>

int isfaulty(){
int d=rand()%4;
    printf("%d\n",d);
    return (d>2);
}
int main() {
    srand(time(0));
    int c_sock;
    c_sock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in client;
    memset(&client, 0, sizeof(client));
    client.sin_family = AF_INET;
    client.sin_port = htons(9009);
    client.sin_addr.s_addr = inet_addr("127.0.0.1");
    if(connect(c_sock, (struct
sockaddr*)&client,sizeof(client)) == -1) {
        printf("Connection failed");
        return 0;
    }
    printf("\n\tClient awith individual acknowledgement
scheme\n\n");
    char msg1[50]="akwnowledgementof-";
    char msg3[50]="negative akwn-";
    char msg2[50];
    char buff[100];
    int count=-1,flag=1;
    while(count<8){
          bzero(buff,sizeof(buff));
         bzero(msg2,sizeof(msg2));
         if(count==7&&flag==1){
              printf("here\n");
              //simulate loss //i--;
              flag=0;
              read(c_sock,buff,sizeof(buff));
              continue;
         }
         int n = read(c_sock, buff, sizeof(buff));
         char i=buff[strlen(buff)-1];
         printf("Message received from server : %s \
n",buff);
         int isfault=isfaulty();
         printf("correption status : %d \n",isfault);
         printf("Response/akwn sent for message \n");
         if(isfault)
               strcpy(msg2,msg3);
         else{
               strcpy(msg2,msg1);
               count++;
          }
         msg2[strlen(msg2)]=i;
         write(c_sock,msg2, sizeof(msg2));
    }
    close(c_sock);
    return 0;
}
```

Date : 23/03/2023

Experiment No : 6
Selective Repeat

AIM :

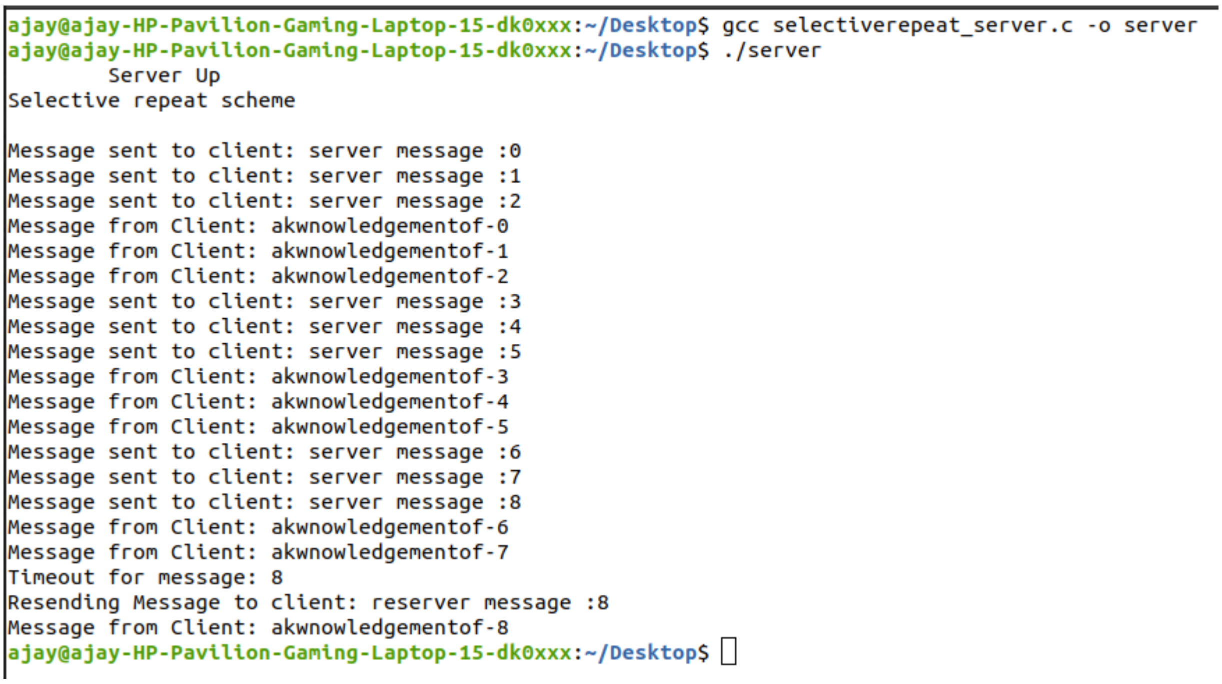
To write a program to simulate the Selective Repeat protocol

THEORY :

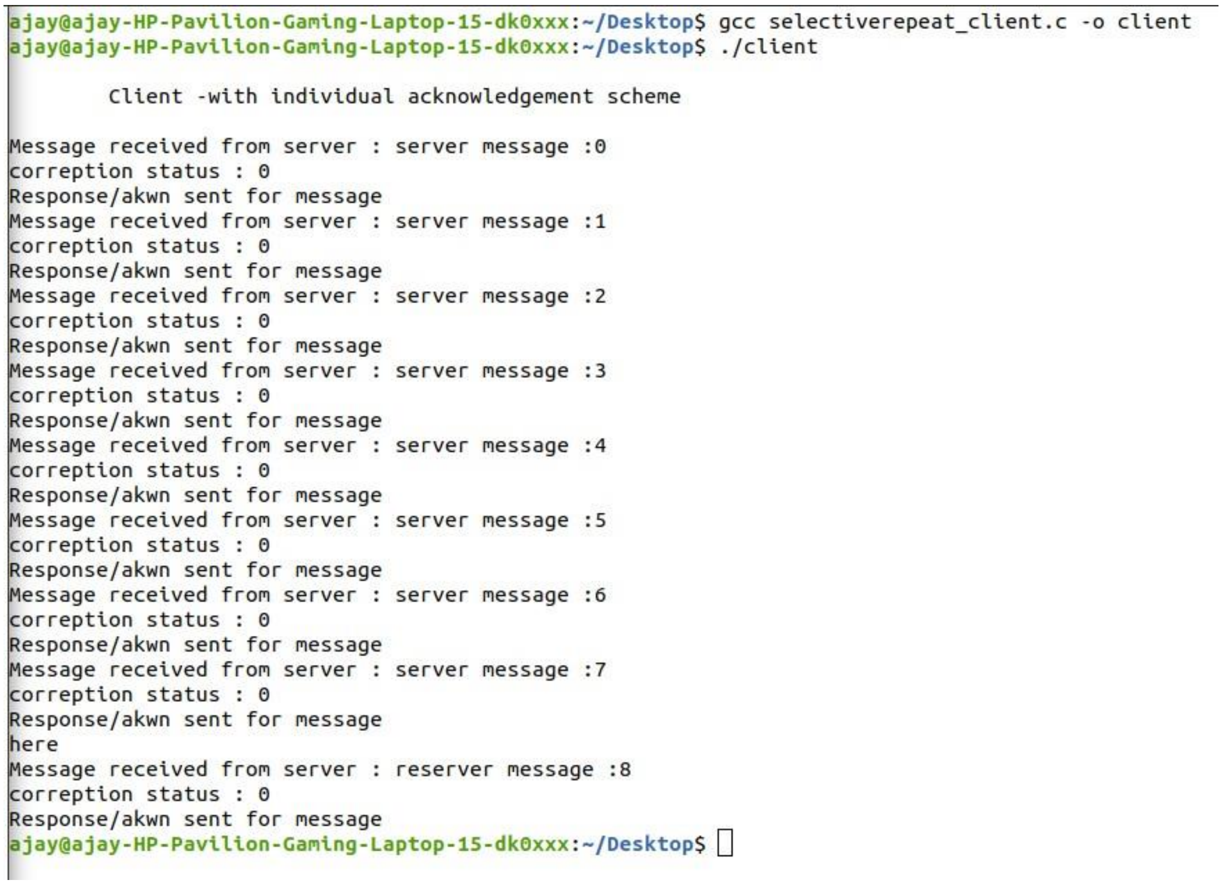
Reliable data transfer is a crucial aspect of computer networks as it ensures accurate and error-free transmission of data. In this context, the Selective Repeat Protocol plays a significant role. It is a variant of the Automatic Repeat Request (ARQ) protocol and aims to provide reliable data transfer over unreliable channels. The protocol addresses the challenges posed by unreliable channels by employing selective retransmission, where only the lost or corrupted frames are retransmitted, reducing unnecessary retransmissions and improving network efficiency.

Working of Selective Repeat Protocol:

The Selective Repeat Protocol involves two components: the sender and the receiver. The sender divides the data into manageable frames and assigns a unique sequence number to each frame. These frames are then transmitted to the receiver. The receiver acknowledges the correct receipt of frames using acknowledgment packets. If the receiver detects any errors or missing frames, it selectively requests retransmission of only the affected frames, eliminating the need for retransmitting the entire window. This selective retransmission feature enables efficient error recovery and inimizes network overhead. The protocol uses a window size to control the number of outstanding, unacknowledged frames, ensuring reliable transmission.



```
ajay@ajay-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Desktop$ gcc selectiverepeat_server.c -o server
ajay@ajay-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Desktop$ ./server
        Server Up
Selective repeat scheme

Message sent to client: server message :0
Message sent to client: server message :1
Message sent to client: server message :2
Message from Client: akwnowledgementof-0
Message from Client: akwnowledgementof-1
Message from Client: akwnowledgementof-2
Message sent to client: server message :3
Message sent to client: server message :4
Message sent to client: server message :5
Message from Client: akwnowledgementof-3
Message from Client: akwnowledgementof-4
Message from Client: akwnowledgementof-5
Message sent to client: server message :6
Message sent to client: server message :7
Message sent to client: server message :8
Message from Client: akwnowledgementof-6
Message from Client: akwnowledgementof-7
Timeout for message: 8
Resending Message to client: reserver message :8
Message from Client: akwnowledgementof-8
ajay@ajay-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Desktop$
```



```
ajay@ajay-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Desktop$ gcc selectiverepeat_client.c -o client
ajay@ajay-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Desktop$ ./client

        Client -with individual acknowledgement scheme

Message received from server : server message :0
correption status : 0
Response/akwn sent for message
Message received from server : server message :1
correption status : 0
Response/akwn sent for message
Message received from server : server message :2
correption status : 0
Response/akwn sent for message
Message received from server : server message :3
correption status : 0
Response/akwn sent for message
Message received from server : server message :4
correption status : 0
Response/akwn sent for message
Message received from server : server message :5
correption status : 0
Response/akwn sent for message
Message received from server : server message :6
correption status : 0
Response/akwn sent for message
Message received from server : server message :7
correption status : 0
Response/akwn sent for message
here
Message received from server : reserver message :8
correption status : 0
Response/akwn sent for message
ajay@ajay-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Desktop$
```

ALGORITHM:

Selective Repeat Server:

1. Create a socket using the socket() function.
2. Set up the server address and port.
3. Bind the socket to the server address using the bind() function.
4. Listen for incoming connections using the listen() function.
5. Accept a client connection using the accept() function.
6. Initialize necessary variables and data structures.
7. Enter a loop until all messages are successfully received and acknowledged:
    a. Send a batch of messages (e.g., 3 messages) to the client.
    b. For each message sent, wait for an acknowledgment from the client using the select() function with a timeout.
    c. If a timeout occurs, re-send the message by calling the rsendd() function.
    d. If an acknowledgment is received, check for corruption. If the message is corrupted, re-send the message. e. If the message is not corrupted, increment the total count.
8. Close the client and server sockets

Selective Repeat Client:

1. Create a socket using the socket() function.
2. Set up the server address and port.
3. Connect to the server using the connect() function.
4. Initialize necessary variables and data structures.
5. Enter a loop until all messages are received and acknowledged:
    a. Receive a message from the server.
    b. Simulate corruption of the message using the isfaulty() function.
    c. Send an acknowledgment message to the server indicating whether the message is acknowledged or not.
    d. If the message is not acknowledged due to corruption, the server will resend the message. e. If the message is acknowledged, increment the count.
6. Close the client socket.

RESULT :

The program for Selective Repeat was executed and output was verified successfully.

PROGRAM:

Server
```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <fcntl.h>
int main()
{
      int s_sock, c_sock;
      s_sock = socket(AF_INET, SOCK_STREAM, 0);
      struct sockaddr_in server, other;
      memset(&server, 0, sizeof(server));
      memset(&other, 0, sizeof(other));
      server.sin_family = AF_INET;
      server.sin_port = htons(9009);
      server.sin_addr.s_addr = INADDR_ANY;
      socklen_t add;
      if (bind(s_sock, (struct sockaddr *)&server, sizeof(server)) == -1){
            printf("Binding failed\n");
            return 0;
      }
      printf("\tServer Up\n Go back n (n=3) used to send 10 messages \n\n");
      listen(s_sock, 10);
      add = sizeof(other);
      c_sock = accept(s_sock, (struct sockaddr *)&other, &add);
      time_t t1, t2;
      char msg[50] = "server message :";
      char buff[50];
      int flag = 0;
      fd_set set1, set2, set3;
      struct timeval timeout1, timeout2, timeout3;
      int rv1, rv2, rv3;
      int i = -1;
      qq:
      i = i + 1;
      bzero(buff, sizeof(buff));
      char buff2[60];
      bzero(buff2, sizeof(buff2));
      strcpy(buff2, "server message :");
      buff2[strlen(buff2)] = i + '0';
      buff2[strlen(buff2)] = '\0';
      printf("Message sent to client :%s \n", buff2);
      write(c_sock, buff2, sizeof(buff2));
      usleep(1000);
      i = i + 1;

      bzero(buff2, sizeof(buff2));
      strcpy(buff2, msg);
      buff2[strlen(msg)] = i + '0';
      printf("Message sent to client :%s \n", buff2);
      write(c_sock, buff2, sizeof(buff2));
      i = i + 1;
      usleep(1000);
```

Date : 11/04/2023

Experiment No : 7
Go Back N

AIM :

To write a C program to simulate the Go Back N.

THEORY :

Sliding window protocols are data link layer protocols for reliable and sequential delivery of data frames. In this protocol, multiple frames can be sent by a sender at a time before receiving an acknowledgment from the receiver. The term sliding window refers to the imaginary boxes to hold frames. Go-Back-N ARQ protocol is also known as Go-Back-N Automatic Repeat Request. In this, if any frame is corrupted or lost, all subsequent frames have to be sent again. The size of the sender window is N in this protocol. The receiver window size is always 1. If the receiver receives a corrupted frame, it cancels it. The receiver does not accept a corrupted frame. When the timer expires, the sender sends the correct frame again.

```
        qqq:
        bzero(buff2, sizeof(buff2));
        strcpy(buff2, msg);
        buff2[strlen(msg)] = i + '0';
        printf("Message sent to client :%s \n", buff2);
        write(c_sock, buff2, sizeof(buff2));
        FD_ZERO(&set1);
        FD_SET(c_sock, &set1);
        timeout1.tv_sec = 2;
        timeout1.tv_usec = 0;
        rv1 = select(c_sock + 1, &set1, NULL, NULL, &timeout1);
        if (rv1 == -1)
                perror("select error ");
        else if (rv1 == 0){
                printf("Going back from %d:timeout \n", i);
                i = i - 3;
                goto qq;
        }
        else{
                read(c_sock, buff, sizeof(buff));
                printf("Message from Client: %s\n", buff);
                i++;
                if (i <= 9)
                goto qqq;
        }
        qq2:
        FD_ZERO(&set2);
        FD_SET(c_sock, &set2);
        timeout2.tv_sec = 3;
        timeout2.tv_usec = 0;
        rv2 = select(c_sock + 1, &set2, NULL, NULL, &timeout2);
        if (rv2 == -1)
                perror("select error "); // an error accured
        else if (rv2 == 0){
                printf("Going back from %d:timeout on last 2\n", i - 1);
                i = i - 2;
                bzero(buff2, sizeof(buff2));
                strcpy(buff2, msg);
                buff2[strlen(buff2)] = i + '0';
                write(c_sock, buff2, sizeof(buff2));
                usleep(1000);
                bzero(buff2, sizeof(buff2));
                i++;
                strcpy(buff2, msg);
                buff2[strlen(buff2)] = i + '0';
                write(c_sock, buff2, sizeof(buff2));
                goto qq2;
        } // a timeout occured
        else{
                read(c_sock, buff, sizeof(buff));
                printf("Message from Client: %s\n", buff);
                bzero(buff, sizeof(buff));
                read(c_sock, buff, sizeof(buff));
                printf("Message from Client: %s\n", buff);
        }
        close(c_sock);
        close(s_sock);
        return 0;
}
```

ALGORITHM:

Go Back N Server:

1. Create a client socket using the socket() function with domain AF_INET,type SOCK_STREAM, and protocol 0.
2. Initialize the client address structure (client) with zeros using memset().
3. Set the address family (AF_INET), port number (htons(9009)), and IP address (inet_addr("127.0.0.1")) in the client structure.
4. Connect the client socket to the server using the connect() function and the client structure as the socket address. If the connection fails, print an error message and exit.
5. Print a message indicating that the client is ready.
6. Initialize message buffers: msg1 for the acknowledgement message, msg2 for constructing acknowledgement messages, and buff for receiving data from the server.
7. Initialize flags: flag and flg.
8. Start a loop from 0 to 9 to receive messages and send acknowledgements.
    a. Clear the buff and msg2 buffers.
    b. If i is 8 and flag is 1, simulate a loss by reading from the server without storing the data.
    c. Read the message from the server into the buff buffer.
    d. Check if the received message is in order by comparing the last character of the message with the expected value (i + '0').
    e. If the message is out of order, print a discard message and decrement by 1.
    f. If the message is in order:
        •Print the received message and the corresponding index.
        •Print a message indicating that the acknowledgement is sent.
        •Construct the acknowledgement message in msg2 by appending the acknowledgement prefix (msg1) and the index value (i + '0').
        •Send the acknowledgement message to the server using the write() function.
9. Close the client socket.
10. End the program.

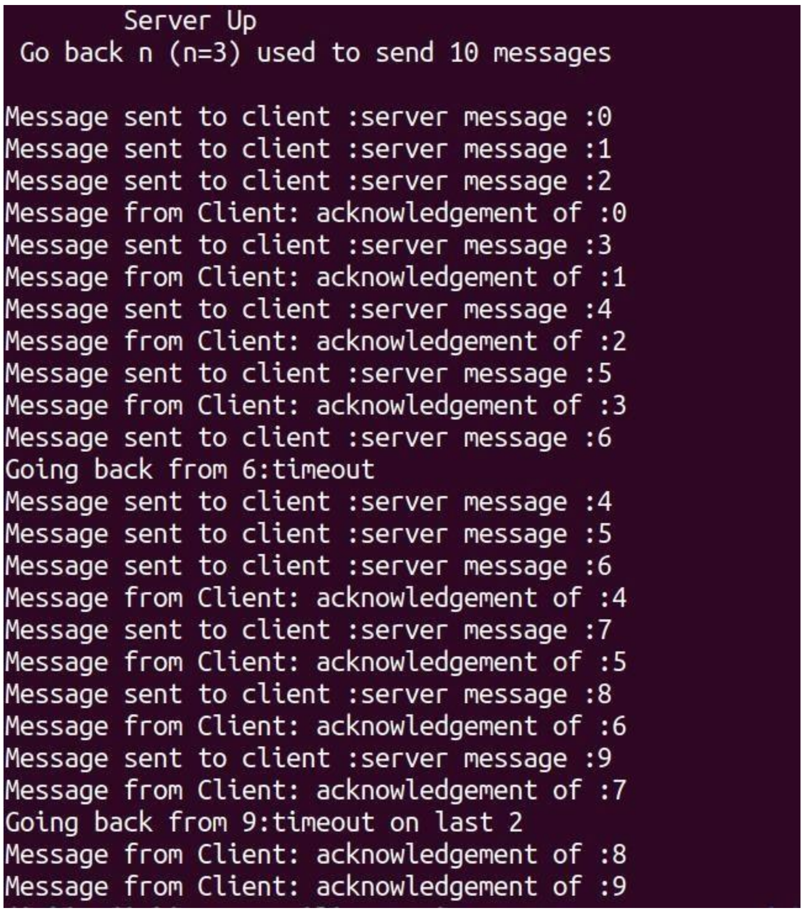
Client

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
int main(){
      int c_sock;
      c_sock = socket(AF_INET, SOCK_STREAM, 0);
      struct sockaddr_in  client;
      memset(&client, 0, sizeof(client));
      client.sin_family = AF_INET;
      client.sin_port = htons(9009);
      client.sin_addr.s_addr = inet_addr("127.0.0.1");
      if(connect(c_sock, (struct sockaddr *)&client, sizeof(client))==-1){
            printf("Connection failed");
            return 0;
      }
      printf("\n\tClient -with individual acknowledgement scheme\n\n");
      char msg1[50] = "acknowledgement of :";
      char msg2[50];
      char buff[100];
      int flag = 1, flg = 1;
      for (int i = 0; i <= 9; i++){
            flg = 1;
            bzero(buff, sizeof(buff));
            bzero(msg2, sizeof(msg2));
            if (i == 8 && flag == 1){
                  printf("here\n"); // simulating loss
                  flag = 0;
                  read(c_sock, buff, sizeof(buff));
            }
            int n = read(c_sock, buff, sizeof(buff));
            if (buff[strlen(buff) - 1] != i + '0'){ // out of order
                  printf("Discarded as out of order \n");
                  i--;
            }else{
                  printf("Message received from server:%s \t%d\n",buff, i);
                  printf("Acknowledgement sent for message \n");
                  strcpy(msg2, msg1);
                  msg2[strlen(msg2)] = i + '0';
                  write(c_sock, msg2, sizeof(msg2));
            }
      }
      close(c_sock);
      return 0;
}
```

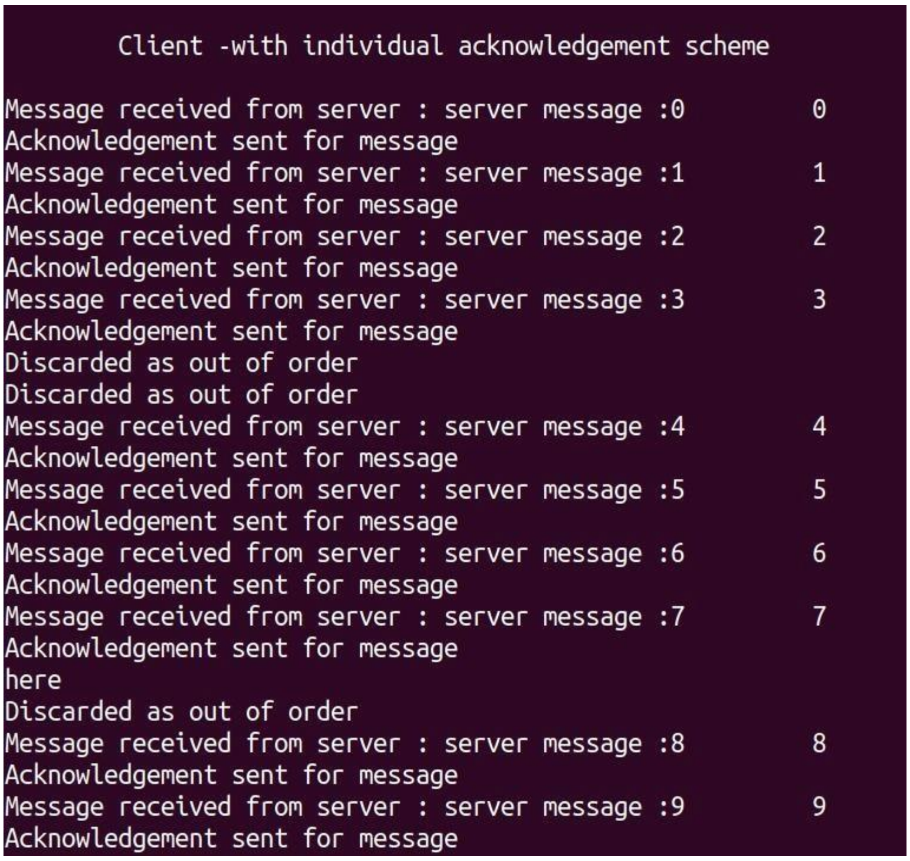
Go Back N Client :

1. Include the necessary header files.
2. Declare a variable for the client socket descriptor: c_sock.
3. Create a socket using the socket() function with domain AF_INET, type SOCK_STREAM, and protocol 0.
4. Initialize the client address structure (client) and set the port, IP address and family.
5. Connect the client socket to the server using the connect() function. If the connection fails, print an error message and exit.
6. Print a message indicating that the client is ready.
7. Initialize message buffers: msg1 for the acknowledgement message, msg2 for constructing acknowledgement messages, and buff for receiving data from the server.
8. Initialize flags: flag and flg.
9. Enter a loop from 0 to 9 to receive messages and send acknowledgements.
10. Clear the message and buffer.
11. If i is 8 and flag is 1, simulate a loss by reading from the server without storing the data.
12. Read the message from the server into the buff buffer.
13. Check if the received message is in order by comparing the last character of the message with the expected value.
14. If the message is out of order, print a discard message and decrement i by 1.
15. If the message is in order, print the received message and the corresponding index.
16. Print a message indicating that the acknowledgement is sent.
17. Construct the acknowledgement message in msg2 using the acknowledgement prefix and the index value.
18. Send the acknowledgement message to the server using the write() function.
19. Close the client socket.
20. End the program.

OUTPUT:



```
        Server Up
 Go back n (n=3) used to send 10 messages

Message sent to client :server message :0
Message sent to client :server message :1
Message sent to client :server message :2
Message from Client: acknowledgement of :0
Message sent to client :server message :3
Message from Client: acknowledgement of :1
Message sent to client :server message :4
Message from Client: acknowledgement of :2
Message sent to client :server message :5
Message from Client: acknowledgement of :3
Message sent to client :server message :6
Going back from 6:timeout
Message sent to client :server message :4
Message sent to client :server message :5
Message sent to client :server message :6
Message from Client: acknowledgement of :4
Message sent to client :server message :7
Message from Client: acknowledgement of :5
Message sent to client :server message :8
Message from Client: acknowledgement of :6
Message sent to client :server message :9
Message from Client: acknowledgement of :7
Going back from 9:timeout on last 2
Message from Client: acknowledgement of :8
Message from Client: acknowledgement of :9
```



```
        Client -with individual acknowledgement scheme

Message received from server : server message :0        0
Acknowledgement sent for message
Message received from server : server message :1        1
Acknowledgement sent for message
Message received from server : server message :2        2
Acknowledgement sent for message
Message received from server : server message :3        3
Acknowledgement sent for message
Discarded as out of order
Discarded as out of order
Message received from server : server message :4        4
Acknowledgement sent for message
Message received from server : server message :5        5
Acknowledgement sent for message
Message received from server : server message :6        6
Acknowledgement sent for message
Message received from server : server message :7        7
Acknowledgement sent for message
here
Discarded as out of order
Message received from server : server message :8        8
Acknowledgement sent for message
Message received from server : server message :9        9
Acknowledgement sent for message
```

RESULT:

The program for Go Back N was executed and output was verified successfully.

PROGRAM

```
#include <stdio.h>
struct router{
    unsigned cost[20];
    unsigned from[20];
} routingTable[10];

int main(){
    int costmat[20][20];
    int routers, i, j, k, count = 0;
    printf("\nEnter the number of routers : ");
    scanf("%d", &routers); // Enter the routers
    printf("\nEnter the cost matrix :\n");
    for (i = 0; i < routers; i++)
         {
        for (j = 0; j < routers; j++)
             {
            scanf("%d", &costmat[i][j]);
            costmat[i][i] = 0;
            routingTable[i].cost[j] = costmat[i][j];
            routingTable[i].from[j] = j;
        }
    }

    int otherShorterPathExists;
    do
    {
        otherShorterPathExists = 0;
        for (i = 0; i < routers; i++)
            for (j = 0; j < routers; j++)
                for (k = 0; k < routers; k++)
                    if (routingTable[i].cost[j] > costmat[i][k]+
routingTable[k].cost[j])
                    {
                        routingTable[i].cost[j] = routingTable[i].cost[k] +
routingTable[k].cost[j];
                        routingTable[i].from[j] = k;
                        otherShorterPathExists = 1;
                    }
    } while (otherShorterPathExists != 0);

    for (i = 0; i < routers; i++)
    {
        printf("\n\n For router %d\n", i + 1);
        for (j = 0; j < routers; j++)
        {
 printf("\t\nRouter %d via %d distance %d ", j + 1, routingTable[i].from[j] + 1,
routingTable[i].cost[j]);
        }
    }
    printf("\n\n");
}
```

Date : 11/04/2023

Experiment No : 8
Distance Vector Routing

AIM :

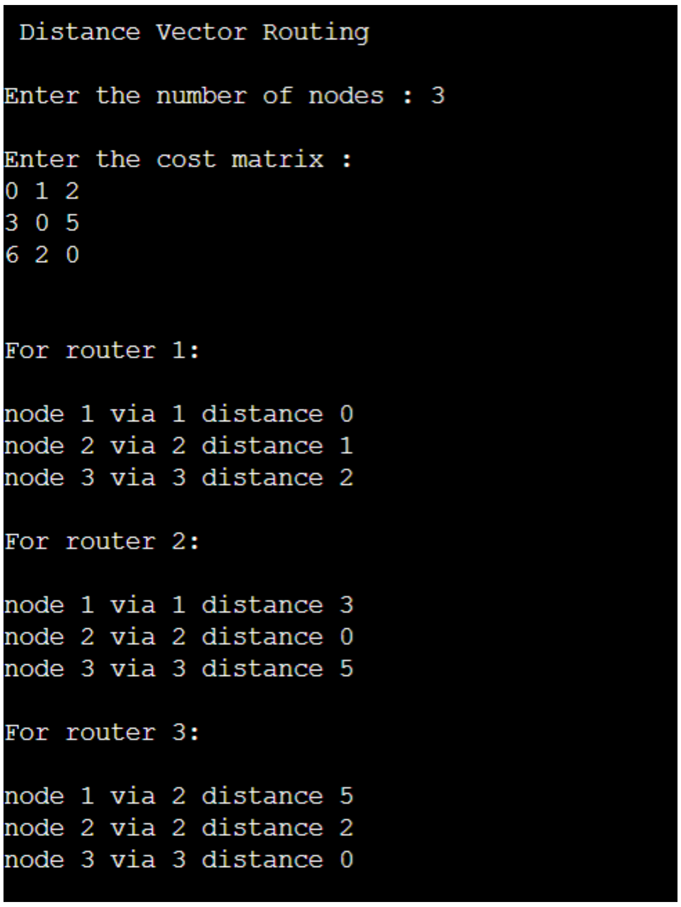
To implement and simulate algorithm for Distance Vector Routing protocol.

THEORY :

Distance Vector Routing (DVR) is a routing algorithm used in computer networks to determine the optimal path for sending data packets between nodes. In DVR, each node maintains a routing table that contains information about the distances to reach other nodes in the network. Nodes exchange this information with their neighbouring nodes, allowing each node to update its routing table based on the received information. This process continues until all nodes have converged on the shortest paths to all destinations. By implementing and simulating the DVR algorithm, researchers and network administrators can gain practical insights into how routers exchange routing information, collaborate to find the best paths, and adapt to changes in network topology.

The DVR experiment enables the study of various aspects of routing, such as efficiency, scalability, and stability. It helps evaluate how well the algorithm performs under different network conditions, including link failures and topology changes. By observing the evolution of routing tables over time, researchers can analyse the effectiveness of the DVR algorithm in finding optimal paths and maintaining network connectivity. Additionally, the experiment provides valuable insights into the dynamics of information propagation and routing decisions in a decentralized network. This understanding contributes to the development of more robust and efficient routing protocols for real-world networks.

OUTPUT



```
 Distance Vector Routing

Enter the number of nodes : 3

Enter the cost matrix :
0 1 2
3 0 5
6 2 0


For router 1:

node 1 via 1 distance 0
node 2 via 2 distance 1
node 3 via 3 distance 2

For router 2:

node 1 via 1 distance 3
node 2 via 2 distance 0
node 3 via 3 distance 5

For router 3:

node 1 via 2 distance 5
node 2 via 2 distance 2
node 3 via 3 distance 0
```

ALGORITHM:

1. Start by defining the structure for the routing table, which includes arrays for cost and next hop information for each router.
2. Initialize variables: costmat to hold the cost matrix, routers to store the number of routers, i, j, k as loop counters, and count to keep track of iterations.
3. Read the number of routers from the user.
4. Read the cost matrix from the user, populating the costmat array and initializing the routing table arrays.
5. Perform the distance vector algorithm until no more changes occur:
6. Set otherShorterPathExists flag to 0.
   a) Iterate over each router (outer loop).
   b) Iterate over each destination router (middle loop).
   c) Iterate over each intermediate router (inner loop).
   d) If a shorter path exists from i to j through k, update the routing table:
      - routingTable[i].cost[j] is updated to the sum of costs from i to k and from k to j.
      - routingTable[i].from[j] is updated to k.
      - Set otherShorterPathExists flag to 1 to indicate a change in the routing table.
      - Repeat the above loops until no more changes occur (otherShorterPathExists is 0).
7. Print the routing table for each router:
   a) Iterate over each router.
   b) Print the router's number.
   c) Iterate over each destination router.
   d) Print the destination router's number, the next hop (routingTable[i].from[j] + 1), and the cost (routingTable[i].cost[j]).
8. End the program

RESULT :

The program for Distance Vector Routing was executed and output was verified successfully.

PROGRAM

Server

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<stdlib.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<unistd.h>
#define BUF_SIZE 256

int main(int argc,char* argv[]) {
	struct sockaddr_in server,client;
	char str[50],msg[20];

	if(argc!=2)
		printf("Input format not correct");
	int sockfd=socket(AF_INET,SOCK_DGRAM,0);
	if(sockfd==-1)
		printf("Error in socket();");

	server.sin_family=AF_INET;
	server.sin_addr.s_addr=INADDR_ANY;
	server.sin_port=htons(atoi(argv[1]));
	client.sin_family=AF_INET;
	client.sin_addr.s_addr=INADDR_ANY;
	client.sin_port=htons(atoi(argv[1]));

	if(bind(sockfd,(struct sockaddr *)&server,sizeof(server))<0)
		printf("Error in bind()! \n");

	socklen_t client_len=sizeof(client);
	printf("server waiting...........");
	sleep(3);

	if(recvfrom(sockfd,str,100,0,(struct sockaddr *)&client, &client_len ) < 0)
		printf("Error in recvfrom()!");

	printf("\nGot message from client:%s",str);
	printf("\nSending greeting message to client");
	strcpy(str,"220 127.0.0.1");
	sleep(10);

	if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr*)&client,
sizeof(client))< 0)
		printf("Error in send");
	sleep(3);

	if((recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr*)&client,
&client_len)),0)
		printf("Error in recv");
	if(strncmp(str,"HELO",4))
		printf("\n'HELO' expected from client........");
	printf("\n%s",str);
	printf("\nSending response......");
	strcpy(str,"250 ok");

	if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr*)&client ,
sizeof(client))<0)
		printf("Error in send");
	sleep(3);
```

Date : 18/04/2023

Experiment No : 9
Simple Mail Transfer Protocol

AIM :

To implement and simulate algorithm for Simple Mail Transfer Protocol.

THEORY :

SMTP is part of the application layer of the TCP/IP protocol and traditionally operates on port 25. It utilizes a process called “store and forward” which is used to orchestrate sending your email across different networks. Within the SMTP protocol, there are smaller software services called Mail Transfer Agents that help manage the transfer of the email and its final delivery to a recipient’s mailbox. Not only does SMTP define this entire communication flow, it can also support delayed delivery of an email either at the sender site, receiver site, or at any intermediate server.

```
	if((recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr *)&client ,
&client_len))<0)
		printf("Error in recv");

	if(strncmp(str,"MAIL FROM",9))
		printf("MAIL FROM expected from client...");
	printf("\n%s",str);
	printf("\nSending response........");
	strcpy(str,"250 ok");

	if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr *)&client ,
sizeof(client))<0)
		printf("Error in send");
	sleep(3);

	if((recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr *)&client,
&client_len))<0)
		printf("Error in recv");

	if(strncmp(str,"RCPT TO",7))
		printf("\nRCPT TO expected from client........");
	printf("\n%s",str);
	printf("\nSending response........");
	strcpy(str,"250 ok");

	if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr *)&client,
sizeof(client))<0)
		printf("Error in send");
	sleep(3);

	if((recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr *)&client,
&client_len))<0)
		printf("Error in recv");
	if(strncmp(str,"DATA",4))
		printf("\nDATA expected from client........");
	printf("\n%s",str);
	printf("\nSending response........");
	strcpy(str,"354 Go ahead");

	if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr*)&client,
sizeof(client))<0)
		printf("Error in send");

	if((recvfrom(sockfd,msg,sizeof(str),0,(struct sockaddr *)&client,
&client_len))<0)
		printf("Error in recv");
	printf("mail body received");
	printf("\n%s",msg);

	if((recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr *)&client,
&client_len))<0)
		printf("Error in recv");

	if(strncmp(str,"QUIT",4))
		printf("quit expected from client........");
	printf("\nSending quit......");
	strcpy(str,"221 OK");

	if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr *)&client,
sizeof(client))<0)
		printf("Error in send");
	close(sockfd);
```

ALGORITHM:

Server :

1. Create a socket with type as SOCK_DGRAM to create a udp socket using socket() system call.Create a socket with type as SOCK_DGRAM to create a udp socket using socket() system call.
2. Bind the socket to a specific port using bind() system call.
3. Receive first packet from client using recvfrom() and reply with status code 220 with fully qualified domain name to indicate server ready.
4. Receive HELO command with fully qualified domain name from client using recvfrom() system call and respond with status code 250.
5. Receive MAIL command with FROM address from client using recvfrom() system call and respond with status code 250.
6. Receive RCPT command with TO address from client using recvfrom() system call and respond with status code 250.
7. Receive DATA command from client using recvfrom() system call and respond with status code 354 to start receiving mail body.
8. Receive the mail body from the client using recvfrom() system call.
9. Receive QUIT command] from client using recvfrom() system call and respond with status code 221.
10. Close the socket.

```
	return 0;
}
```

Client

```
#include<string.h>
#include<sys/socket.h>
#include<netdb.h>
#include<stdlib.h>
#include<stdio.h>
#include<netinet/in.h>
#include<sys/types.h>
#include<stddef.h>
#include<unistd.h>
#define BUF_SIZE 256

int main(int argc,char* argv[]){
	struct sockaddr_in server,client;
	char str[50]="hi";
	char mail_f[50],mail_to[50],msg[20],c;
	int t=0;
	socklen_t l=sizeof(server);
	if(argc!=3)
		printf("Input format not correct");
	int sockfd=socket(AF_INET,SOCK_DGRAM,0);
if(sockfd==-1)
		printf("Error in socket();");

	server.sin_family=AF_INET;
	server.sin_addr.s_addr=INADDR_ANY;
	server.sin_port=htons(atoi(argv[2]));
	client.sin_family=AF_INET;client.sin_addr.s_addr=INADDR_ANY;
	client.sin_port=htons(atoi(argv[2]));

	printf("Sending hi to server");
	sleep(10);
	if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr*)&server,
sizeof(server))<0)
		printf("Error in sento");

	if(recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,&l)<0)
		printf("Error in recv");

	printf("\ngreeting msg is %s",str);
	if(strncmp(str,"220",3))
		printf("\nConn not established \n code 220 expected");
	printf("\nSending HELO");
	strcpy(str,"HELO 127.0.0.1");

	if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,
sizeof(server))<0)
		printf("Error in sendto");
	sleep(3);

	printf("\nReceiving from server");
	if((recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,&l))<0)
		printf("Error in recv");

	if(strncmp(str,"250",3))
		printf("\nOk not received from server");

	printf("\nServer has send %s",str);
	printf("\nEnter FROM address\n");
```

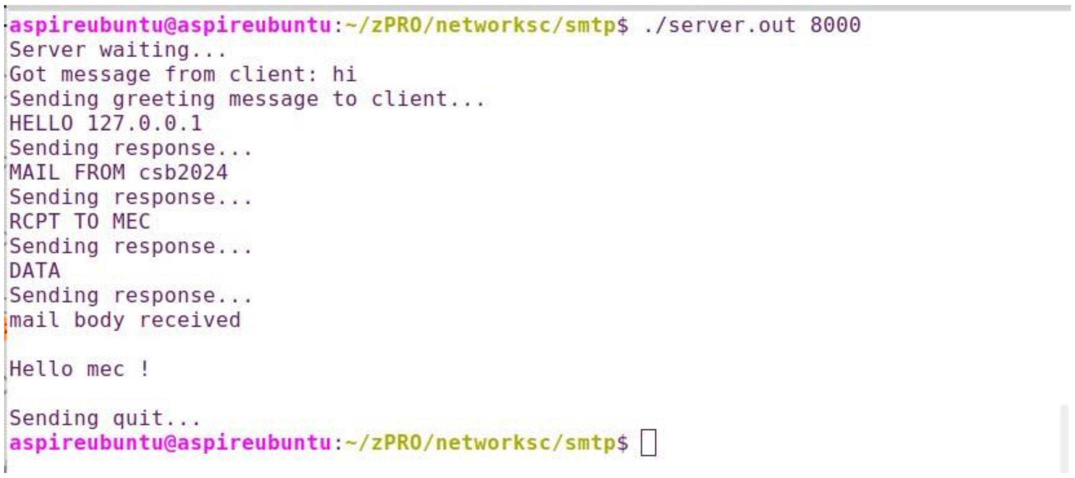
Client :

1. Create a socket with type as SOCK_DGRAM to create a udp socket using socket() system call.
2. Send an initial message to the server using sendto() and expect a response with status code 220 and fully qualified domain name of the server.
3. Send a HELO command to the server using sendto() system call with a fully qualified domain name and expect a response with status code 250.
4. Send a MAIL command along with FROM address to the server using sendto() system call and expect a response with status code 250.
5. Send a RCPT command along with TO address to the server using sendto() system call and expect a response with status code 250.
6. Send a DATA command to the server using sendto() system call and expect a response with status code 354.
7. Read the body of the mail from the user and send it to the server using sendto() system call.
8. Send a QUIT command to the server using sendto() system call and expect a response with status code 221.
9. Close the socket

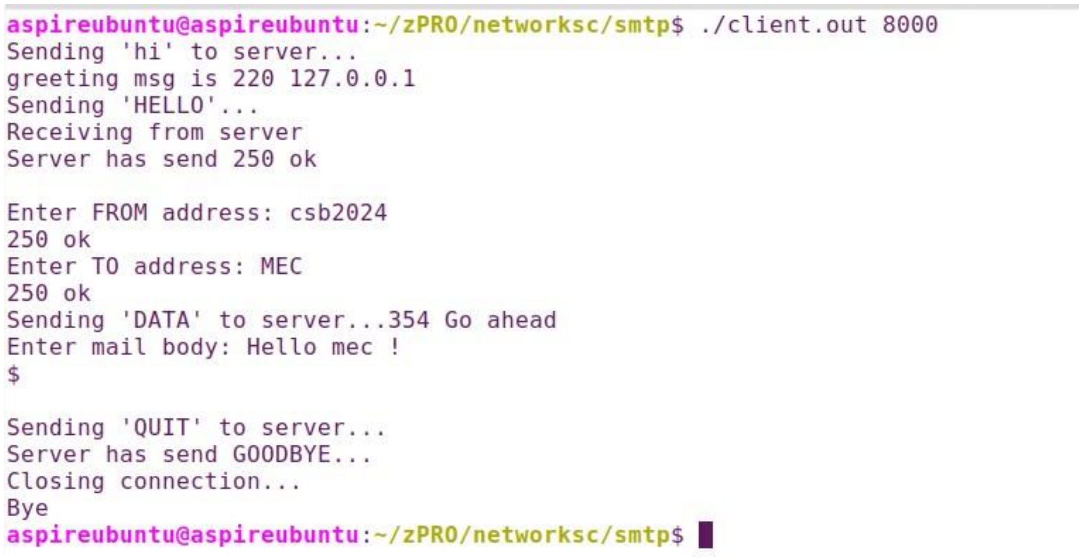
```
        scanf("%s",mail_f);
        strcpy(str,"MAIL FROM");
        strcat(str,mail_f);

        if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,
sizeof(server))<0)
                printf("Error in sendto");
        sleep(3);

        if((recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,&l))<0)
                printf("Error in recv");

        if(strncmp(str,"250",3))
                printf("\nOk not received from server");
        printf("%s",str);
        printf("\nEnter TO address\n");
        scanf("%s",mail_to);
        strcpy(str,"RCPT TO");
        strcat(str,mail_to);

        if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,
sizeof(server))<0)
                printf("Error in sendto");
        sleep(3);

        if((recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,&l))<0)
                printf("Error in recv");

        if(strncmp(str,"250",3))
        printf("\nOk not received from server");
        printf("%s",str);
        printf("\nSending DATA to server");
        strcpy(str,"DATA");

        if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,
sizeof(server))<0)
                printf("Error in sendto");
        sleep(3);
        if((recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,&l))<0)
                printf("Error in recv");

        if(strncmp(str,"354",3))
        printf("\nOk not received from server");
        printf("%s",str);
        printf("\nEnter mail body");

        while(1){
                c=getchar();
                if(c=='$'){
                        msg[t]='\0';
                        break;
                }
                if(c=='\0')
                        continue;
                msg[t++]=c;
                }

        if(sendto(sockfd,msg,sizeof(msg),0,(struct sockaddr *)&server,
sizeof(server))<0)
                printf("Error in sendto");
        sleep(3);
```

```
        printf("\nSending QUIT to server");
        strcpy(str,"QUIT");

        if(sendto(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,
sizeof(server))<0)
                printf("Error in sendto");

        if((recvfrom(sockfd,str,sizeof(str),0,(struct sockaddr *)&server,&l))<0)
                printf("Error in recv");

        if(strncmp(str,"221",3))
        printf("\nOk not received from server");
        printf("\nServer has send GOODBYE.....Closing conn\n");
        printf("\n Bye");
        close(sockfd);
        return 0;
}
```

OUTPUT



```
aspireubuntu@aspireubuntu:~/zPRO/networksc/smtp$ ./server.out 8000
Server waiting...
Got message from client: hi
Sending greeting message to client...
HELLO 127.0.0.1
Sending response...
MAIL FROM csb2024
Sending response...
RCPT TO MEC
Sending response...
DATA
Sending response...
mail body received

Hello mec !

Sending quit...
aspireubuntu@aspireubuntu:~/zPRO/networksc/smtp$
```



```
aspireubuntu@aspireubuntu:~/zPRO/networksc/smtp$ ./client.out 8000
Sending 'hi' to server...
greeting msg is 220 127.0.0.1
Sending 'HELLO'...
Receiving from server
Server has send 250 ok

Enter FROM address: csb2024
250 ok
Enter TO address: MEC
250 ok
Sending 'DATA' to server...354 Go ahead
Enter mail body: Hello mec !
$

Sending 'QUIT' to server...
Server has send GOODBYE...
Closing connection...
Bye
aspireubuntu@aspireubuntu:~/zPRO/networksc/smtp$
```

RESULT:

The program for Selective Repeat was executed and output was verified successfully.

PROGRAM

Server

```
#include <stdio.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

int main(){
    FILE *fp;
    int sd, newsd, ser, n, a, cli, pid, bd, port, clilen;
    char name[100], fileread[100], fname[100], ch, file[100], rcv[100];
    struct sockaddr_in servaddr, cliaddr;
    printf("Enter the port address\n");
    scanf("%d", &port);

    sd = socket(AF_INET, SOCK_STREAM, 0);
    if (sd < 0)
        printf("Cant 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(port);

    a = sizeof(servaddr);
    bd = bind(sd, (struct sockaddr *)&servaddr, a);
    if (bd < 0)
        printf("Cant bind\n");
    else
        printf("Binded\n");

    listen(sd, 5);
    clilen = sizeof(cliaddr);
    newsd = accept(sd, (struct sockaddr *)&cliaddr, &clilen);
    if (newsd < 0){
        printf("Cant accept\n");
    }
    else
        printf("Accepted\n");

    n = recv(newsd, rcv, 100, 0);
    rcv[n] = '\0';
    fp = fopen(rcv, "r");
    if (fp == NULL){
        send(newsd, "error", 5, 0);
        close(newsd);
    }else {
        while (fgets(fileread, sizeof(fileread), fp))  {
            if (send(newsd, fileread, sizeof(fileread), 0) < 0) {
                printf("Can't send file contents\n");
            }
            sleep(1);
        }
        send(newsd, "completed", 100, 0);
        return 0;
    }
}
```

Date : 18/04/2023

Experiment No : 10
File Transfer Protocol

AIM :

To implement and simulate algorithm for Distance Vector Routing protocol.

THEORY :

FTP, which stands for File Transfer Protocol, was developed in the 1970s to allow files to be transferred between a client and a server on a computer network. The FTP protocol uses two separate channels the command (or control) channel and the data channel to exchange files. The command channel is responsible for accepting client connections and executing other simple commands. It typically uses server port 21. FTP clients will connect to this port to initiate a conversation for file transfer and authenticate themselves by sending a username and password.

After authentication, the client and server will then negotiate a new common server port for the data channel, over which the file will be transferred. Once the file transfer is complete, the data channel is closed. If multiple files are to be sent concurrently, a range of data channel ports must be used. The control channel remains idle until the file transfer is complete. It then reports that the file transfer was either successful or failed.

Client

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <string.h>

int main(){
    FILE *fp;
    int csd, n, ser, s, cli, cport, newsd;
    char name[100], rcvmsg[100], rcvg[100], fname[100];
    struct sockaddr_in servaddr;
    printf("Enter the port");
    scanf("%d", &cport);
    csd = socket(AF_INET, SOCK_STREAM, 0);
    if (csd < 0){
        printf("Error…….\n");
        exit(0);
    }
    else
        printf("Socket is created\n");
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(cport);
    if (connect(csd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0)
        printf("Error in connection\n");
    else
        printf("connected\n");
    printf("Enter the existing file name\t");
    scanf("%s", name);
    printf("Enter the new file name\t");
    scanf("%s", fname);
    fp = fopen(fname, "w");
    send(csd, name, sizeof(name), 0);
    else
        printf("connected\n");
    printf("Enter the existing file name\t");
    scanf("%s", name);
    printf("Enter the new file name\t");
    scanf("%s", fname);
    fp = fopen(fname, "w");
    send(csd, name, sizeof(name), 0);
    while (1){
        bzero(rcvg, 100);
        s = recv(csd, rcvg, 100, 0);
        rcvg[s] = '\0';
        if (strcmp(rcvg, "error") == 0)
            printf("File is not available\n");
        if (strcmp(rcvg, "completed") == 0){
            printf("File is transferred……………\n");
            fclose(fp);
            close(csd);
            break;
        }
        else
            fputs(rcvg, stdout);
        fprintf(fp, "%s", rcvg);

    }
    return 0;
}
```

ALGORITHM

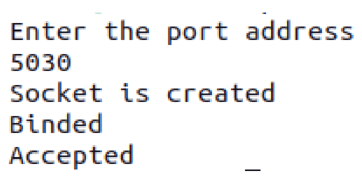
Server :

1. Create socket using socket() system call with address family AF_INET, type SOCK_STREAM and default protocol.
2. Bind server address and port using bind() system call.
3. Wait for client connection to complete accepting connections using accept() system call.
4. Receive the client file using recv() system call.
5. Using fgets(char *str, int n, FILE *stream) function, we need a line of text from the specified stream and store it into the string pointed by str. It stops when either (n-1) characters are read or when the end of file is reached.
6. On successful execution is when the file pointer reaches the end of file, file transfer “completed” message is sent by server to accepted client connection.

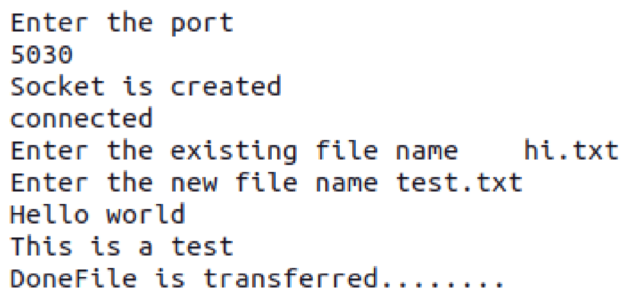
Client :

1. Create socket using socket() system call with address family AF_INET, type SOCK_STREAM and default protocol.
2. Enter the client port number.
3. Fill in the internal socket address structure
4. Connect to the server address using connect() system call.
5. Read the existing and new file name from the user.
6. Send existing file to server using send() system call.
7. Receive feedback from server “completed” regarding file transfer completion.
8. Write file is transferred to the standard output stream.
9. Close the socket connection and file pointer

OUTPUT



```
Enter the port address
5030
Socket is created
Binded
Accepted
```



```
Enter the port
5030
Socket is created
connected
Enter the existing file name    hi.txt
Enter the new file name test.txt
Hello world
This is a test
DoneFile is transferred........
```

<u>hi.txt</u>

```
Hello world
This is a test
Done
```

<u>test.txt</u>

```
Hello world
This is a test
Done
```

RESULT:

The programs for client-server communication for FTP was executed and output was verified successfully.

PROGRAM

```
#include<stdio.h>

void main(){
  int in,out,bsize,n,bucket=0;
  printf("Enter the bucket size:");//bucket size
  scanf("%d",&bsize);
  printf("Enter the number of inputs:");//number of inputs
  scanf("%d",&n);
  printf("Enter the packet outgoing rate:");//packet outgoing rate
  scanf("%d",&out);
  while(n!=0)
  {
    printf("\nEnter the incoming packet size:");
    scanf("%d",&in);
    printf("Incoming packet size:%d\n",in)     ;
    if(in<=(bsize-bucket))
    {
        bucket+=in;  //add to bucket
        printf("Bucket status:%d out of %d\n",bucket,bsize);
    }
    else
    {
        printf("Packet size greater than remaining bucket size !!\nPacket
dropped\n");
    }
    bucket=bucket-out;
    if (bucket<0)
      bucket=0;
    printf("After outgoing,bucket status:%d packets out of %d\n",bucket,bsize) ;
    n--;
  }
}
```

Date : 25/04/2023

Experiment No : 11
Leaky Bucket

AIM :

To implement congestion control using leaky bucket algorithm

THEORY :

The Leaky Bucket Algorithm used to control rate in a network. It is implemented as a single-server queue with constant service time. If the bucket (buffer) overflows then packets are discarded. It enforces a constant output rate (average rate) regardless of the burstiness of the input. It does nothing when input is idle. The host injects one packet per clock tick onto the network. This results in a uniform flow of packets, smoothing out bursts and reducing congestion.When packets are the same size (as in ATM cells), the one packet per tick is okay. For variable length packets though, it is better to allow a fixed number of bytes per tick.

ALGORITHM :

```
1.  Initialize variables: bsize, n, out, bucket as integers.
2.  Read and store the bucket size (bsize) from the user.
3.  Read and store the number of inputs (n) from the user.
4.  Read and store the packet outgoing rate (out) from the user.
5.  Read and store the incoming packet size (in) from the user.
6.  If in is less than or equal to the available space in the bucket (bsize - bucket), then:
    1.  Add the incoming packet size (in) to the bucket (bucket += in).
7.  If in is greater than the available space in the bucket (bsize - bucket), then:
    1.  Calculate the number of dropped packets (in - (bsize - bucket)).
    2.  7.2-Set the bucket size (bucket) to its maximum capacity (bsize).
    3.  Print the number of dropped packets.
8.  Print the bucket status (bucket packets out of bsize).
9.  Subtract the packet outgoing rate (out) from the bucket size (bucket).
10. Print the bucket status after outgoing packets (bucket packets out of bsize).
11. Decrement the value of n by 1.
12. Repeat steps 5 to 11 until n becomes zero
```

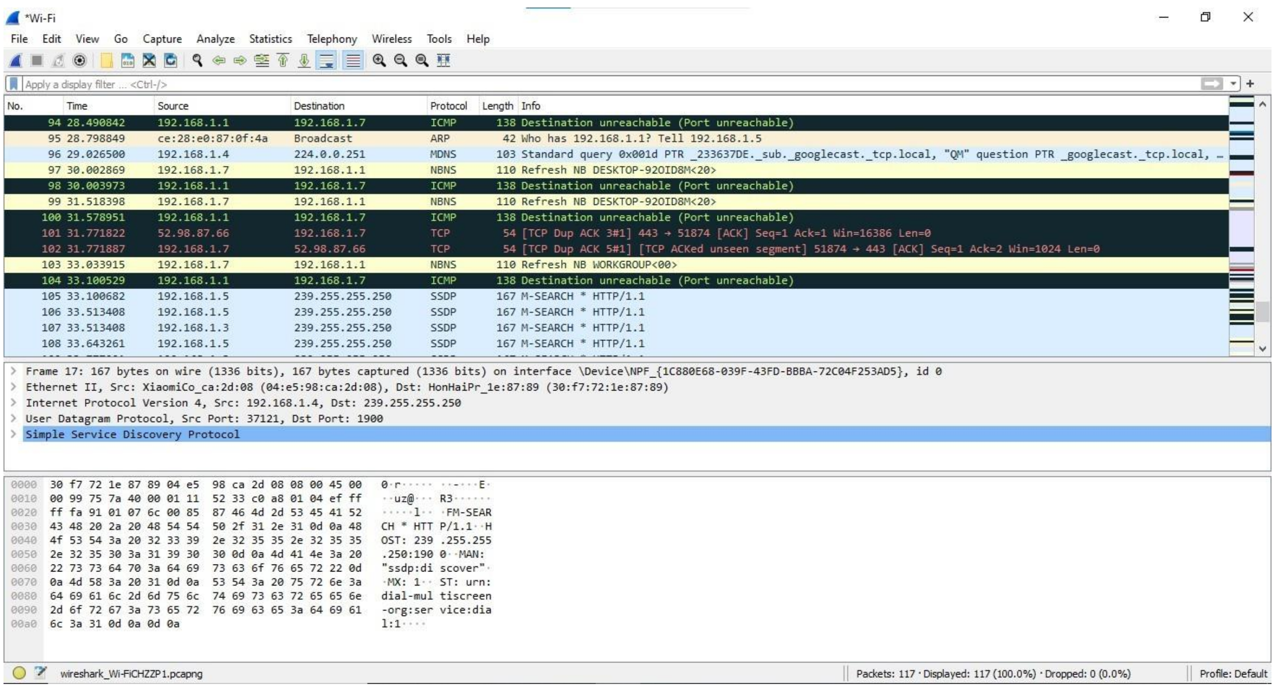
OUTPUT

```
Enter the bucket size:1000
Enter the number of inputs:5
Enter the packet outgoing rate:400

Enter the incoming packet size:300
Incoming packet size:300
Bucket status:300 out of 1000
After outgoing,bucket status:0 packets out of 1000

Enter the incoming packet size:400
Incoming packet size:400
Bucket status:400 out of 1000
After outgoing,bucket status:0 packets out of 1000

Enter the incoming packet size:700
Incoming packet size:700
Bucket status:700 out of 1000
After outgoing,bucket status:300 packets out of 1000

Enter the incoming packet size:200
Incoming packet size:200
Bucket status:500 out of 1000
After outgoing,bucket status:100 packets out of 1000

Enter the incoming packet size:1000
Incoming packet size:1000
Packet size greater than remaining bucket size !!
Packet dropped
After outgoing,bucket status:0 packets out of 1000
```

RESULT :

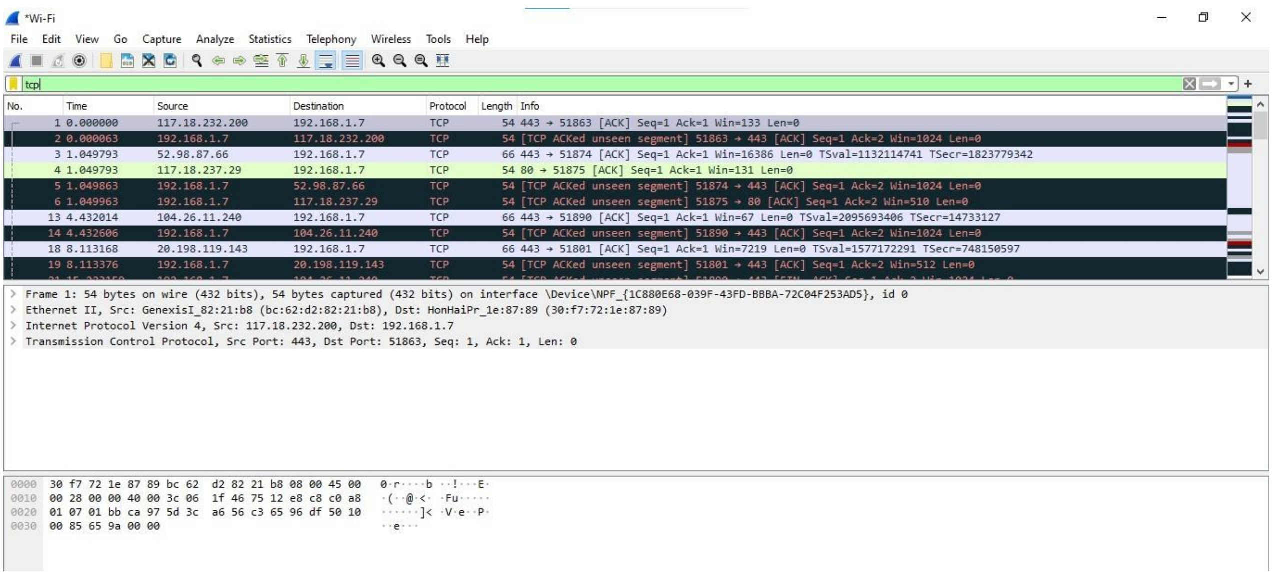
The program for Leaky Bucket was executed and output was verified successfully.

OUTPUT :



```
No.   Time        Source              Destination       Protocol Length Info
   94 28.490842   192.168.1.1         192.168.1.7       ICMP     138 Destination unreachable (Port unreachable)
   95 28.798849   ce:28:e0:87:0f:4a   Broadcast         ARP       42 Who has 192.168.1.1? Tell 192.168.1.5
   96 29.026500   192.168.1.4         224.0.0.251       MDNS     103 Standard query 0x001d PTR _233637DE._sub._googlecast._tcp.local, "QM" question PTR _googlecast._tcp.local, …
   97 30.002869   192.168.1.7         192.168.1.1       NBNS     110 Refresh NB DESKTOP-92OID8M<20>
   98 30.003973   192.168.1.1         192.168.1.7       ICMP     138 Destination unreachable (Port unreachable)
   99 31.518398   192.168.1.7         192.168.1.1       NBNS     110 Refresh NB DESKTOP-92OID8M<20>
  100 31.578951   192.168.1.1         192.168.1.7       ICMP     138 Destination unreachable (Port unreachable)
  101 31.771822   52.98.87.66         192.168.1.7       TCP       54 [TCP Dup ACK 3#1] 443 → 51874 [ACK] Seq=1 Ack=1 Win=16386 Len=0
  102 31.771887   192.168.1.7         52.98.87.66       TCP       54 [TCP Dup ACK 5#1] [TCP ACKed unseen segment] 51874 → 443 [ACK] Seq=1 Ack=2 Win=1024 Len=0
  103 33.033915   192.168.1.7         192.168.1.1       NBNS     110 Refresh NB WORKGROUP<00>
  104 33.100529   192.168.1.1         192.168.1.7       ICMP     138 Destination unreachable (Port unreachable)
  105 33.100682   192.168.1.5         239.255.255.250   SSDP     167 M-SEARCH * HTTP/1.1
  106 33.513408   192.168.1.5         239.255.255.250   SSDP     167 M-SEARCH * HTTP/1.1
  107 33.513408   192.168.1.3         239.255.255.250   SSDP     167 M-SEARCH * HTTP/1.1
  108 33.643261   192.168.1.5         239.255.255.250   SSDP     167 M-SEARCH * HTTP/1.1

> Frame 17: 167 bytes on wire (1336 bits), 167 bytes captured (1336 bits) on interface \Device\NPF_{1C880E68-039F-43FD-BBBA-72C04F253AD5}, id 0
> Ethernet II, Src: XiaomiCo_ca:2d:08 (04:e5:98:ca:2d:08), Dst: HonHaiPr_1e:87:89 (30:f7:72:1e:87:89)
> Internet Protocol Version 4, Src: 192.168.1.4, Dst: 239.255.255.250
> User Datagram Protocol, Src Port: 37121, Dst Port: 1900
> Simple Service Discovery Protocol

0000  30 f7 72 1e 87 89 04 e5  98 ca 2d 08 08 00 45 00   0·r····· ··-···E·
0010  00 99 75 7a 40 00 01 11  52 33 c0 a8 01 04 ef ff   ··uz@··· R3······
0020  ff fa 91 01 07 6c 00 85  87 46 4d 2d 53 45 41 52   ·····l·· ·FM-SEAR
0030  43 48 20 2a 20 48 54 54  50 2f 31 2e 31 0d 0a 48   CH * HTT P/1.1··H
0040  4f 53 54 3a 20 32 33 39  2e 32 35 35 2e 32 35 35   OST: 239 .255.255
0050  2e 32 35 30 3a 31 39 30  30 0d 0a 4d 41 4e 3a 20   .250:190 0··MAN: 
0060  22 73 73 64 70 3a 64 69  73 63 6f 76 65 72 22 0d   "ssdp:di scover"·
0070  0a 4d 58 3a 20 31 0d 0a  53 54 3a 20 75 72 6e 3a   ·MX: 1·· ST: urn:
0080  64 69 61 6c 2d 6d 75 6c  74 69 73 63 72 65 65 6e   dial-mul tiscreen
0090  2d 6f 72 67 3a 73 65 72  76 69 63 65 3a 64 69 61   -org:ser vice:dia
00a0  6c 3a 31 0d 0a 0d 0a                              l:1····

wireshark_Wi-FiCHZZP1.pcapng     Packets: 117 · Displayed: 117 (100.0%) · Dropped: 0 (0.0%)     Profile: Default
```

Tcp filter



```
tcp

No.   Time        Source              Destination       Protocol Length Info
    1 0.000000    117.18.232.200      192.168.1.7       TCP       54 443 → 51863 [ACK] Seq=1 Ack=1 Win=133 Len=0
    2 0.000063    192.168.1.7         117.18.232.200    TCP       54 [TCP ACKed unseen segment] 51863 → 443 [ACK] Seq=1 Ack=2 Win=1024 Len=0
    3 1.049793    52.98.87.66         192.168.1.7       TCP       66 443 → 51874 [ACK] Seq=1 Ack=1 Win=16386 Len=0 TSval=1132114741 TSecr=1823779342
    4 1.049793    117.18.237.29       192.168.1.7       TCP       54 80 → 51875 [ACK] Seq=1 Ack=1 Win=131 Len=0
    5 1.049863    192.168.1.7         52.98.87.66       TCP       54 [TCP ACKed unseen segment] 51874 → 443 [ACK] Seq=1 Ack=2 Win=1024 Len=0
    6 1.049963    192.168.1.7         117.18.237.29     TCP       54 [TCP ACKed unseen segment] 51875 → 80 [ACK] Seq=1 Ack=2 Win=510 Len=0
   13 4.432014    104.26.11.240       192.168.1.7       TCP       66 443 → 51890 [ACK] Seq=1 Ack=1 Win=67 Len=0 TSval=2095693406 TSecr=14733127
   14 4.432606    192.168.1.7         104.26.11.240     TCP       54 [TCP ACKed unseen segment] 51890 → 443 [ACK] Seq=1 Ack=2 Win=1024 Len=0
   18 8.113168    20.198.119.143      192.168.1.7       TCP       66 443 → 51801 [ACK] Seq=1 Ack=1 Win=7219 Len=0 TSval=1577172291 TSecr=748150597
   19 8.113376    192.168.1.7         20.198.119.143    TCP       54 [TCP ACKed unseen segment] 51801 → 443 [ACK] Seq=1 Ack=2 Win=512 Len=0

> Frame 1: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF_{1C880E68-039F-43FD-BBBA-72C04F253AD5}, id 0
> Ethernet II, Src: GenexisI_82:21:b8 (bc:62:d2:82:21:b8), Dst: HonHaiPr_1e:87:89 (30:f7:72:1e:87:89)
> Internet Protocol Version 4, Src: 117.18.232.200, Dst: 192.168.1.7
> Transmission Control Protocol, Src Port: 443, Dst Port: 51863, Seq: 1, Ack: 1, Len: 0

0000  30 f7 72 1e 87 89 bc 62  d2 82 21 b8 08 00 45 00   0·r····b ··!···E·
0010  00 28 00 00 40 00 3c 06  1f 46 75 12 e8 c8 c0 a8   ·(··@·<· ·Fu·····
0020  01 07 01 bb ca 97 5d 3c  a6 56 c3 65 96 df 50 10   ······]< ·V·e··P·
0030  00 85 65 9a 00 00                                  ··e···
```

Date : 25/04/2023

Experiment No : 12
Wireshark

AIM :

To understand Wireshark tool and explore its features like filters, flow graphs, statistics and protocol hierarchy.

THEORY :

Wireshark is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development.It can parse and display the fields, along with their meanings as specified by different networking protocols. Wireshark uses pcap to capture packets, so it can only capture packets on the types of networks that pcap supports. Wireshark can color packets based on rules that match particular fields in packets, to help the user identify the types of traffic at a glance.

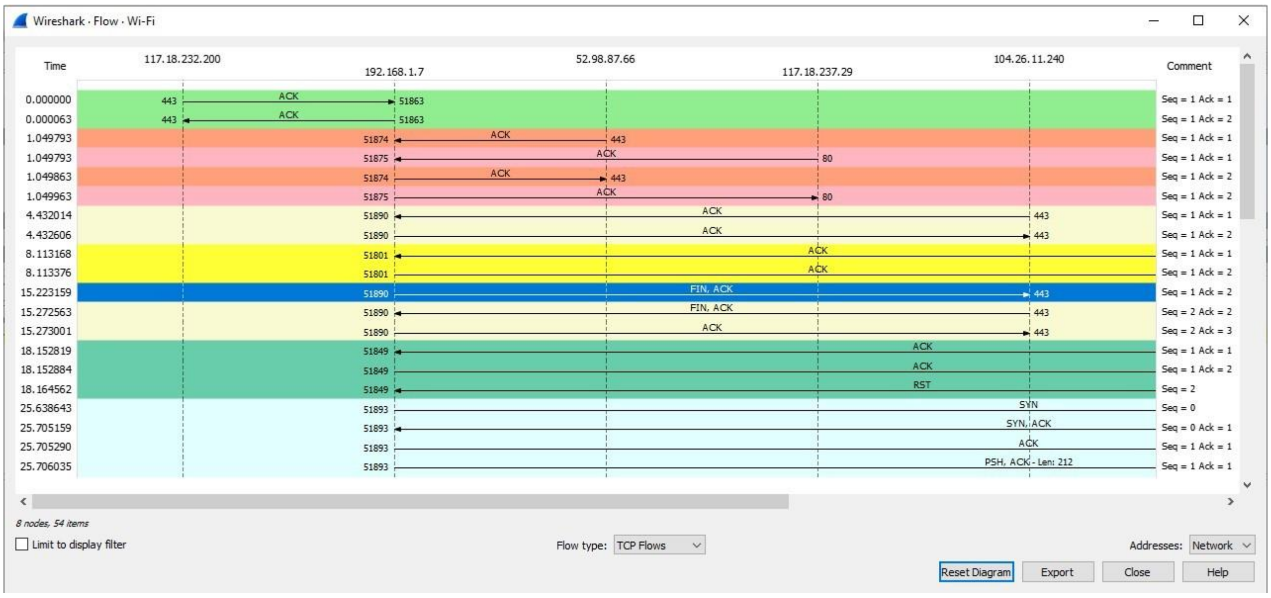
Filters :

Wireshark share a powerful filter engine that helps remove the noise from a packet trace and lets you see only the packets that interest you. If a packet meets the requirements expressed in your filter, then it is displayed in the list of packets. Display filters let you compare the fields within a protocol against a specific value, compare fields against fields, and check the existence of specified fields or protocols.

Flow Graph :

The Flow Graph window shows connections between hosts. It displays the packet time, direction, ports and comments for each captured connection. You can filter all connections by ICMP Flows, ICMPv6 Flows, UIM Flows and TCP Flows. Flow Graph window is used for showing multiple different topics. Each vertical line represents the specific host, which you can see in the top of the window. The numbers in each row at the very left of the window represent the time packet. The numbers at the both ends of each arrow between hosts represent the port numbers.

Flow graphs


Wireshark · Flow · Wi-Fi

Time	117.18.232.200		192.168.1.7		52.98.87.66		117.18.237.29		104.26.11.240	Comment
0.000000	443	ACK →	51863							Seq = 1 Ack = 1
0.000063	443	← ACK	51863							Seq = 1 Ack = 2
1.049793			51874	← ACK	443					Seq = 1 Ack = 1
1.049793			51875		← ACK		80			Seq = 1 Ack = 1
1.049863			51874	ACK →	443					Seq = 1 Ack = 2
1.049963			51875		ACK →		80			Seq = 1 Ack = 2
4.432014			51890			← ACK			443	Seq = 1 Ack = 1
4.432606			51890			ACK →			443	Seq = 1 Ack = 2
8.113168			51801				← ACK			Seq = 1 Ack = 1
8.113376			51801				ACK →			Seq = 1 Ack = 2
15.223159			51890			FIN, ACK →			443	Seq = 1 Ack = 2
15.272563			51890			← FIN, ACK			443	Seq = 2 Ack = 2
15.273001			51890			ACK →			443	Seq = 2 Ack = 3
18.152819			51849					← ACK		Seq = 1 Ack = 1
18.152884			51849					ACK →		Seq = 1 Ack = 2
18.164562			51849					← RST		Seq = 2
25.638643			51893						SYN →	Seq = 0
25.705159			51893						← SYN, ACK	Seq = 0 Ack = 1
25.705290			51893						ACK →	Seq = 1 Ack = 1
25.706035			51893						PSH, ACK - Len: 212 →	Seq = 1 Ack = 1

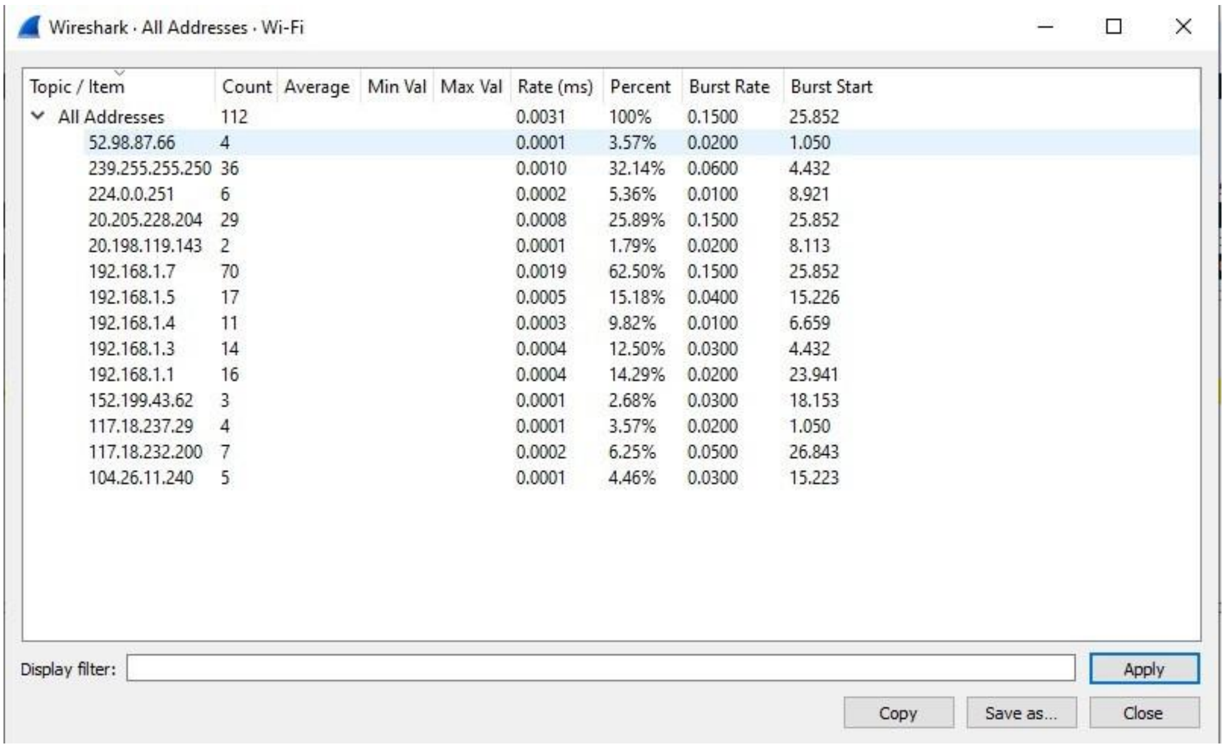
8 nodes, 54 items

☐ Limit to display filter

Flow type: TCP Flows

Addresses: Network

Statistics


Wireshark · All Addresses · Wi-Fi

Topic / Item	Count	Average	Min Val	Max Val	Rate (ms)	Percent	Burst Rate	Burst Start
All Addresses	112				0.0031	100%	0.1500	25.852
52.98.87.66	4				0.0001	3.57%	0.0200	1.050
239.255.255.250	36				0.0010	32.14%	0.0600	4.432
224.0.0.251	6				0.0002	5.36%	0.0100	8.921
20.205.228.204	29				0.0008	25.89%	0.1500	25.852
20.198.119.143	2				0.0001	1.79%	0.0200	8.113
192.168.1.7	70				0.0019	62.50%	0.1500	25.852
192.168.1.5	17				0.0005	15.18%	0.0400	15.226
192.168.1.4	11				0.0003	9.82%	0.0100	6.659
192.168.1.3	14				0.0004	12.50%	0.0300	4.432
192.168.1.1	16				0.0004	14.29%	0.0200	23.941
152.199.43.62	3				0.0001	2.68%	0.0300	18.153
117.18.237.29	4				0.0001	3.57%	0.0200	1.050
117.18.232.200	7				0.0002	6.25%	0.0500	26.843
104.26.11.240	5				0.0001	4.46%	0.0300	15.223

Display filter:

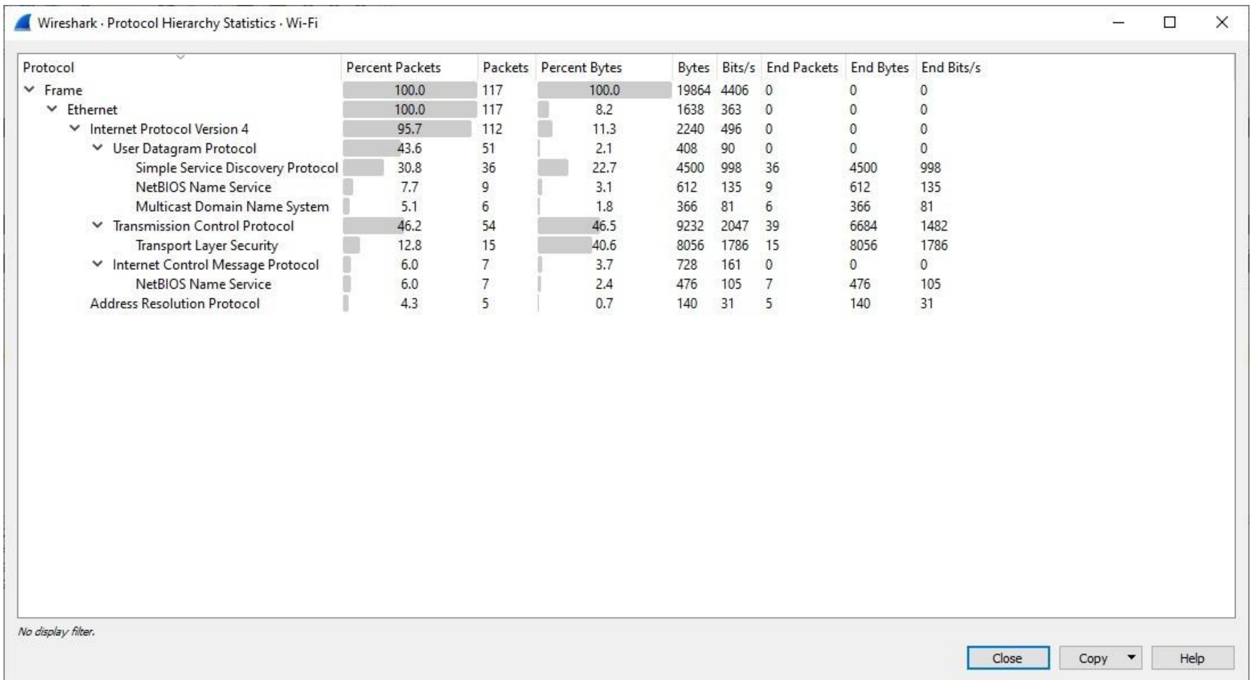
Statistics :

Wireshark provides a wide range of network statistics. These statistics range from general information about the loaded capture file (like the number of captured packets), to statistics about specific protocols (e.g. statistics about the number of HTTP requests and responses captured). General statistics involve Summary about the capture file like: packet counts, captured time period, Protocol Hierarchy of the captured packets, Conversations like traffic between specific Ethernet/IP/… addresses etc.

Protocol Hierarchy :

This is a tree of all the protocols in the capture. Each row contains the statistical values of one protocol. Two of the columns (Percent Packets and Percent Bytes) serve double duty as bar graphs. If a display filter is set it will be shown at the bottom.

Protocol Hierarchy



Wireshark · Protocol Hierarchy Statistics · Wi-Fi

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	117	100.0	19864	4406	0	0	0
Ethernet	100.0	117	8.2	1638	363	0	0	0
Internet Protocol Version 4	95.7	112	11.3	2240	496	0	0	0
User Datagram Protocol	43.6	51	2.1	408	90	0	0	0
Simple Service Discovery Protocol	30.8	36	22.7	4500	998	36	4500	998
NetBIOS Name Service	7.7	9	3.1	612	135	9	612	135
Multicast Domain Name System	5.1	6	1.8	366	81	6	366	81
Transmission Control Protocol	46.2	54	46.5	9232	2047	39	6684	1482
Transport Layer Security	12.8	15	40.6	8056	1786	15	8056	1786
Internet Control Message Protocol	6.0	7	3.7	728	161	0	0	0
NetBIOS Name Service	6.0	7	2.4	476	105	7	476	105
Address Resolution Protocol	4.3	5	0.7	140	31	5	140	31

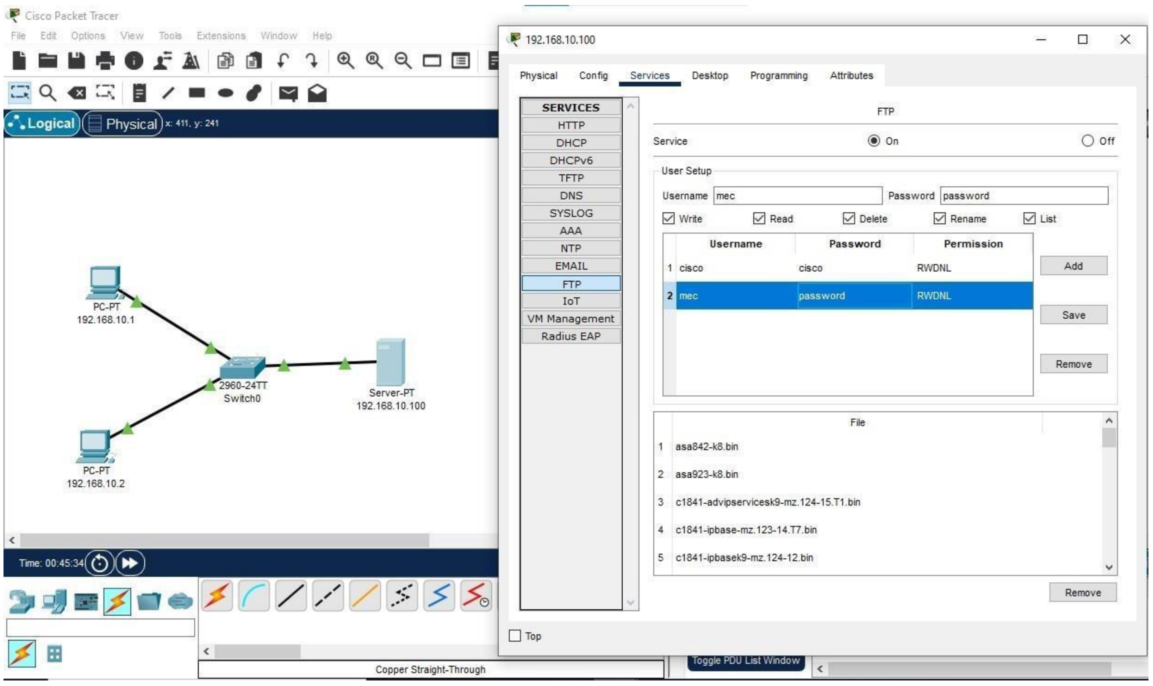
No display filter.

RESULT:

The experiment was executed successfully.

OUTPUT

FTP:



FTP

Service [✓ On] Off

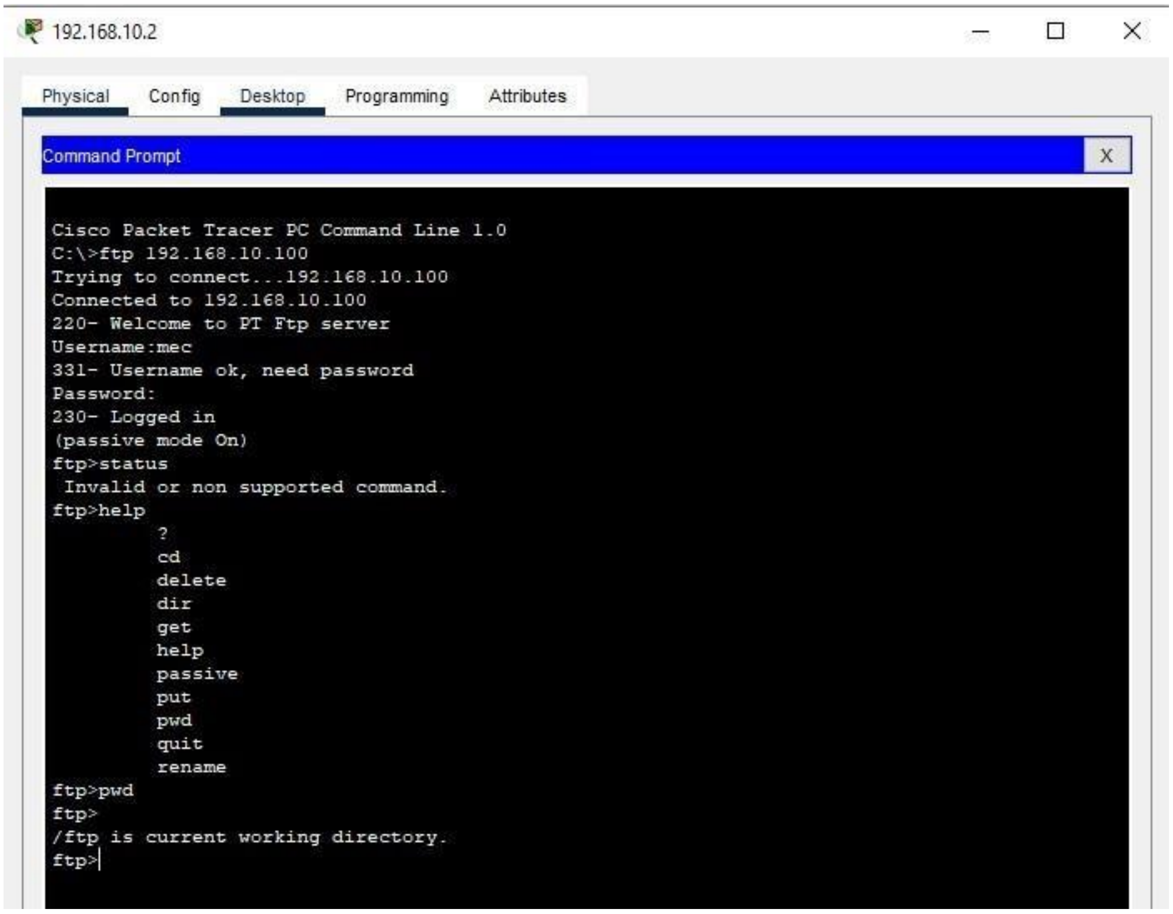
User Setup

Username mec Password password

☑ Write ☑ Read ☑ Delete ☑ Rename ☑ List

	Username	Password	Permission
1	cisco	cisco	RWDNL
2	mec	password	RWDNL

	File
1	asa842-k8.bin
2	asa923-k8.bin
3	c1841-advipservicesk9-mz.124-15.T1.bin
4	c1841-ipbase-mz.123-14.T7.bin
5	c1841-ipbasek9-mz.124-12.bin



```
Cisco Packet Tracer PC Command Line 1.0
C:\>ftp 192.168.10.100
Trying to connect...192.168.10.100
Connected to 192.168.10.100
220- Welcome to PT Ftp server
Username:mec
331- Username ok, need password
Password:
230- Logged in
(passive mode On)
ftp>status
 Invalid or non supported command.
ftp>help
         ?
         cd
         delete
         dir
         get
         help
         passive
         put
         pwd
         quit
         rename
ftp>pwd
ftp>
/ftp is current working directory.
ftp>
```

Date : 01/06/2023

Experiment No : 13
Network With Multiple Subnets

AIM :

Study of Cisco Packet Tracer and configure FTP server, DHCP server and DNS server in a wired network using required network devices.

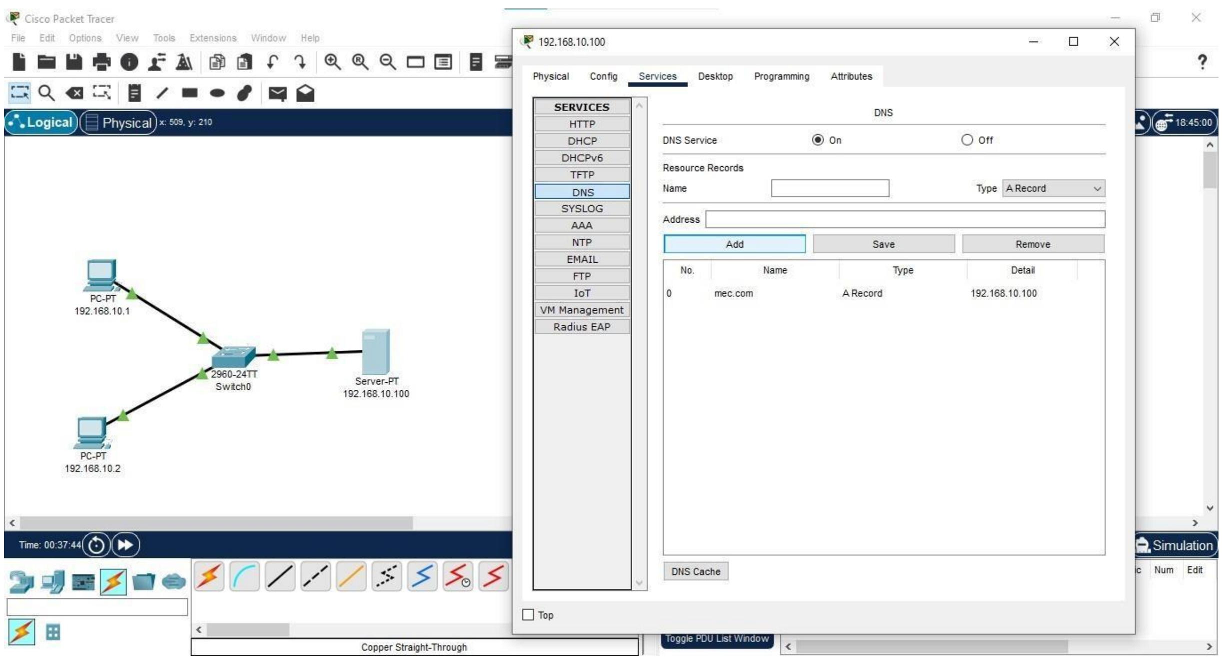
THEORY :

Packet Tracer is a cross-platform visual simulation tool designed by Cisco Systems that allows users to create network topologies and imitate modern computer networks. The software allows users to simulate the configuration of Cisco routers and switches using a simulated command line interface. Packet Tracer supports an array of simulated Application Layer protocols, as well as basic routing with RIP, OSPF, EIGRP and BGP.

DNS: The Domain Name System (DNS) is the hierarchical and decentralized naming system used to identify computers reachable through the Internet or other Internet Protocol (IP) networks. The resource records contained in the DNS associate domain names with other forms of information. These are most commonly used to map human-friendly domain names to the numerical IP addresses computers need to locate services and devices using the underlying network protocols.

DHCP: The Dynamic Host Configuration Protocol (DHCP) is a network management protocol used on Internet Protocol networks for automatically assigning IP addresses and other communication parameters to devices connected to the network using a client–server architecture. It employs a connectionless service model, using the User Datagram Protocol (UDP). It is implemented with two UDP port numbers, 67 is the destination port of a server, and is used by the client.

DNS:



192.168.10.100

Physical Config Services Desktop Programming Attributes

SERVICES: HTTP, DHCP, DHCPv6, TFTP, DNS, SYSLOG, AAA, NTP, EMAIL, FTP, IoT, VM Management, Radius EAP

DNS

DNS Service [✓ On] Off

Resource Records

Name

Type A Record

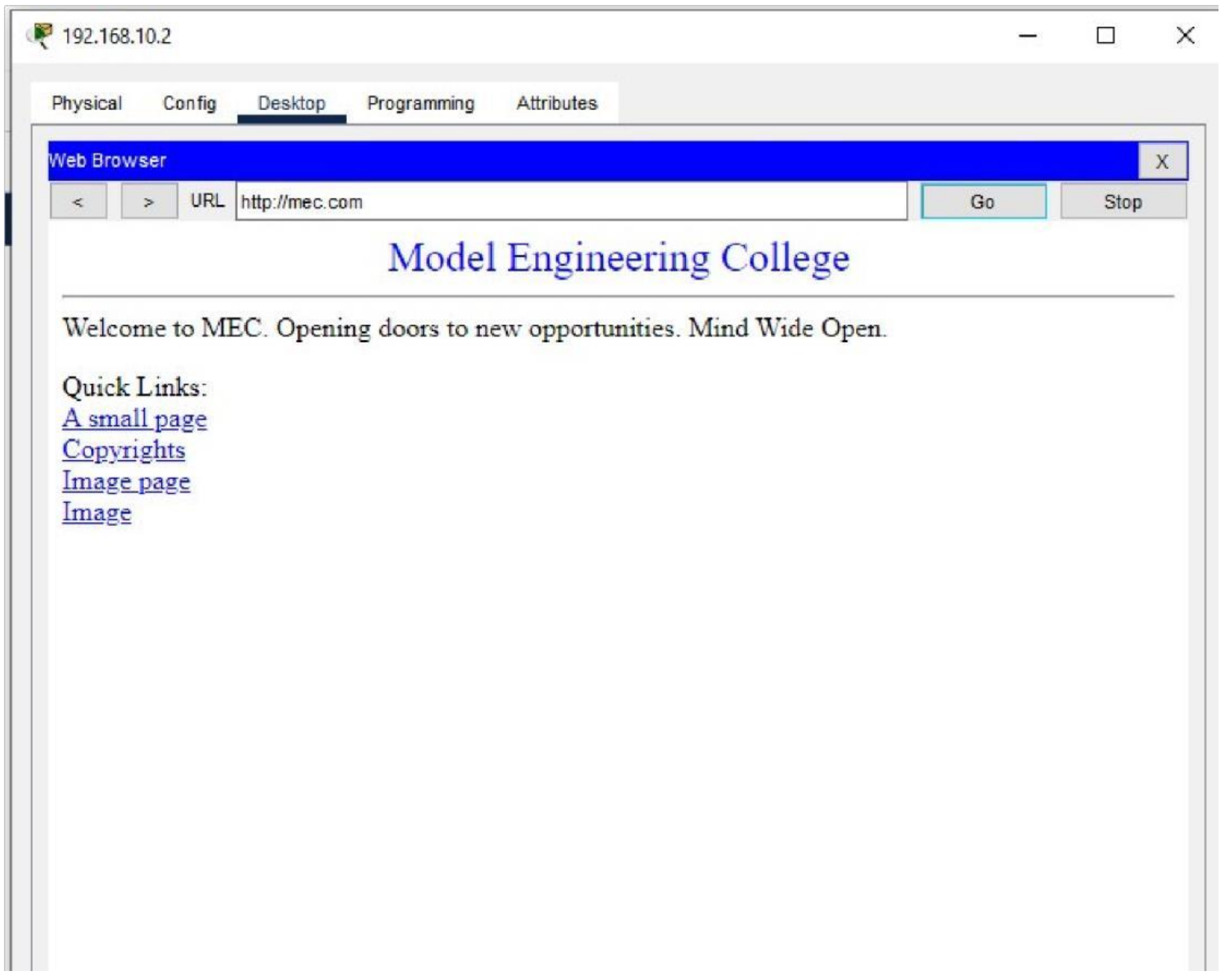
Address

Add Save Remove

No.	Name	Type	Detail
0	mec.com	A Record	192.168.10.100

DNS Cache

Top



192.168.10.2

Physical Config Desktop Programming Attributes

Web Browser

URL http://mec.com

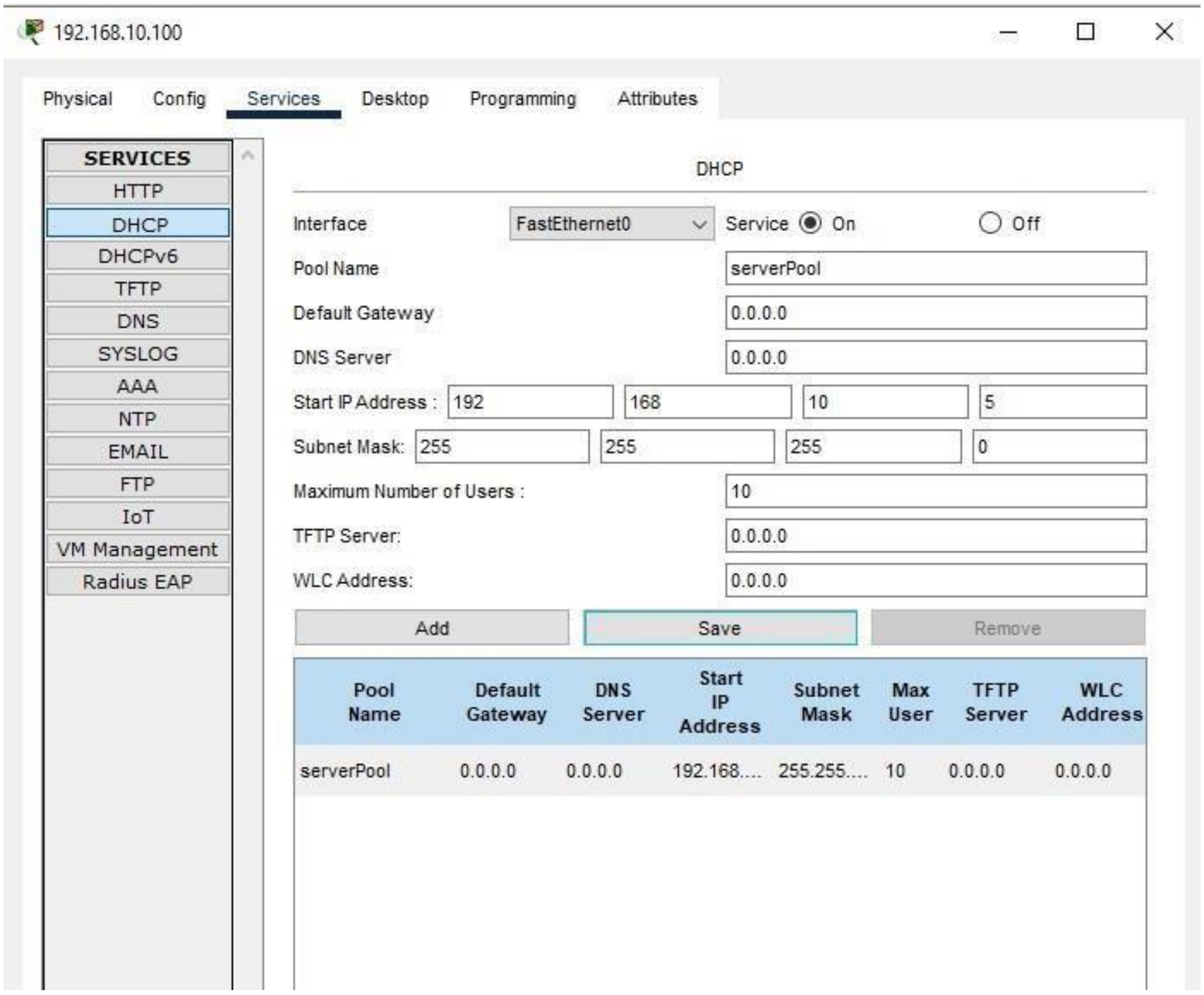
Go Stop

Model Engineering College

Welcome to MEC. Opening doors to new opportunities. Mind Wide Open.

Quick Links:
A small page
Copyrights
Image page
Image

DHCP:



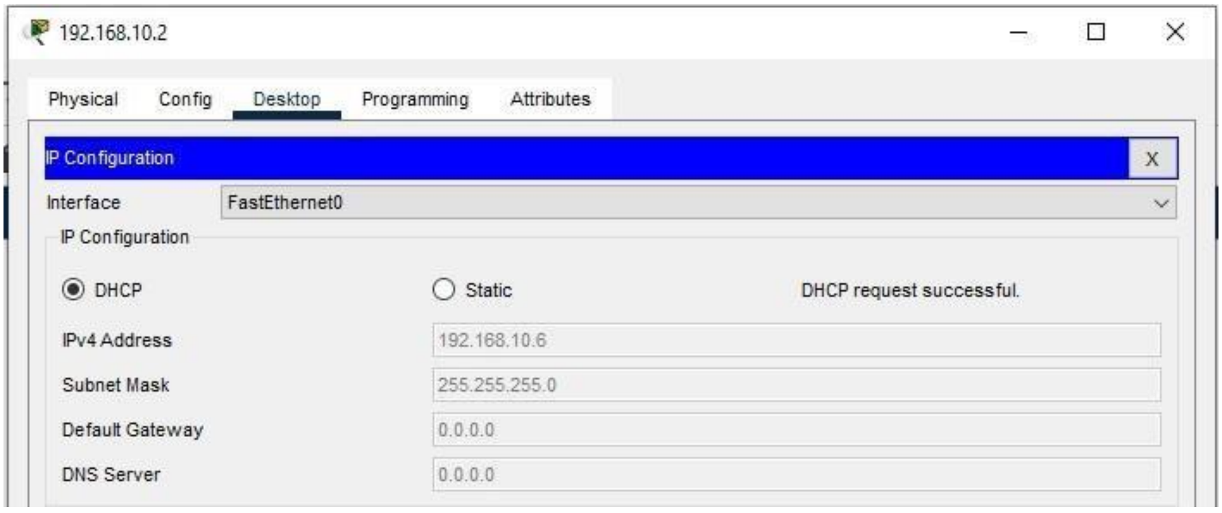
192.168.10.100

DHCP

Interface: FastEthernet0 — Service [✓ On] Off

Pool Name	serverPool
Default Gateway	0.0.0.0
DNS Server	0.0.0.0
Start IP Address :	192 . 168 . 10 . 5
Subnet Mask:	255 . 255 . 255 . 0
Maximum Number of Users :	10
TFTP Server:	0.0.0.0
WLC Address:	0.0.0.0

Pool Name	Default Gateway	DNS Server	Start IP Address	Subnet Mask	Max User	TFTP Server	WLC Address
serverPool	0.0.0.0	0.0.0.0	192.168....	255.255....	10	0.0.0.0	0.0.0.0



192.168.10.2

Interface: FastEthernet0

IP Configuration

[✓ DHCP] Static — DHCP request successful.

IPv4 Address	192.168.10.6
Subnet Mask	255.255.255.0
Default Gateway	0.0.0.0
DNS Server	0.0.0.0

RESULT:

The experiment was executed successfully.

PROGRAM:

link-state.tcl

```
set ns [new Simulator]
$ns rtproto LS
set nf [open ls1.tr w]
$ns trace-all $nf
set nr [open ls2.nam w]
$ns namtrace-all $nr

proc finish {} {
  global ns nf nr
  $ns flush-trace
  close $nf
  close $nr
  exec nam ls2.nam
  exit 0
}

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

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

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005

set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005

set null1 [new Agent/Null]
$ns attach-agent $n3 $null1

$ns connect $udp0 $null0
$ns connect $udp1 $null1
$ns at .1 "$cbr1 start"
$ns at .2 "$cbr0 start"
$ns at 45.0 "$cbr1 stop"
$ns at 45.1 "$cbr0 stop"
$ns at 50.0 "finish"
$ns run

AWK Program for LS
BEGIN {
  print "Performance evaluation"
  send = 0
```

Date : 01/06/2023

Experiment No : 14
NS2 Simulator

AIM :

To study of NS2 and simulate Link State Protocol and Distance Vector Routing protocol in it.

THEORY :

NS (Network Simulator) is a discrete event simulator targeted at networking research. Ns provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks. It is an object-oriented, discrete event-driven simulator written in C++ and Otcl/Tcl. NS-2 can be used to implement network protocols such as TCP and UDP, traffic source behavior such as FTP, Telnet, Web, CBR, and VBR, router queues management mechanism such as Drop Tail, RED, and CBQ and routing algorithms.

```
  recv = 0
  dropped = 0
  rout = 0
}


{
  if ($1 == "+" && (($3 == "0") || ($3 == "1")) && $5 == "cbr") {
    send++
  }
  if ($1 == "r" && $4 == "3" && $5 == "cbr") {
    recv++
  }
  if ($1 == "d") {
    dropped++
  }
  if ($1 == "r" && $5 == "rtProtoLS") {
    rout++
  }
}

END {
  print "No of packets Send: " send
  print "No of packets Received: " recv
  print "No of packets dropped: " dropped
  print "No of routing packets: " rout
  NOH = rout / recv
  PDR = recv / send
  print "Normalised overhead: " NOH
  print "Packet delivery ratio: " PDR
}
```

dv.tcl :

```
set ns [new Simulator]
$ns rtproto DV
set nf [open dv1.tr w]
$ns trace-all $nf
set nr [open dv2.nam w]
$ns namtrace-all $nr

proc finish {} {
  global ns nf nr
  $ns flush-trace
  close $nf
  close $nr
  #exec nam dv2.nam
  exit 0
}

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

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

ALGORITHM :

LSR :

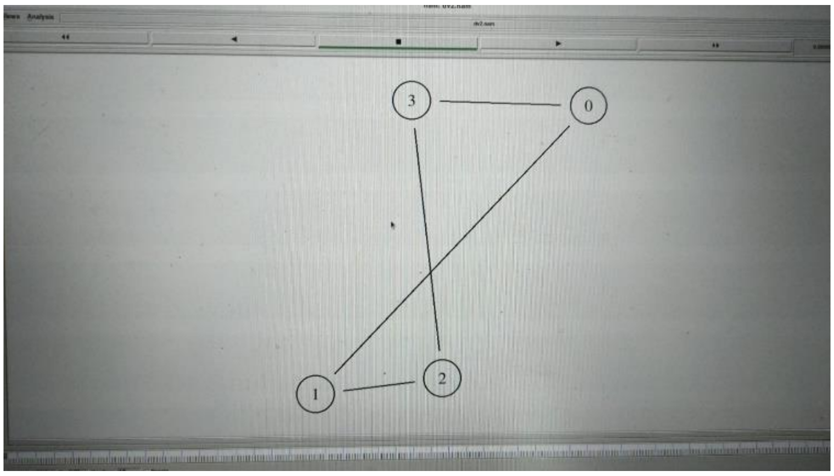
1. Initialize the network simulator (ns).
2. Set the routing protocol to Link State (LS) using "$ns rtproto LS".
3. Open trace files for capturing simulation events and nam output.
4. Define the "finish" procedure to flush traces, close files, and execute nam to display the network animation.
5. Create nodes and duplex links in the network topology.
6. Attach UDP agents and CBR traffic sources to the appropriate nodes.
7. Attach Null agents to the desired nodes.
8. Connect UDP agents to Null agents.
9. Model link failures and recoveries using "rtmodel-at" at specific time intervals.
10. Schedule the start and stop of CBR traffic sources using "$ns at".
11. Run the simulation using "$ns run".

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

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005

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

$ns connect $udp0 $null0

$ns at .1 "$cbr0 start"
$ns at 45.0 "$cbr0 stop"
$ns at 50.0 "finish"
$ns run

$ns attach-agent $n0 $udp0
AWK Program for DV

BEGIN {
  print "Performance evaluation"
  send = 0
  recv = 0
  dropped = 0
  rout = 0
}
{
  if ($1 == "+" && ($3 == "0" || $3 == "1") && $5 == "cbr") {
    send++
  }
  if ($1 == "r" && $4 == "3" && $5 == "cbr") {
    recv++
  }
  if ($1 == "d") {
    dropped++
  }
  if ($1 == "r" && $5 == "rtProtoDV") {
    rout++
  }
}
END {
  print "No of packets Send: " send
  print "No of packets Received: " recv
  print "No of packets dropped: " dropped
  print "No of routing packets: " rout
  NOH = rout / recv
  PDR = recv / send
  print "Normalised overhead: " NOH
  print "Packet delivery ratio: " PDR
}
```

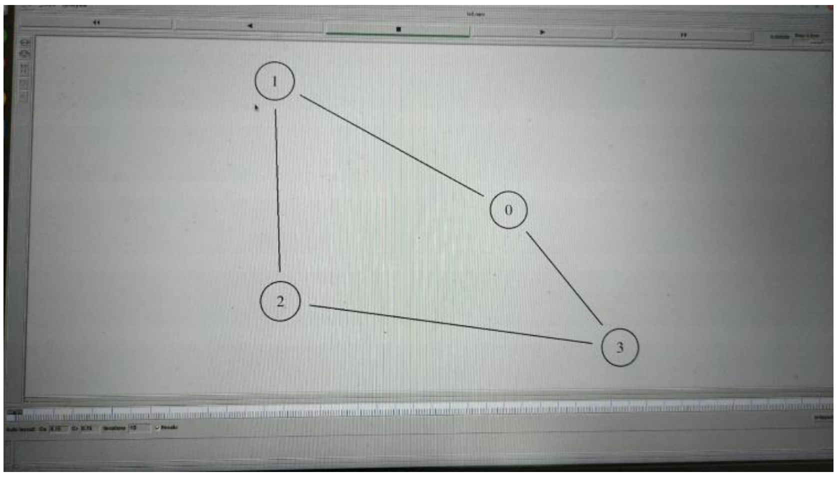
DVR :

1. Initialize the network simulator (ns).
2. Set the routing protocol to Distance Vector (DV) using "$ns rtproto DV".
3. Open trace files for capturing simulation events and nam output.
4. Define the "finish" procedure to flush traces, close files, and exit the simulation.
5. Create nodes and duplex links in the network topology.
6. Attach UDP agent and CBR traffic source to the source node.
7. Attach Null agent to the destination node.
8. Connect the UDP agent to the Null agent.
9. Schedule the start and stop of CBR traffic using "$ns at".
10. Run the simulation using "$ns run".

OUTPUT:



DVR



LSR

RESULT:

The experiment was executed successfully.