GOVT. MODEL ENGINEERING COLLEGE

THRIKKAKARA, ERNAKULAM

CSL 332 NETWORKING LAB

NAME: ABHINAND S
BRANCH: COMPUTER SCIENCE AND ENGINEERING
SEMESTER: 06
ROLL NO: 22CSA01

Certified that this is the bonafide work done by

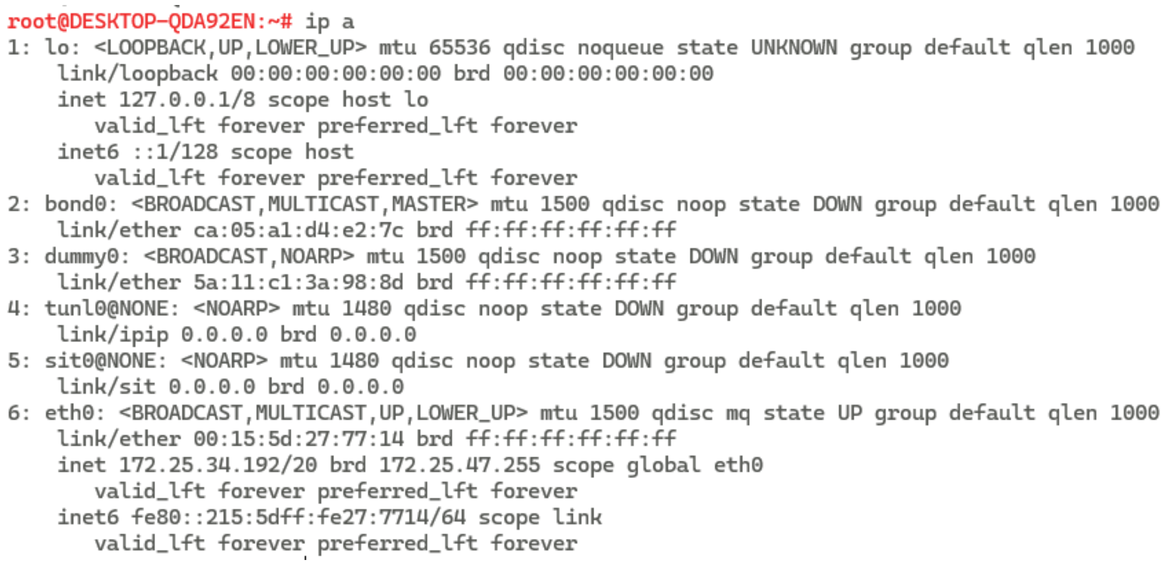
ABHINAND S

Staff-in-Charge	***Head of the department***
Register No.:....................	Date:.......................
Year and Month:.................	Thrikkakara

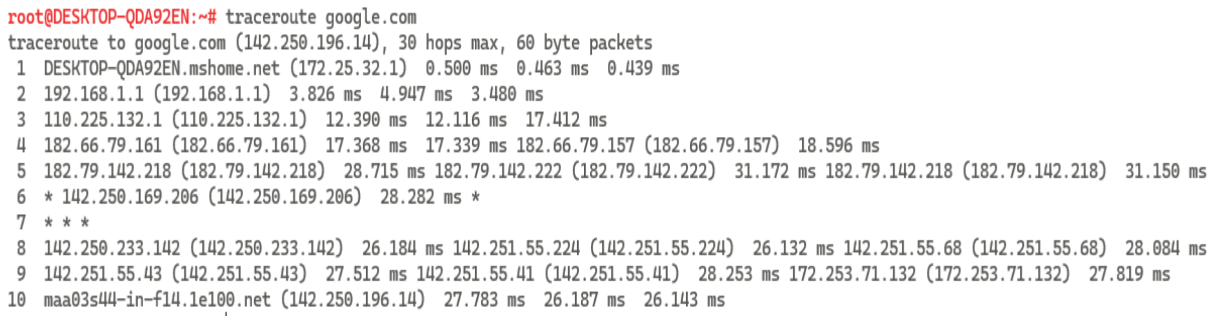
Internal Examiner **External Examiner**

INDEX

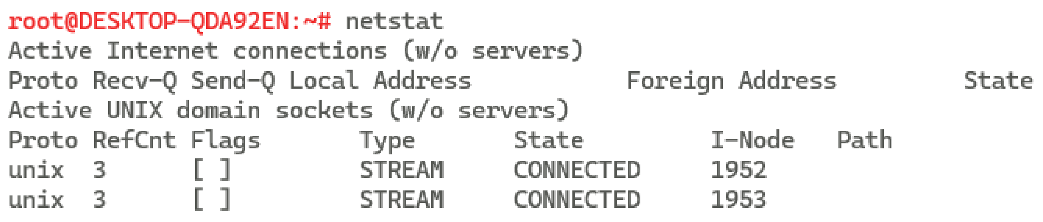
<u>OUTPUT:</u>



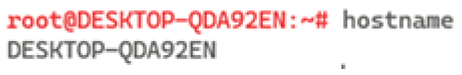
```
root@DESKTOP-QDA92EN:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: bond0: <BROADCAST,MULTICAST,MASTER> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether ca:05:a1:d4:e2:7c brd ff:ff:ff:ff:ff:ff
3: dummy0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 5a:11:c1:3a:98:8d brd ff:ff:ff:ff:ff:ff
4: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
5: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/sit 0.0.0.0 brd 0.0.0.0
6: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:15:5d:27:77:14 brd ff:ff:ff:ff:ff:ff
    inet 172.25.34.192/20 brd 172.25.47.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::215:5dff:fe27:7714/64 scope link
       valid_lft forever preferred_lft forever
```



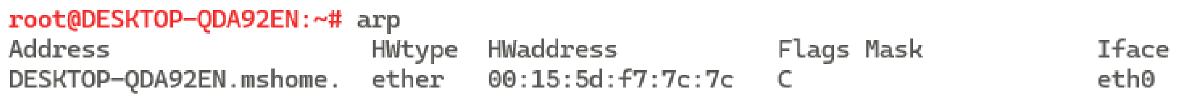
```
root@DESKTOP-QDA92EN:~# traceroute google.com
traceroute to google.com (142.250.196.14), 30 hops max, 60 byte packets
 1  DESKTOP-QDA92EN.mshome.net (172.25.32.1)  0.500 ms  0.463 ms  0.439 ms
 2  192.168.1.1 (192.168.1.1)  3.826 ms  4.947 ms  3.480 ms
 3  110.225.132.1 (110.225.132.1)  12.390 ms  12.116 ms  17.412 ms
 4  182.66.79.161 (182.66.79.161)  17.368 ms  17.339 ms 182.66.79.157 (182.66.79.157)  18.596 ms
 5  182.79.142.218 (182.79.142.218)  28.715 ms 182.79.142.222 (182.79.142.222)  31.172 ms 182.79.142.218 (182.79.142.218)  31.150 ms
 6  * 142.250.169.206 (142.250.169.206)  28.282 ms *
 7  * * *
 8  142.250.233.142 (142.250.233.142)  26.184 ms 142.251.55.224 (142.251.55.224)  26.132 ms 142.251.55.68 (142.251.55.68)  28.084 ms
 9  142.251.55.43 (142.251.55.43)  27.512 ms 142.251.55.41 (142.251.55.41)  28.253 ms 172.253.71.132 (172.253.71.132)  27.819 ms
10  maa03s44-in-f14.1e100.net (142.250.196.14)  27.783 ms  26.187 ms  26.143 ms
```



```
root@DESKTOP-QDA92EN:~# netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags       Type       State         I-Node   Path
unix  3      [ ]         STREAM     CONNECTED     1952
unix  3      [ ]         STREAM     CONNECTED     1953
```



```
root@DESKTOP-QDA92EN:~# hostname
DESKTOP-QDA92EN
```



```
root@DESKTOP-QDA92EN:~# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
DESKTOP-QDA92EN.mshome.  ether   00:15:5d:f7:7c:7c   C                     eth0
```

Name: Abhinand S Date: 16/12/24
Roll no: 22CSA01
Class: CS6A

EXPERIMENT NO: 1
NETWORK CONFIGURATION FILES AND NETWORKING COMMANDS IN LINUX

AIM:
Getting started with the basics of network configuration files and networking commands in Linux.

THEORY:

Networking Commands
Linux networking commands help configure, troubleshoot, and manage network settings.

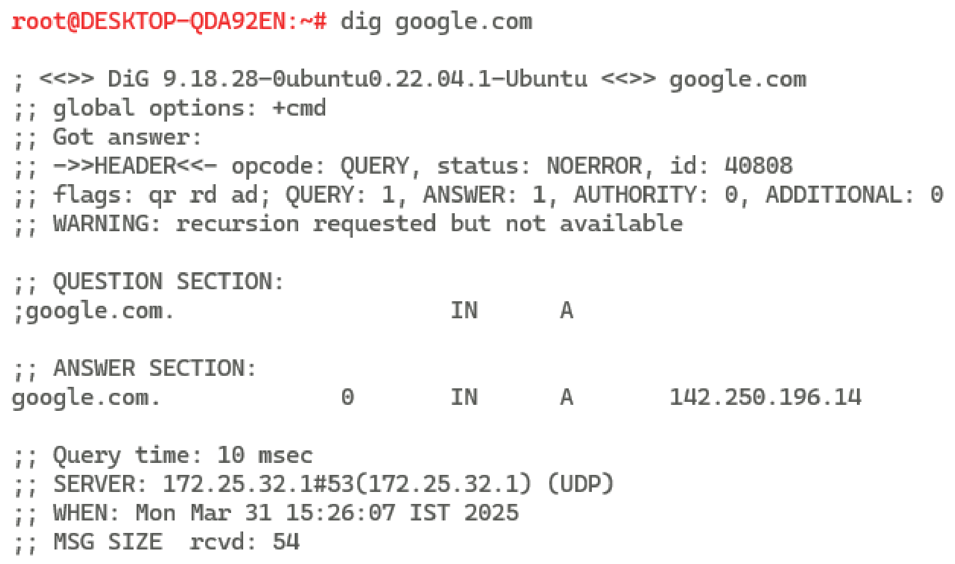
1. ip : The ip command is a powerful tool for configuring and managing network interfaces on Linux. It is widely used by administrators to display and modify IP addresses, routes, and link settings.
   Syntax - ip a

2. traceroute: The traceroute command is a network troubleshooting utility that helps determine the path packets take to reach a destination. It shows the number of hops and IP addresses of routers along the network path.
   Syntax - traceroute <domain>

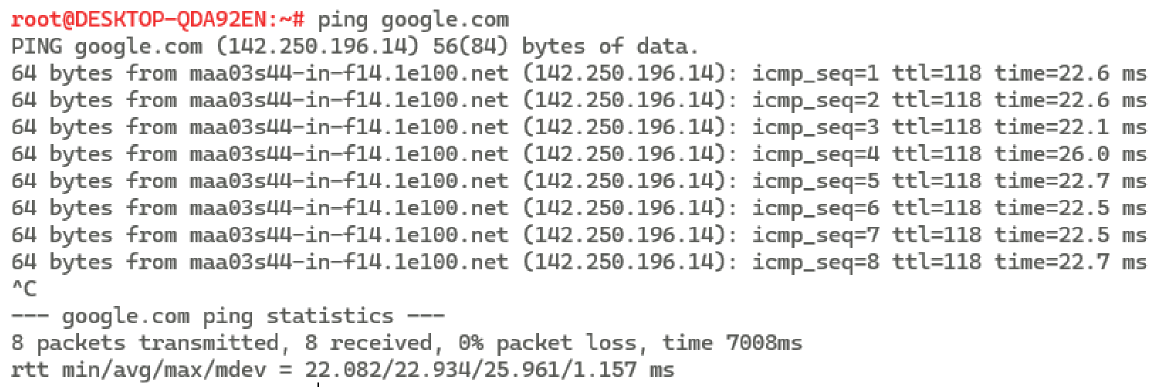
3. netstat: The netstat command displays active network connections, listening ports, routing tables, and statistics about network traffic.
   Syntax – netstat

4. hostname: The hostname command is used to display or set the system's hostname. The hostname is the name assigned to a computer on a network, allowing it to be identified by other devices.
   Syntax – hostname

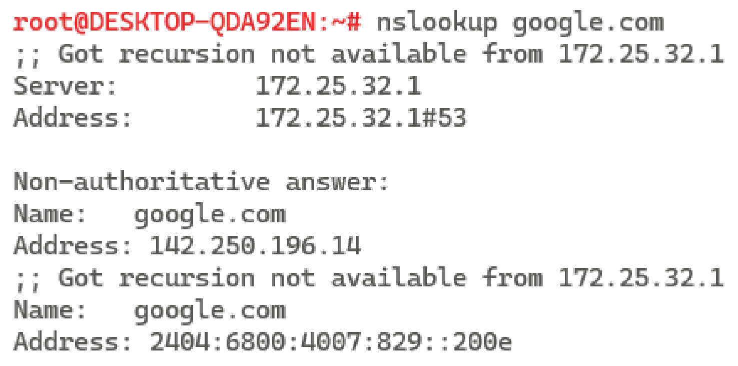
5. arp: The arp (Address Resolution Protocol) command is used in networking to view and manipulate the ARP cache.
   Syntax – arp



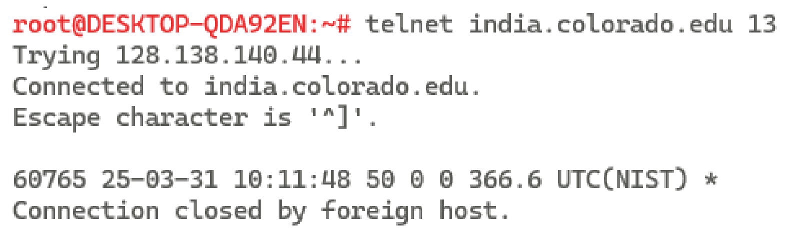
```
root@DESKTOP-QDA92EN:~# dig google.com

; <<>> DiG 9.18.28-0ubuntu0.22.04.1-Ubuntu <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 40808
;; flags: qr rd ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;google.com.                    IN      A

;; ANSWER SECTION:
google.com.             0       IN      A       142.250.196.14

;; Query time: 10 msec
;; SERVER: 172.25.32.1#53(172.25.32.1) (UDP)
;; WHEN: Mon Mar 31 15:26:07 IST 2025
;; MSG SIZE  rcvd: 54
```



```
root@DESKTOP-QDA92EN:~# ping google.com
PING google.com (142.250.196.14) 56(84) bytes of data.
64 bytes from maa03s44-in-f14.1e100.net (142.250.196.14): icmp_seq=1 ttl=118 time=22.6 ms
64 bytes from maa03s44-in-f14.1e100.net (142.250.196.14): icmp_seq=2 ttl=118 time=22.6 ms
64 bytes from maa03s44-in-f14.1e100.net (142.250.196.14): icmp_seq=3 ttl=118 time=22.1 ms
64 bytes from maa03s44-in-f14.1e100.net (142.250.196.14): icmp_seq=4 ttl=118 time=26.0 ms
64 bytes from maa03s44-in-f14.1e100.net (142.250.196.14): icmp_seq=5 ttl=118 time=22.7 ms
64 bytes from maa03s44-in-f14.1e100.net (142.250.196.14): icmp_seq=6 ttl=118 time=22.5 ms
64 bytes from maa03s44-in-f14.1e100.net (142.250.196.14): icmp_seq=7 ttl=118 time=22.5 ms
64 bytes from maa03s44-in-f14.1e100.net (142.250.196.14): icmp_seq=8 ttl=118 time=22.7 ms
^C
--- google.com ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7008ms
rtt min/avg/max/mdev = 22.082/22.934/25.961/1.157 ms
```



```
root@DESKTOP-QDA92EN:~# nslookup google.com
;; Got recursion not available from 172.25.32.1
Server:         172.25.32.1
Address:        172.25.32.1#53

Non-authoritative answer:
Name:   google.com
Address: 142.250.196.14
;; Got recursion not available from 172.25.32.1
Name:   google.com
Address: 2404:6800:4007:829::200e
```



```
root@DESKTOP-QDA92EN:~# telnet india.colorado.edu 13
Trying 128.138.140.44...
Connected to india.colorado.edu.
Escape character is '^]'.

60765 25-03-31 10:11:48 50 0 0 366.6 UTC(NIST) *
Connection closed by foreign host.
```

6. dig: The dig(Domain Information Groper) command is used to retrieve detailed DNS information, such as name servers.
   Syntax – dig <domain>

7. ping: The ping command is used to check network connectivity between two nodes. It sends ICMP (Internet Control Message Protocol) echo request packets to the destination and waits for a response.
   Syntax – ping <destination>

8. nslookup: The nslookup command is used to query DNS records, allowing users to resolve domain names into IP addresses and vice versa.
   Syntax – nslookup <domain>

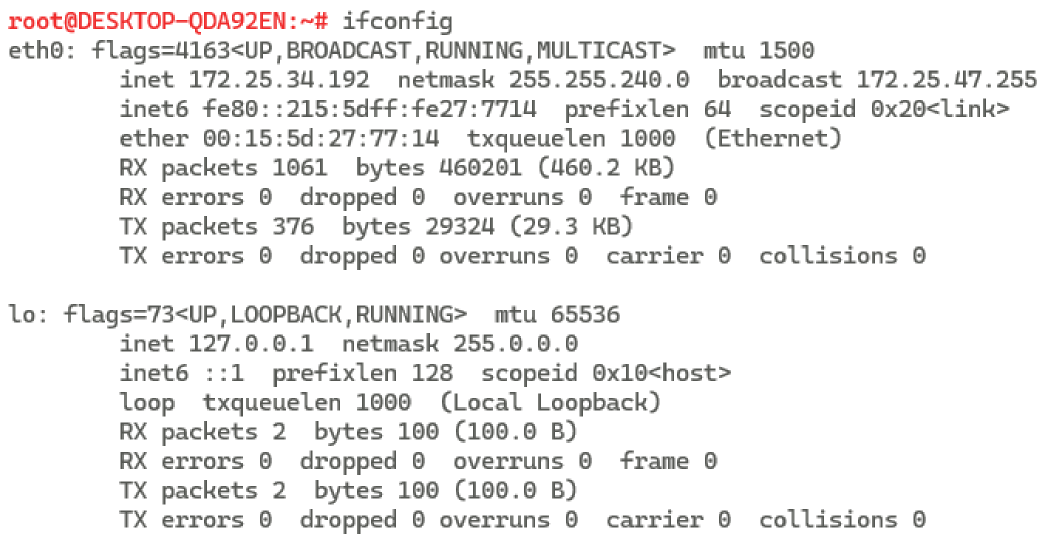
9. telnet : In Linux, the telnet command is used to create a remote connection with a system over a TCP/IP network. It allows us to administrate other systems by the terminal. We can run a program to conduct administration.
   Syntax - telnet hostname/IP address

10. 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.
    Syntax – ifconfig

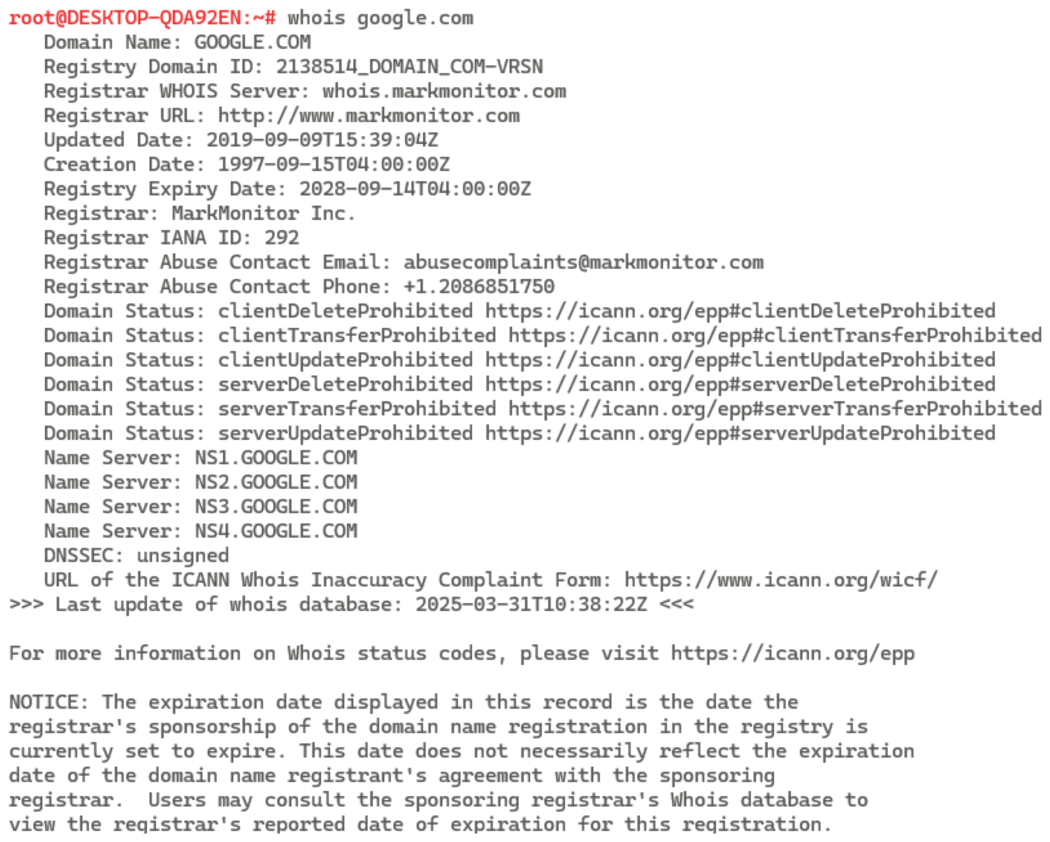
11. whois: Linux whois command is used to fetch all the information related to a website. You can get all the information about a website including the registration and the owner information.
    Syntax – whois <domain>

12. ftp: The ftp (File Transfer Protocol) command is used to transfer files between a local and a remote system. It allows users to connect to an FTP server, upload/download files, and manage directories.
    Syntax: ftp <server_address>

13. curl: The curl command is used to transfer data from or to a server using protocols like HTTP, HTTPS, FTP, etc. It is commonly used for API calls and downloading files.
    Syntax: curl <URL>



```
root@DESKTOP-QDA92EN:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.25.34.192  netmask 255.255.240.0  broadcast 172.25.47.255
        inet6 fe80::215:5dff:fe27:7714  prefixlen 64  scopeid 0x20<link>
        ether 00:15:5d:27:77:14  txqueuelen 1000  (Ethernet)
        RX packets 1061  bytes 460201 (460.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 376  bytes 29324 (29.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2  bytes 100 (100.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2  bytes 100 (100.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```



```
root@DESKTOP-QDA92EN:~# whois google.com
   Domain Name: GOOGLE.COM
   Registry Domain ID: 2138514_DOMAIN_COM-VRSN
   Registrar WHOIS Server: whois.markmonitor.com
   Registrar URL: http://www.markmonitor.com
   Updated Date: 2019-09-09T15:39:04Z
   Creation Date: 1997-09-15T04:00:00Z
   Registry Expiry Date: 2028-09-14T04:00:00Z
   Registrar: MarkMonitor Inc.
   Registrar IANA ID: 292
   Registrar Abuse Contact Email: abusecomplaints@markmonitor.com
   Registrar Abuse Contact Phone: +1.2086851750
   Domain Status: clientDeleteProhibited https://icann.org/epp#clientDeleteProhibited
   Domain Status: clientTransferProhibited https://icann.org/epp#clientTransferProhibited
   Domain Status: clientUpdateProhibited https://icann.org/epp#clientUpdateProhibited
   Domain Status: serverDeleteProhibited https://icann.org/epp#serverDeleteProhibited
   Domain Status: serverTransferProhibited https://icann.org/epp#serverTransferProhibited
   Domain Status: serverUpdateProhibited https://icann.org/epp#serverUpdateProhibited
   Name Server: NS1.GOOGLE.COM
   Name Server: NS2.GOOGLE.COM
   Name Server: NS3.GOOGLE.COM
   Name Server: NS4.GOOGLE.COM
   DNSSEC: unsigned
   URL of the ICANN Whois Inaccuracy Complaint Form: https://www.icann.org/wicf/
>>> Last update of whois database: 2025-03-31T10:38:22Z <<<

For more information on Whois status codes, please visit https://icann.org/epp

NOTICE: The expiration date displayed in this record is the date the
registrar's sponsorship of the domain name registration in the registry is
currently set to expire. This date does not necessarily reflect the expiration
date of the domain name registrant's agreement with the sponsoring
registrar.  Users may consult the sponsoring registrar's Whois database to
view the registrar's reported date of expiration for this registration.
```

Network Configuration Files

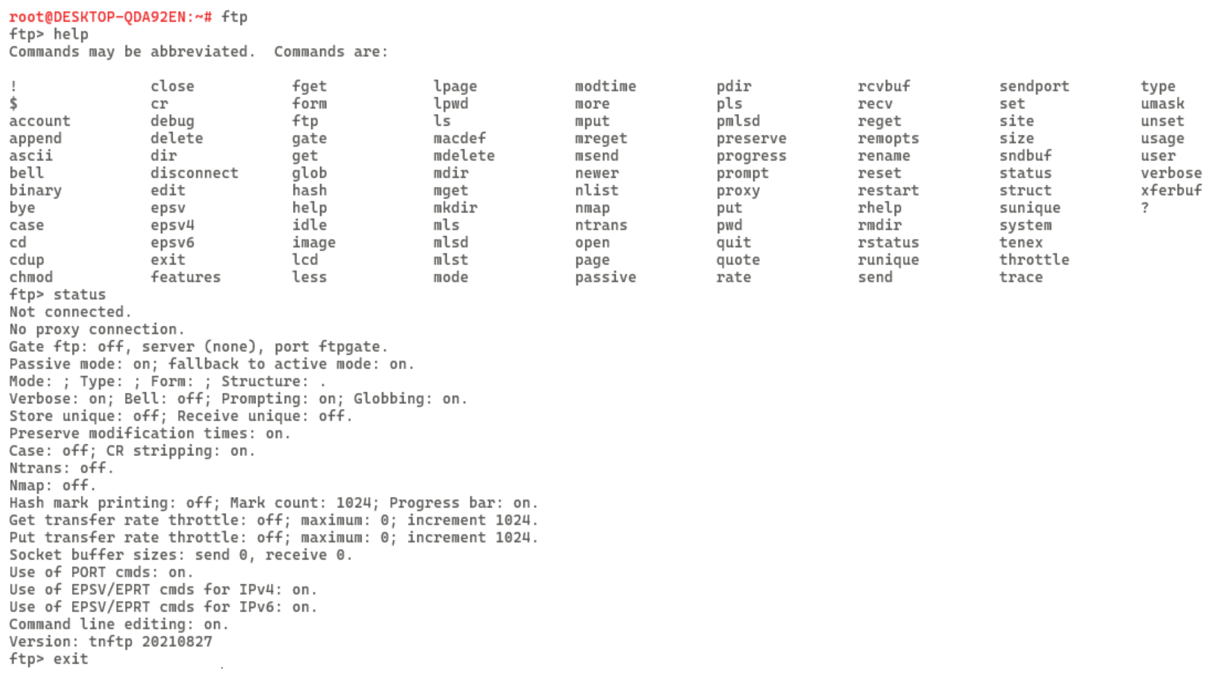
To store IP addresses and other related settings, Linux uses a separate configuration file for each network interface. All these configuration files are stored in the /etc/sysconfig/network-scripts directory.The important linux network configuration files are:

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.

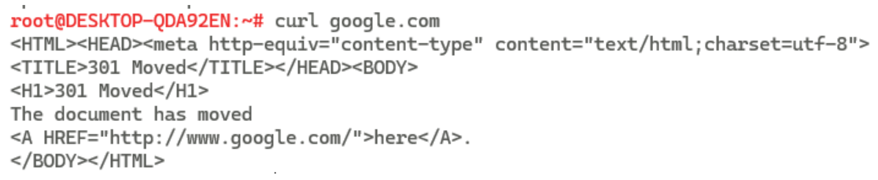
2. /etc/resolv.conf : The /etc/resolv.conf configuration file specifies the IP addresses of DNS servers and the search domain.

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



```
root@DESKTOP-QDA92EN:~# ftp
ftp> help
Commands may be abbreviated.  Commands are:

!               close           fget            lpage           modtime         pdir            rcvbuf          sendport        type
$               cr              form            lpwd            more            pls             recv            set             umask
account         debug           ftp             ls              mput            pmlsd           reget           site            unset
append          delete          gate            macdef          mreget          preserve        remopts         size            usage
ascii           dir             get             mdelete         msend           progress        rename          sndbuf          user
bell            disconnect      glob            mdir            newer           prompt          reset           status          verbose
binary          edit            hash            mget            nlist           proxy           restart         struct          xferbuf
bye             epsv            help            mkdir           nmap            put             rhelp           sunique         ?
case            epsv4           idle            mls             ntrans          pwd             rmdir           system
cd              epsv6           image           mlsd            open            quit            rstatus         tenex
cdup            exit            lcd             mlst            page            quote           runique         throttle
chmod           features        less            mode            passive         rate            send            trace
ftp> status
Not connected.
No proxy connection.
Gate ftp: off, server (none), port ftpgate.
Passive mode: on; fallback to active mode: on.
Mode: ; Type: ; Form: ; Structure: .
Verbose: on; Bell: off; Prompting: on; Globbing: on.
Store unique: off; Receive unique: off.
Preserve modification times: on.
Case: off; CR stripping: on.
Ntrans: off.
Nmap: off.
Hash mark printing: off; Mark count: 1024; Progress bar: on.
Get transfer rate throttle: off; maximum: 0; increment 1024.
Put transfer rate throttle: off; maximum: 0; increment 1024.
Socket buffer sizes: send 0, receive 0.
Use of PORT cmds: on.
Use of EPSV/EPRT cmds for IPv4: on.
Use of EPSV/EPRT cmds for IPv6: on.
Command line editing: on.
Version: tnftp 20210827
ftp> exit
```



```
root@DESKTOP-QDA92EN:~# curl google.com
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
```

RESULT:
Familiarized with the basics of network configuration files and networking commands in Linux.

Name: Abhinand S Date: 16/12/24
Roll no: 22CSA01
Class: CS6A

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() :
   listen() 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:

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

PROGRAM CODE:

Server

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>

int main() {
   int sockfd, newsock;
   struct sockaddr_in server, client;
   socklen_t len = sizeof(client);
   char buffer[1024];

   sockfd = socket(AF_INET, SOCK_STREAM, 0);
   server.sin_family = AF_INET;
   server.sin_addr.s_addr = INADDR_ANY;
   server.sin_port = htons(8080);

   bind(sockfd, (struct sockaddr *)&server, sizeof(server));
   listen(sockfd, 5);
   printf("Waiting for connection...\n");

   newsock = accept(sockfd, (struct sockaddr *)&client, &len);
   read(newsock, buffer, sizeof(buffer));
   printf("Message: %s\n", buffer);

   send(newsock, "Hello from server", 17, 0);
   close(newsock);
   close(sockfd);

   return 0;
}
```

Name: Abhinand S Date: 27/01/25
Roll no: 22CSA01
Class: CS6A

EXPERIMENT NO: 3
TRANSMISSION CONTROL PROTOCOL

AIM:
To implement client-server communication using socket programming and TCP as the transport layer protocol.

ALGORITHM: SERVER

```
1. Start the server program.
2. Include necessary libraries such as <stdio.h>, <string.h>, <stdlib.h>, <unistd.h>, and <arpa/inet.h>.
3. Create a TCP socket using `socket(AF_INET, SOCK_STREAM, 0)`. This sets up an endpoint for communication.
4. Configure the server address structure:
   - Set the address family to `AF_INET` (IPv4).
   - Set the IP address to `INADDR_ANY` (bind to any local address).
   - Set the port number using `htons(8080)`.
5. Bind the socket to the server address using `bind()`. This associates the socket with the specified port.
6. Set up the server to listen for incoming connections using `listen()`. This prepares the server to accept clients.
7. Accept a client connection using `accept()`, which waits for a client to connect.
8. Receive data from the client using `recv()` and store it in a buffer.
9. Process the received message (print it or perform operations).
10. Send a response back to the client using `send()`.
11. Close the client socket and the server socket.
12. Stop the server.
```

ALGORITHM: CLIENT

```
1. Start the client program.
2. Include necessary libraries such as <stdio.h>, <string.h>, <stdlib.h>, <unistd.h>, and <arpa/inet.h>.
3. Create a TCP socket using `socket(AF_INET, SOCK_STREAM, 0)`.
```

Client Code

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>

int main() {
    int sockfd;
    struct sockaddr_in server;
    char buffer[1024] = "Hello Server";

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    server.sin_family = AF_INET;
    server.sin_port = htons(8080);
    server.sin_addr.s_addr = inet_addr("127.0.0.1");

    connect(sockfd, (struct sockaddr *)&server, sizeof(server));
    send(sockfd, buffer, strlen(buffer), 0);
    read(sockfd, buffer, sizeof(buffer));
    printf("Response: %s\n", buffer);
    close(sockfd);

    return 0;
}
```

OUTPUT:

```
$ ./server
Waiting for connection...
Message: Hello Server

$ ./client
Response: Hello from server
```

4. Configure the server address structure:
 - Set the address family to `AF_INET` (IPv4).
 - Set the IP address of the server (e.g., "127.0.0.1").
 - Set the port number using `htons(8080)`.
5. Establish a connection to the server using `connect()`. This initiates communication.
6. Send data to the server using `send()`.
7. Wait for a response from the server using `recv()` and store the received data.
8. Print the server's response.
9. Close the client socket.
10. Stop the client program.

RESULT:

The program to implement client-server communication using socket programming and TCP as the transport layer protocol is successfully completed.

PROGRAM CODE:

UDP Server (udp_server.c):

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>

int main() {
    int sockfd;
    struct sockaddr_in server, client;
    char buffer[1024];
    socklen_t len = sizeof(client);

    // Create UDP socket
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);

    // Server address configuration
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(8080);

    // Bind the socket to the server address
    bind(sockfd, (struct sockaddr *)&server, sizeof(server));

    printf("Waiting for data...\n");

    // Receive data from client
    recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr *)&client, &len);
    printf("Message from client: %s\n", buffer);

    // Respond to client
    sendto(sockfd, "Hello from server", 18, 0, (struct sockaddr *)&client, len);

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

Name: Abhinand S
Date:27/01/25
Roll no: 22CSA01
Class: CS6A

EXPERIMENT NO: 4
USER DATAGRAM PROTOCOL

AIM:
To Implement client-server communication using socket programming and UDP as transport layer protocol.

ALGORITHM:

Server :

```
S0: Start
S1: Include Necessary Headers and declare variables:
        sockfd – Socket descriptor for the server,
        struct sockaddr_in server, client – Structures for storing server and client
        addresses,
        char buffer[1024] – Buffer to store received data,
        socklen_t len – Variable to store the size of the client's address.
S2: Create UDP Socket:
        Call socket(AF_INET, SOCK_DGRAM, 0) to create a UDP socket.
        If the socket creation fails (returns -1), handle the error.
S3: Set Up Server Address:
        Set the server's address:
        server.sin_family = AF_INET,
        server.sin_addr.s_addr = INADDR_ANY,
        server.sin_port = htons(8080).
S4: Bind the Socket to the Server Address:
        Call bind(sockfd, (struct sockaddr *)&server, sizeof(server)); to bind the socket
        to the specified address and port.
        If binding fails (returns -1), handle the error.
S5: Wait for Data from Client:
        Print "Waiting for data..." to indicate that the server is ready.
        Call recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr *)&client, &len);
        to wait for a message from the client and store the data in buffer.
S6: Display the Message:
        Print the received message using printf("Message from client: %s\n", buffer);.
```

UDP Client (udp_client.c):

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>

int main() {
   int sockfd;
   struct sockaddr_in server;
   char buffer[1024] = "Hello from client";
   char response[1024];
   socklen_t len = sizeof(server);

   // Create UDP socket
   sockfd = socket(AF_INET, SOCK_DGRAM, 0);

   // Server address configuration
   server.sin_family = AF_INET;
   server.sin_port = htons(8080);
   server.sin_addr.s_addr = inet_addr("127.0.0.1");  // Server IP address

   // Send message to server
   sendto(sockfd, buffer, strlen(buffer), 0, (struct sockaddr *)&server, len);
   printf("Message sent to server\n");

   // Receive response from server
   recvfrom(sockfd, response, sizeof(response), 0, (struct sockaddr *)&server, &len);
   printf("Message from server: %s\n", response);

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

OUTPUT:

```
Server Terminal:
Waiting for data...
Message from client: Hello from client

Client Terminal:
Message sent to server
Message from server: Hello from server
S7: Respond to Client:
        Call sendto(sockfd, "Hello from server", 17, 0, (struct sockaddr *)&client, len);
    to send a response message to the client.
S8: Close the socket using close(sockfd) after communication is finished.
S9: Stop
```

Client :

```
S1: Include Necessary Headers and Declare Variables
        sockfd – Client socket descriptor.
        struct sockaddr_in server – Structure for storing server address.
        char buffer[1024] = "Hello from client"; – Message to send to the server.
        char response[1024]; – Buffer to store the server's response.
        socklen_t len – Variable to store the size of the server's address.
S2: Create UDP Socket:
        Call socket(AF_INET, SOCK_DGRAM, 0) to create a UDP socket.
        If the socket creation fails (returns -1), handle the error.
S3: Set Up Server Address:
        server.sin_family = AF_INET,
        server.sin_port = htons(8080),
        server.sin_addr.s_addr = inet_addr("127.0.0.1").
S4: Send Data to Server:
        Call sendto(sockfd, buffer, strlen(buffer), 0, (struct sockaddr *)&server, len); to
        send the message to the server.
        Print "Message sent to server" to indicate the message has been sent.
S5: Receive Response from Server:
        Call recvfrom(sockfd, response, sizeof(response), 0, (struct sockaddr
        *)&server, &len); to wait for the server's response and store it in response.
        Print the server's response using printf("Message from server: %s\n",
        response);.
S6: Close the Socket using close(sockfd) after communication is finished.
S7: Stop
```

<u>RESULT:</u>

The program to Implement client-server communication using socket programming and UDP as transport layer protocol is successfully completed.

PROGRAM CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#define TIMEOUT 3
#define TOTAL_PACKETS 5

int simulate_acknowledgment() {
return rand() % 10 < 7;
}
int main() {
srand(time(0));
int packet = 1;
int ack_received;
while (packet <= TOTAL_PACKETS) {
            printf("Sender: Sending packet %d...\n", packet);
            sleep(1);
             ack_received = simulate_acknowledgment();
            if (ack_received) {
                     printf("Receiver: ACK for packet %d received.\n\n", packet);
                     packet++;
            } else {
                     printf("Receiver: ACK for packet %d lost! Retransmitting...\n\n", packet);
                     sleep(TIMEOUT);
                     }
            }
            printf("All packets sent successfully!\n");
            return 0;
   }
```

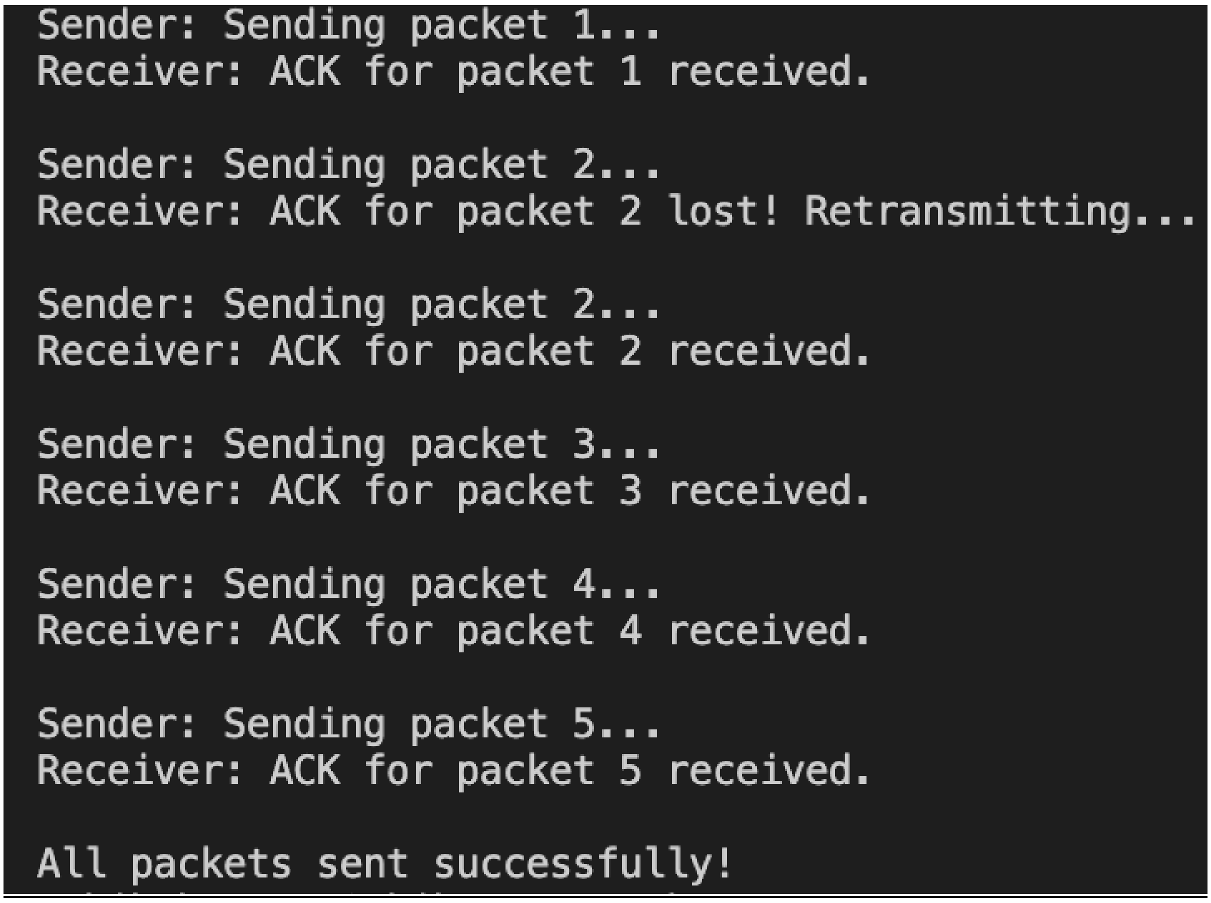
Name: Abhinand S Date:03/02/25
Roll no: 22CSA01
Class: CS6A

EXPERIMENT NO: 5
STOP AND WAIT PROTOCOL

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

ALGORITHM:
1. Start.
2. Initialize TOTAL_PACKETS = 5, TIMEOUT = 3 seconds, and packet = 1.
3. Seed the random number generator.
4. While packet ≤ TOTAL_PACKETS, repeat steps 5–9.
5. Send packet and print a message.
6. Wait for 1 second to simulate transmission delay.
7. Check if ACK is received using a random function.
8. If ACK is received, print acknowledgment and move to the next packet.
9. If ACK is lost, print loss message, wait for TIMEOUT seconds, and retransmit the same packet.
10. End after all packets are successfully transmitted.

OUTPUT:



```
Sender: Sending packet 1...
Receiver: ACK for packet 1 received.

Sender: Sending packet 2...
Receiver: ACK for packet 2 lost! Retransmitting...

Sender: Sending packet 2...
Receiver: ACK for packet 2 received.

Sender: Sending packet 3...
Receiver: ACK for packet 3 received.

Sender: Sending packet 4...
Receiver: ACK for packet 4 received.

Sender: Sending packet 5...
Receiver: ACK for packet 5 received.

All packets sent successfully!
```

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

PROGRAM CODE:
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

#define TOTAL_PACKETS 10
#define WINDOW_SIZE 4

int simulate_acknowledgment() {
    return rand() % 10 < 7;
}

int main() {
    srand(time(0));

    int base = 1;
    int next_seq_num = 1;

    while (base <= TOTAL_PACKETS) {
    printf("\n--- Sending window ---\n");
    while (next_seq_num < base + WINDOW_SIZE && next_seq_num <=
TOTAL_PACKETS) {
            printf("Sender: Sending packet %d\n", next_seq_num);
            next_seq_num++;
    }

    sleep(1);

    printf("\n--- Waiting for ACKs ---\n");
    for (int i = base; i < next_seq_num; i++) {
            if (simulate_acknowledgment()) {
            printf("ACK for packet %d received \n", i);
            } else {
            printf("ACK for packet %d lost \n", i);
            printf("Retransmitting from packet %d...\n", i);
            next_seq_num = i;
            break;
            }
    }
```

Name: Abhinand S   Date: 03/02/25
Roll no: 22CSA01
Class: CS6A

EXPERIMENT NO: 6
GO BACK N PROTOCOL

AIM:

To implement and analyze the Go-Back-N Automatic Repeat Request (ARQ) protocol.

ALGORITHM:

1. Start
2. Initialize
   * Define TOTAL_PACKETS (total number of packets to send).
   * Define WINDOW_SIZE (number of packets sent before waiting for ACKs).
   * Set base = 1 (starting packet of the window).
   * Set next_seq_num = 1 (next packet to send).
3. Send Packets
   * Send packets within the current window [base, base + WINDOW_SIZE - 1].
4. Simulate Acknowledgment
   * Randomly determine if a packet's acknowledgment (ACK) is received.
5. Process Acknowledgments
   * If an ACK is received, move the window forward.
   * If an ACK is lost, retransmit all packets from the lost packet onward.
6. Repeat Steps 3-5 until all packets are successfully acknowledged.
7. Stop

```
    base = next_seq_num;
    sleep(1);
    }

    printf("\nAll packets sent and acknowledged successfully! \n");
    return 0;
}
```

OUTPUT:

```
--- Sending window ---
Sender: Sending packet 1
Sender: Sending packet 2
Sender: Sending packet 3
Sender: Sending packet 4

--- Waiting for ACKs ---
ACK for packet 1 received
ACK for packet 2 received
ACK for packet 3 lost
Retransmitting from packet 3...
--- Waiting for ACKs ---
ACK for packet 3 received
ACK for packet 4 received

--- Sending window ---
Sender: Sending packet 5
Sender: Sending packet 6
Sender: Sending packet 7
Sender: Sending packet 8

--- Waiting for ACKs ---
ACK for packet 5 received
ACK for packet 6 received
ACK for packet 7 lost
Retransmitting from packet 7…
```

```
--- Waiting for ACKs ---
ACK for packet 7 received
ACK for packet 8 received

--- Sending window ---
Sender: Sending packet 9
Sender: Sending packet 10

--- Waiting for ACKs ---
ACK for packet 9 received
ACK for packet 10 received

All packets sent and acknowledged successfully!
```

RESULT:

The Go-Back-N ARQ experiment has been successfully completed with all packets transmitted and acknowledged, with retransmissions for lost packets as per the protocol.

PROGRAM CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>

#define WINDOW_SIZE 4 // Sliding window size
#define TOTAL_FRAMES 10 // Total number of frames to send
#define LOSS_PROBABILITY 20 // Percentage probability of frame loss (0-100)

// Simulate frame transmission
int send_frame(int frame_number) {
printf("Sending frame %d...\n", frame_number);
sleep(1); // Simulate delay
int rand_value = rand() % 100; // Generate random number between 0 and 99
if (rand_value < LOSS_PROBABILITY) {
printf("Frame %d lost during transmission!\n", frame_number);
return 0;
}
printf("Frame %d sent successfully.\n", frame_number);
return 1;
}

// Simulate receiving acknowledgment
int receive_ack(int frame_number) {
printf("Receiving acknowledgment for frame %d...\n", frame_number);
sleep(1); // Simulate delay
int rand_value = rand() % 100;
if (rand_value < LOSS_PROBABILITY) {
printf("Acknowledgment for frame %d lost!\n", frame_number);
return 0;
}
printf("Acknowledgment for frame %d received.\n", frame_number);
return 1;
}

// Selective Repeat ARQ Protocol
void selective_repeat_arq() {
int sent_frames[TOTAL_FRAMES] = {0}; // Track sent frames
int ack_received[TOTAL_FRAMES] = {0}; // Track acknowledgments received
```

Name: Abhinand S Date: 17/02/25
Roll no: 22CSA01
Class: CS6A

EXPERIMENT NO: 7
SELECTIVE REPEAT PROTOCOL

AIM:
To write a program to simulate the Selective Repeat protocol

ALGORITHM:
1.Start
2.Initialization
- Set up a window size (N) to control how many frames can be sent before receiving acknowledgments.
- Initialize counters for sent frames and received acknowledgments.

3.Send Frames
- Transmit frames within the current sliding window.
- Each frame is assigned a sequence number.

4.Simulate Frame Loss (Optional)
- Frames may be randomly lost during transmission (simulated by probability).

5.Receive Acknowledgments
- The receiver acknowledges individual frames as they arrive.
- Out-of-order frames are buffered until missing frames are received.

6.Retransmit Lost Frames
- If an acknowledgment for a frame is not received, only that specific frame is retransmitted.

7.Slide the Window
- Once the lowest-numbered frame (base) is acknowledged, the window slides forward.
- The sender can now transmit new frames.

8.Repeat Until Completion
- The process continues until all frames are sent and acknowledged.

9.stop.

```
int base = 0; // Start of sliding window

while (base < TOTAL_FRAMES) {
// Send frames within the window
for (int i = base; i < base + WINDOW_SIZE && i < TOTAL_FRAMES; i++) {
if (!sent_frames[i]) {
sent_frames[i] = send_frame(i); // Send frame if not sent
}
}


// Check for acknowledgments
for (int i = base; i < base + WINDOW_SIZE && i < TOTAL_FRAMES; i++) {
if (sent_frames[i] && !ack_received[i]) {
ack_received[i] = receive_ack(i); // Mark acknowledgment if received
}
}


// Slide the window if the base frame is acknowledged
while (base < TOTAL_FRAMES && ack_received[base]) {
printf("Sliding window forward. Frame %d fully acknowledged.\n", base);
base++;
}
}
printf("All frames sent and acknowledged successfully.\n");
}


// Main function
int main() {
srand(time(0)); // Seed random number generator
selective_repeat_arq();
return 0;
}
```

OUTPUT:

```
Sending frame 0...
Frame 0 sent successfully.
Sending frame 1...
Frame 1 lost during transmission!
Sending frame 2…
```

```
Frame 2 lost during transmission!
Sending frame 3...
Frame 3 sent successfully.
Receiving acknowledgment for frame 0...
Acknowledgment for frame 0 received.
Receiving acknowledgment for frame 3...
Acknowledgment for frame 3 lost!
Sliding window forward. Frame 0 fully acknowledged.
Sending frame 1...
Frame 1 sent successfully.
Sending frame 2...
Frame 2 lost during transmission!
Sending frame 4...
Frame 4 sent successfully.
Receiving acknowledgment for frame 1...
Acknowledgment for frame 1 received.
Receiving acknowledgment for frame 3...
Acknowledgment for frame 3 lost!
Receiving acknowledgment for frame 4...
Acknowledgment for frame 4 received.
Sliding window forward. Frame 1 fully acknowledged.
Sending frame 2...
Frame 2 lost during transmission!
Sending frame 5...
Frame 5 sent successfully.
Receiving acknowledgment for frame 3...
Acknowledgment for frame 3 received.
Receiving acknowledgment for frame 5...
Acknowledgment for frame 5 received.
Sending frame 2...
Frame 2 sent successfully.
Receiving acknowledgment for frame 2...
Acknowledgment for frame 2 received.
Sliding window forward. Frame 2 fully acknowledged.
Sliding window forward. Frame 3 fully acknowledged.
Sliding window forward. Frame 4 fully acknowledged.
Sliding window forward. Frame 5 fully acknowledged.
Sending frame 6...
Frame 6 sent successfully.
Sending frame 7...
Frame 7 sent successfully.
Sending frame 8…
```

```
Frame 8 lost during transmission!
Sending frame 9...
Frame 9 sent successfully.
Receiving acknowledgment for frame 6...
Acknowledgment for frame 6 received.
Receiving acknowledgment for frame 7...
Acknowledgment for frame 7 received.
Receiving acknowledgment for frame 9...
Acknowledgment for frame 9 lost!
Sliding window forward. Frame 6 fully acknowledged.
Sliding window forward. Frame 7 fully acknowledged.
Sending frame 8...
Frame 8 sent successfully.
Receiving acknowledgment for frame 8...
Acknowledgment for frame 8 received.
Receiving acknowledgment for frame 9...
Acknowledgment for frame 9 lost!
Sliding window forward. Frame 8 fully acknowledged.
Receiving acknowledgment for frame 9...
Acknowledgment for frame 9 received.
Sliding window forward. Frame 9 fully acknowledged.
All frames sent and acknowledged successfully.
```

RESULT:
The simulation of selective repeat ARQ using C has been implemented and output has been obtained successfully.

PROGRAM CODE:

```
#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");
```

Name: Abhinand S Date: 17/02/25
Roll no: 22CSA01
Class: CS6A

EXPERIMENT NO: 8
DISTANCE VECTOR ROUTING

AIM:
Write a program to implement and simulate Distance Vector Routing protocol.

ALGORITHM:
S1: Start by defining the structure for the routing table, which includes arrays for cost and next hop information for each router.
S2: Initialize variables: cost matrix 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.
S3: Read the number of routers from the user.
S4:Read the cost matrix from the user, populating the cost matrix array and initializing the routing table arrays.
S5: Perform the distance vector algorithm until no more changes occur:
S6: 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).

S7: 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]).

S8: End the program

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
```

RESULT:

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

PROGRAM CODE:

FTP Server (ftp_server.c):

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>

    #define PORT 8080
    #define BUFFER_SIZE 1024

    void handle_client(int client_socket) {
    char buffer[BUFFER_SIZE] = {0};
    char filename[BUFFER_SIZE] = {0};

    // Receive filename from client

    int bytes_received = recv(client_socket, filename, BUFFER_SIZE, 0);
    if (bytes_received < 0) {
    perror("Error receiving filename");
    close(client_socket);
    return;
    }

    // Open the requested file

    FILE *file = fopen(filename, "r");
    if (file == NULL) {
    char *error_message = "File not found";
    send(client_socket, error_message, strlen(error_message), 0);
    perror("File not found");
    } else {

    // Read file and send its content

    while (fgets(buffer, BUFFER_SIZE, file) != NULL) {
    send(client_socket, buffer, strlen(buffer), 0);
    memset(buffer, 0, BUFFER_SIZE);
    }

    printf("File Send to client\n");
```

Name: Abhinand S Date: 24/02/25
Roll no: 22CSA01
Class: CS6A

EXPERIMENT NO: 9
FILE TRANSFER PROTOCOL

AIM:
To Implement file transfer protocol.

ALGORITHM:

Server :

S0: Start

S1: Include Necessary Headers and Declare Variables

- Include <stdio.h>, <stdlib.h>, <string.h>, <unistd.h>, and <arpa/inet.h>.
- Declare:
  - o server_socket – Socket descriptor for the server.
  - o client_socket – Socket descriptor for the client.
  - o struct sockaddr_in server_addr, client_addr – Structures for storing server and client addresses.
  - o socklen_t addr_len – Variable to store the size of the client's address.
  - o char buffer[1024] – Buffer to store file data.
  - o char filename[1024] – Buffer to store the requested filename.

S2: Create TCP Socket

- Call socket(AF_INET, SOCK_STREAM, 0) to create a TCP socket.
- If the socket creation fails (returns -1), print an error message and exit.

S3: Set Up Server Address

- Assign values to server_addr:
  - o server_addr.sin_family = AF_INET (Use IPv4).
  - o server_addr.sin_addr.s_addr = INADDR_ANY (Bind to all available interfaces).
  - o server_addr.sin_port = htons(8080) (Convert port number to network byte order).

S4: Bind the Socket to the Server Address

- Call bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) to bind the socket to the specified address and port.
- If binding fails, print an error message and exit.

S5: Start Listening for Connections

- Call listen(server_socket, 3) to listen for incoming client connections.

```
fclose(file);}
close(client_socket);
}
int main() {
int server_socket, client_socket;
struct sockaddr_in server_addr, client_addr;
socklen_t addr_len = sizeof(client_addr);

// Create server socket
server_socket = socket(AF_INET, SOCK_STREAM, 0);
if (server_socket == 0) {
perror("Socket failed");
exit(EXIT_FAILURE);
}

// Configure server address
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = INADDR_ANY;
server_addr.sin_port = htons(PORT);

// Bind socket to address
if (bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0)
{
perror("Bind failed");
exit(EXIT_FAILURE);
}
// Start listening
if (listen(server_socket, 3) < 0) {
perror("Listen failed");
exit(EXIT_FAILURE);
}
printf("Server is listening on port %d\n", PORT);
while (1) {
// Accept client connection
client_socket = accept(server_socket, (struct sockaddr *)&client_addr,
&addr_len);
if (client_socket < 0) {
perror("Accept failed");
exit(EXIT_FAILURE);
}
printf("Client connected\n");
```

- If listening fails, print an error message and exit.
- Print "Server is listening on port 8080".

S6: Accept Client Connection in a Loop
- Start an infinite loop:
    - Call accept(server_socket, (struct sockaddr *)&client_addr, &addr_len).
    - If accept fails, print an error message and exit.
    - Print "Client connected".

S7: Handle Client Request
- Call recv(client_socket, filename, BUFFER_SIZE, 0) to receive the requested filename from the client.
- If recv fails, print "Error receiving filename" and close the socket.

S8: Open Requested File
- Call fopen(filename, "r") to open the requested file.
- If the file does not exist:
    - Send "File not found" to the client.
    - Print "File not found" and close the client socket.
- If the file exists:
    - Read file content using fgets in a loop.
    - Send file content to the client using send(client_socket, buffer, strlen(buffer), 0).
    - Clear the buffer after each iteration using memset(buffer, 0, BUFFER_SIZE).
    - Print "File sent to client".

S9: Close File and Client Socket
- Close the file using fclose(file).
- Close the client socket using close(client_socket).
- Go back to Step S6 to accept the next client.

S10: Close Server Socket (After Loop Ends)
- Close the server socket using close(server_socket).
- End.

<u>Client:</u>

S0: Start

S1: Include Necessary Headers and Declare Variables
- Include <stdio.h>, <stdlib.h>, <string.h>, <unistd.h>, and <arpa/inet.h>.
- Declare:
    - client_socket – Socket descriptor for the client.
    - struct sockaddr_in server_addr – Structure for storing server address.
    - char buffer[1024] – Buffer to store received data.
    - char filename[1024] – Buffer to store the filename.

```
}
printf("Server is listening on port %d\n", PORT);
while (1) {
// Accept client connection
client_socket = accept(server_socket, (struct sockaddr *)&client_addr,
&addr_len);
if (client_socket < 0) {
perror("Accept failed");
exit(EXIT_FAILURE);
}
printf("Client connected\n");
// Handle client
handle_client(client_socket);
}
close(server_socket);
return 0;
}
```

<u>FTP Client (ftp_client.c)</u>

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
int client_socket;
struct sockaddr_in server_addr;
char buffer[BUFFER_SIZE] = {0};
char filename[BUFFER_SIZE] = {0};

// Create client socket

client_socket = socket(AF_INET, SOCK_STREAM, 0);
if (client_socket < 0) {
  perror("Socket creation failed");
  exit(EXIT_FAILURE);
  }
```

S2: Create TCP Socket

- Call socket(AF_INET, SOCK_STREAM, 0) to create a TCP socket.
- If the socket creation fails (returns -1), print an error message and exit.

S3: Set Up Server Address

- Assign values to server_addr:
    - o server_addr.sin_family = AF_INET (Use IPv4).
    - o server_addr.sin_port = htons(8080) (Convert port number to network byte order).
    - o server_addr.sin_addr.s_addr = inet_addr("127.0.0.1") (Set server IP to localhost).

S4: Connect to the Server

- Call connect(client_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)).
- If connection fails, print "Connection failed" and exit.

S5: Get Filename from User

- Print "Enter the filename to request: ".
- Read the filename using scanf("%s", filename).

S6: Send Filename to Server

- Call send(client_socket, filename, strlen(filename), 0) to send the filename to the server.

S7: Receive and Display File Content

- Print "File content received:".
- Start a loop:
    - o Call recv(client_socket, buffer, BUFFER_SIZE, 0).
    - o If data is received, print the buffer.
    - o Clear the buffer using memset(buffer, 0, BUFFER_SIZE).
- When no more data is received, exit the loop.

S8: Close Client Socket and End

- Close the client socket using close(client_socket).
- End.

If the socket creation fails (returns -1), handle the error.

```
// Configure server address
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(PORT);
server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

// Connect to server
if (connect(client_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
perror("Connection failed");
exit(EXIT_FAILURE);

}
// Get filename from user
printf("Enter the filename to request: ");
scanf("%s", filename);

// Send filename to server
send(client_socket, filename, strlen(filename), 0);

// Receive and print file content

printf("File content received:\n");
while (recv(client_socket, buffer, BUFFER_SIZE, 0) > 0) {
printf("%s", buffer);
memset(buffer, 0, BUFFER_SIZE);
}

printf("\n");
close(client_socket);

return 0;
}
```

OUTPUT:

```
Server is listening on port 8080
Client connected
File Sent to client

Enter the filename to request: example.txt
File content received:
Hello World!
```

RESULT:

The program to Implement client-server communication using socket programming and UDP as transport layer protocol is successfully completed.

PROGRAM CODE:

```
#include <stdio.h>
#include <stdlib.h>

void leaky_bucket(int bucket_capacity, int leak_rate, int num_packets, int packets[]) {
   int bucket = 0; // Current bucket level
   printf("Time\tIncoming\tBucket\tLeaked\tRemaining\n");
   for (int i = 0; i < num_packets; i++) {
      printf("%d%10d", i + 1, packets[i]);

      // Add incoming packets to the bucket
      bucket += packets[i];
      if (bucket > bucket_capacity) {
         printf("%10d(Overflowed, Dropped %d)", bucket_capacity, bucket -
bucket_capacity);
         bucket = bucket_capacity; // Discard excess packets
      } else {
         printf("%10d", bucket);
      }

      // Leak out packets at the constant rate
      int leaked = (bucket >= leak_rate) ? leak_rate : bucket;
      bucket -= leaked;

      printf("%10d%10d\n", leaked, bucket);
   }

 int time = num_packets + 1;
   while (bucket > 0) {
      int leaked = (bucket >= leak_rate) ? leak_rate : bucket;
      printf("%d%10d%10d%10d%10d\n", time,0,bucket, leaked, bucket - leaked);
      bucket -= leaked;
      time++;
   }
}
int main() {
   int bucket_capacity, leak_rate, num_packets;
   printf("Enter the bucket capacity: ");
 scanf("%d", &bucket_capacity);

   printf("Enter the leak rate: ");
   scanf("%d", &leak_rate);
```

Name: Abhinand S Date: 05/03/25
Roll no: 22CSA01
Class: CS6A

EXPERIMENT NO: 10
LEAKY BUCKET

AIM:
To implement congestion control using a leaky bucket algorithm

ALGORITHM:

Step 1: Input Parameters

- Read the bucket capacity, leak rate, and number of packets.
- Read the sizes of each incoming packet.

Step 2: Initialize Variables

- Set bucket = 0 (to keep track of the current amount of data in the bucket).
- Print table headers: Time, Incoming, Bucket, Leaked, Remaining.

Step 3: Process Incoming Packets

- Loop through each incoming packet:
  - o Add the packet size to the bucket.
  - o If the bucket exceeds its capacity:
    - § Display an overflow message and drop the excess.
    - § Set bucket = bucket_capacity.
  - o Else, print the updated bucket size.
  - o Leak packets at the defined rate (leak_rate), ensuring that the leaked amount is not greater than the current bucket level.
- int packets[num_packets];
  - o Update the bucket level after leakage and print results.

```
printf("Enter the number of packets: ");
   scanf("%d", &num_packets);

    int packets[num_packets];

            printf("Enter the size of each incoming packet:\n");

            for (int i = 0; i < num_packets; i++) {

            scanf("%d", &packets[i]);

            }


            printf("\nLeaky Bucket Simulation:\n");

            leaky_bucket(bucket_capacity, leak_rate, num_packets, packets);


            return 0;

    }
```

Step 4: Empty the Bucket After All Packets are Processed

- Continue leaking packets until the bucket is empty, printing the status at each step.

Step 5: Termination

- End the program

OUTPUT:

```
Enter the bucket capacity: 10

Enter the leak rate: 4

Enter the number of packets: 5

Enter the size of each incoming packet:

5 8 4 3 6

Leaky Bucket Simulation:

Time   Incoming      Bucket         Leaked         Remaining

1      5      5      4      1

2      8      9      4      5

3      4      9      4      5

4      3      8      4      4

5      6      10      4      6

6      0      6      4      2

7      0      2      2      0
```

<u>RESULT:</u>
Successfully simulated congestion control using leaky bucket algorithm.

PROGRAM:

```
set ns [new Simulator]

set nf [open out.nam w]
$ns namtrace-all $nf
set nf1 [open out.tr w]
$ns trace-all $nf1

proc finish { } {
   global ns nf nf1
   $ns flush-trace

   close $nf
   close $nf1

   exec nam out.nam &
   exit 0
}
set n0 [$ns node]
set n1 [$ns node]

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

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

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

set null0 [new Agent/Null]
$ns attach-agent $n1 $null0

$ns connect $udp0 $null0

$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
$ns at 5.0 "finish"

$ns run
```

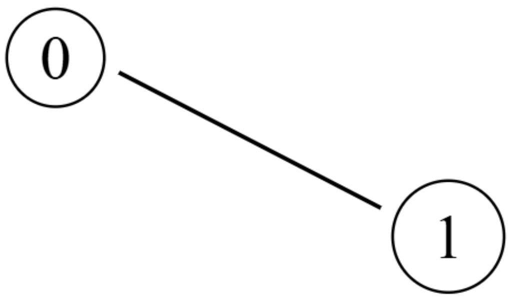
Name: Abhinand S Date: 30/12/24
Roll no: 22CSA01
Class: CS6A

EXPERIMENT NO: 11
NS2 SIMULATOR

AIM:
To study NS2 and to simulate wired network topology using NS2

THEORY:
NS2 (Network Simulator 2) is a discrete-event, object-oriented network simulator widely used for research and education in networking. Developed at UC Berkeley, it supports the simulation of LAN and WAN networks and is implemented using C++ for performance and OTcl (Object Tool Command Language) for scripting and control. NS2 provides built-in support for various network protocols like TCP and UDP, traffic models such as FTP, CBR, and VBR, and queue management techniques including DropTail and RED. It also supports routing algorithms and MAC layer protocols, making it a comprehensive tool for analyzing network behavior. The event scheduler in NS2 manages packet transmissions and network events, ensuring accurate simulation of network dynamics. Users create simulations using OTcl scripts, defining network topology, traffic flows, and timing events, while C++ handles lower-level packet processing to optimize efficiency.
NS2's plumbing mechanism links network objects, making configuration easy. It includes tools like NAM (Network Animator) for visualization and generates trace files for analysis. By combining OTcl for control and C++ for performance, NS2 enables researchers to test network protocols and performance efficiently, making it a valuable tool for studying network communication.

OUTPUT:



PROGRAM:

```
set ns [new Simulator]
$ns color 1 Blue
$ns color 2 Red

set file1 [open out.tr w]
$ns trace-all $file1
set file2 [open out.nam w]
$ns namtrace-all $file2

proc finish {} {
   global ns file1 file2
   $ns flush-trace
   close $file1
   close $file2
   exec nam out.nam &
   exit 0
}
```

a) First Simulation Scenario

ALGORITHM:

1. Initialize the network simulator using set ns [new Simulator].
2. Open trace files for capturing simulation events (out.tr) and NAM output (out.nam).
3. Define the "finish" procedure to flush traces, close files, execute NAM, and exit the simulation.
4. Create two nodes using $ns node.
5. Establish a duplex link between the nodes with 1 Mbps bandwidth, 10 ms delay, and DropTail queue.
6. Create a UDP agent and attach it to the source node (n0).
7. Create a CBR traffic source, set its parameters (packet size and interval), and attach it to the UDP agent.
8. Create a Null agent and attach it to the destination node (n1).
9. Connect the UDP agent to the Null agent, forming a data flow between n0 and n1.
10. Schedule events for starting and stopping the CBR traffic at 0.5s and 4.5s, respectively.
11. Call the finish procedure at 5.0s to terminate the simulation and launch NAM.
12. Run the simulation using $ns run. Initialize the network simulator using set ns [new Simulator].
13. Open trace files for capturing simulation events (out.tr) and NAM output (out.nam).
14. Define the "finish" procedure to flush traces, close files, execute NAM, and exit the simulation.
15. Create two nodes using $ns node.
16. Establish a duplex link between the nodes with 1 Mbps bandwidth, 10 ms delay, and DropTail queue.
17. Create a UDP agent and attach it to the source node (n0).
18. Create a CBR traffic source, set its parameters (packet size and interval), and attach it to the UDP agent.
19. Create a Null agent and attach it to the destination node (n1).
20. Connect the UDP agent to the Null agent, forming a data flow between n0 and n1.
21. Schedule events for starting and stopping the CBR traffic at 0.5s and 4.5s, respectively.
22. Call the finish procedure at 5.0s to terminate the simulation and launch NAM.
23. Run the simulation using $ns run.

```
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

$ns at 0.1 "$n1 label CBR"
$ns at 1.0 "$n0 label FTP"

$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 2Mb 10ms DropTail
$ns simplex-link $n3 $n2 0.3Mb 100ms DropTail
$ns simplex-link $n2 $n3 0.3Mb 100ms DropTail
$ns duplex-link $n3 $n4 0.5Mb 40ms DropTail
$ns duplex-link $n5 $n3 0.5Mb 30ms DropTail
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n3 $n2 orient left
$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link-op $n5 $n3 orient right-down

$ns queue-limit $n2 $n3 40
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n4 $sink

$ns connect $tcp $sink
$tcp set fid_ 1
$tcp set window_ 8000
$tcp set packetSize_ 552

set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP
```

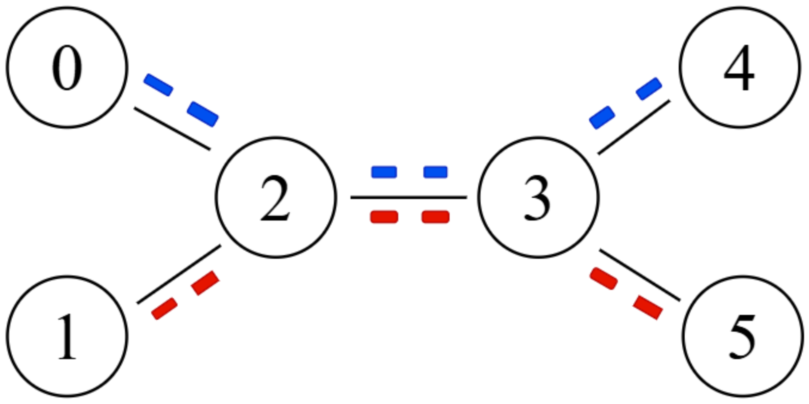
b) Second Simulation Scenario

ALGORITHM:

1. Initialize the network simulator using set ns [new Simulator].
2. Open trace files for capturing simulation events (out.tr) and NAM output (out.nam).
3. Define the "finish" procedure to flush traces, close files, execute NAM, and exit the simulation.
4. Create six nodes using $ns node.
5. Assign labels to nodes (e.g., CBR to n1 and FTP to n0).
6. Establish links between nodes:
7. Duplex links: $n0 - $n2, $n2 - $n3, $n3 - $n4, $n5 - $n3 with different bandwidth, delay, and DropTail queue.
8. Simplex links: $n3 -> $n2, $n2 -> $n3 with 0.3 Mbps bandwidth, 100 ms delay, and DropTail queue.
9. Set node positions in NAM for visualization.
10. Set queue size of link (n2-n3) to 40 packets.
11. Create a TCP agent and attach it to the source node (n0).
12. Create a TCP Sink agent and attach it to the destination node (n4).
13. Connect the TCP agent to the TCP Sink agent, forming a data flow.
14. Configure TCP parameters such as congestion window, packet size, and flow ID.
15. Create an FTP application, attach it to the TCP agent, and set its type to FTP.
16. Create a UDP agent and attach it to the source node (n1).
17. Create a Null agent and attach it to the destination node (n5).
18. Connect the UDP agent to the Null agent, forming a UDP data flow.
19. Create a CBR traffic source, set its parameters (packet size, rate, and randomization), and attach it to the UDP agent.
20. Schedule the start of CBR and FTP traffic at 0.1s and 1.0s, respectively.
21. Schedule the stop of CBR and FTP traffic at 624.5s and 624.0s, respectively.
22. Enable TCP congestion window and RTT tracing, attaching them to a trace file.
23. Schedule the finish procedure at 625.0s to terminate the simulation and launch NAM.
24. Run the simulation using $ns run.

```
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_ 2

set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 0.01mb
$cbr set random_ false

$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 624.0 "$ftp stop"
$ns at 624.5 "$cbr stop"
set file [open cwnd_rtt.tr w]
$tcp attach $file
$tcp trace cwnd_
$tcp trace rtt_

$ns at 625.0 "finish"
$ns run
```

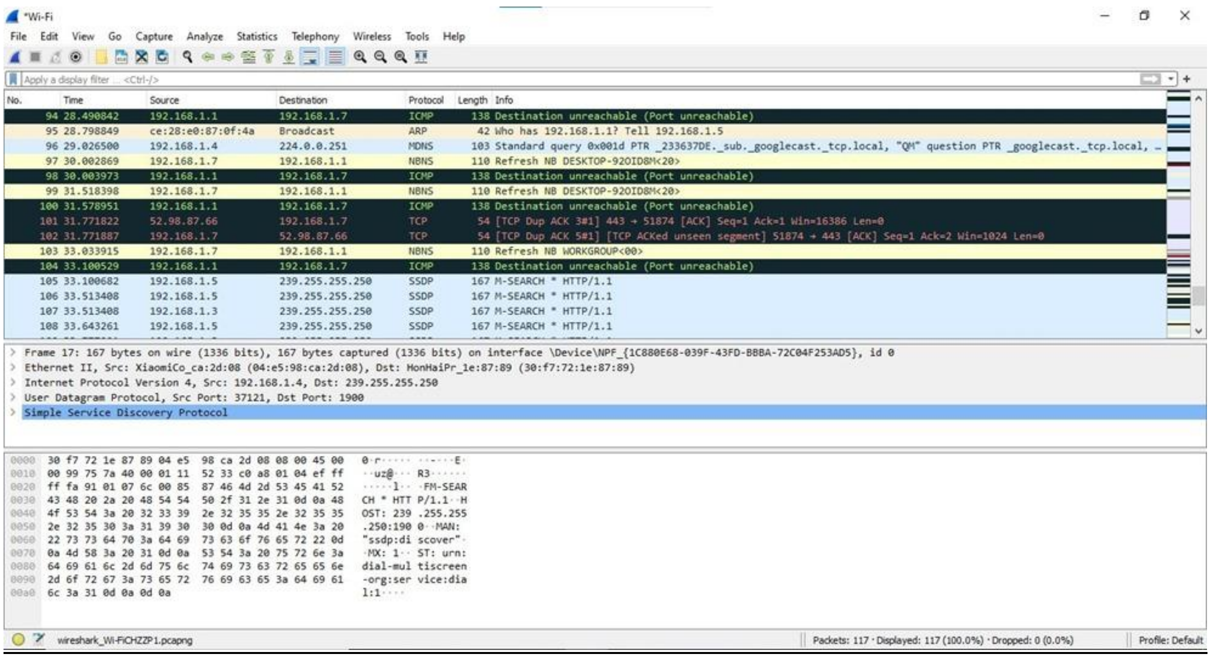
OUTPUT:



RESULT:

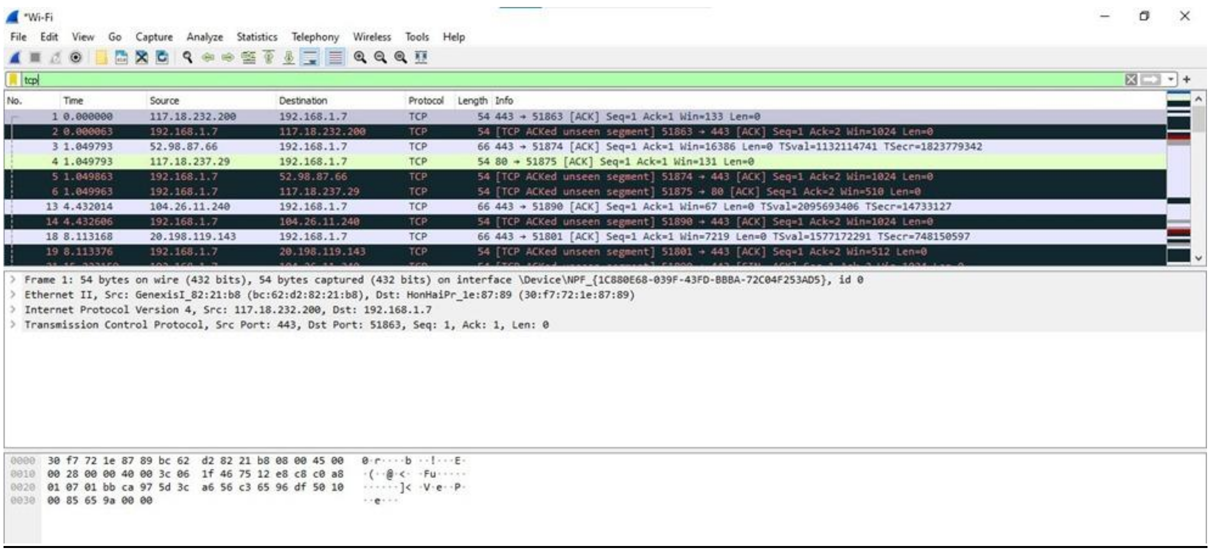
The experiment was executed successfully

OUTPUT:


*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
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:


*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
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···
```

Name: Abhinand S Date: 02/04/25
Roll no: 22CSA01
Class: CS6A

EXPERIMENT NO: 12
WIRESHARK

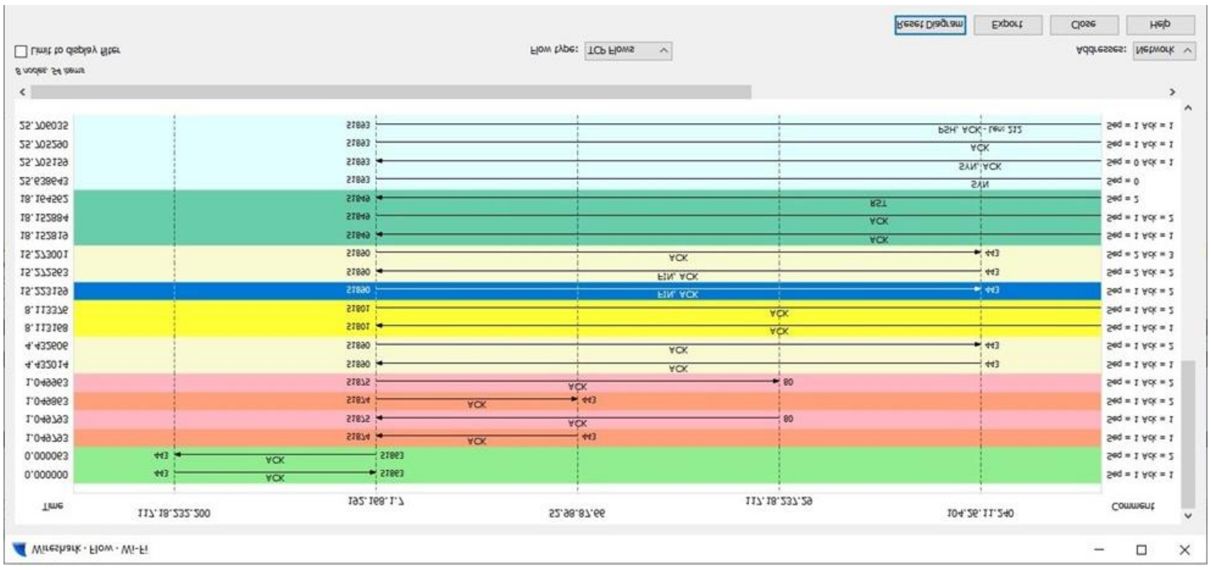
AIM:
To understand Wireshark tool and explore its features like filters, flow graphs, statistics and protocol hierarchy.

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.

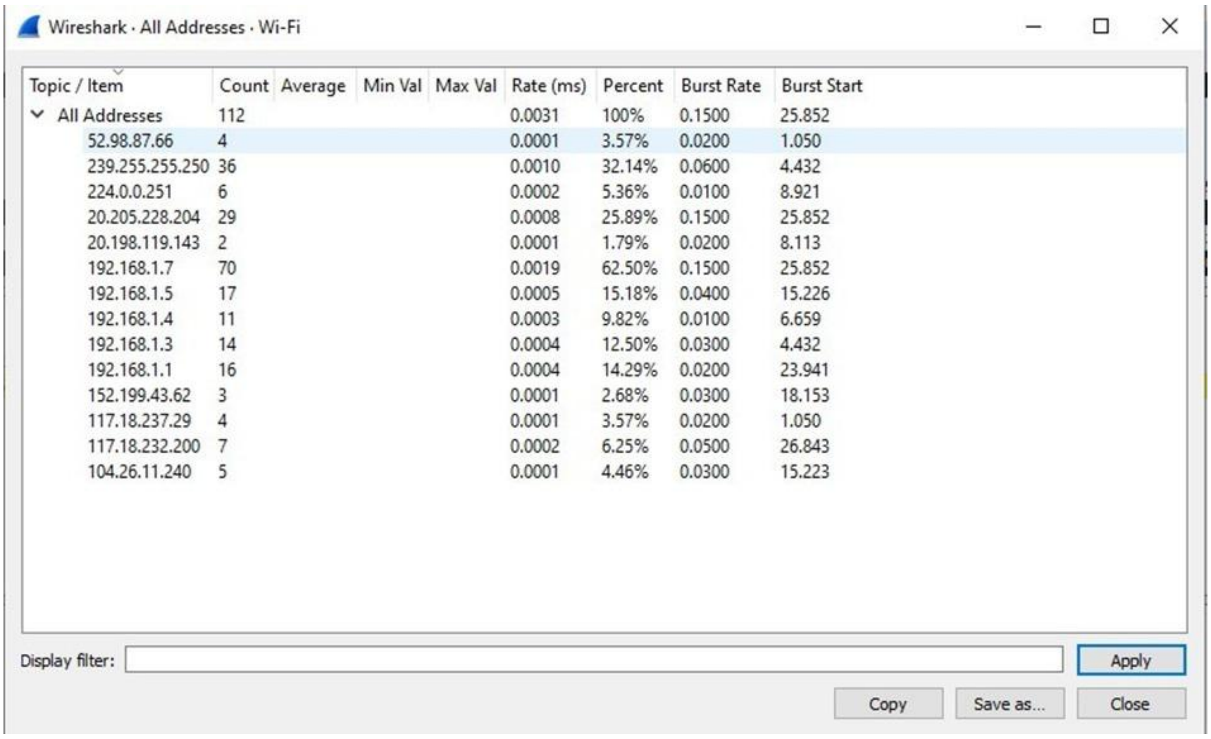
Filters :
Wireshark share a powerful filter engine that helps remove the noise from a packet trace and lets you see only the packets that interest you. If a packet meets the requirements expressed in your filter, then it is displayed in the list of packets. Display filters let you compare the fields 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:



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:

Apply

Copy Save as... Close

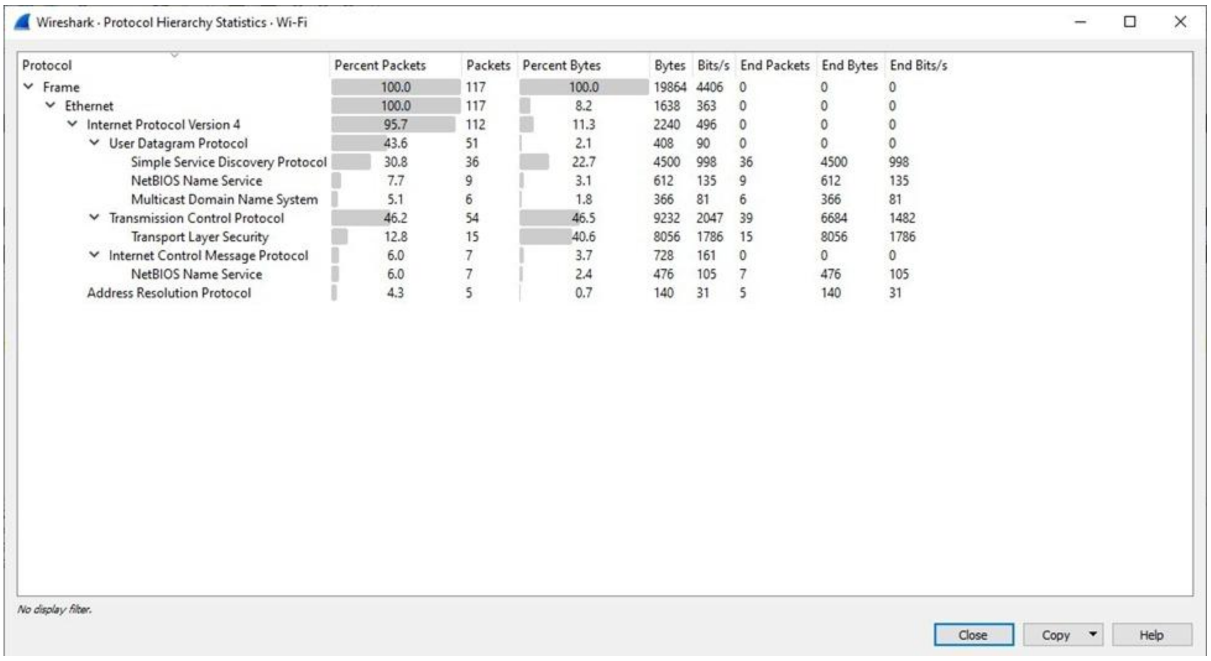
Statistics:

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

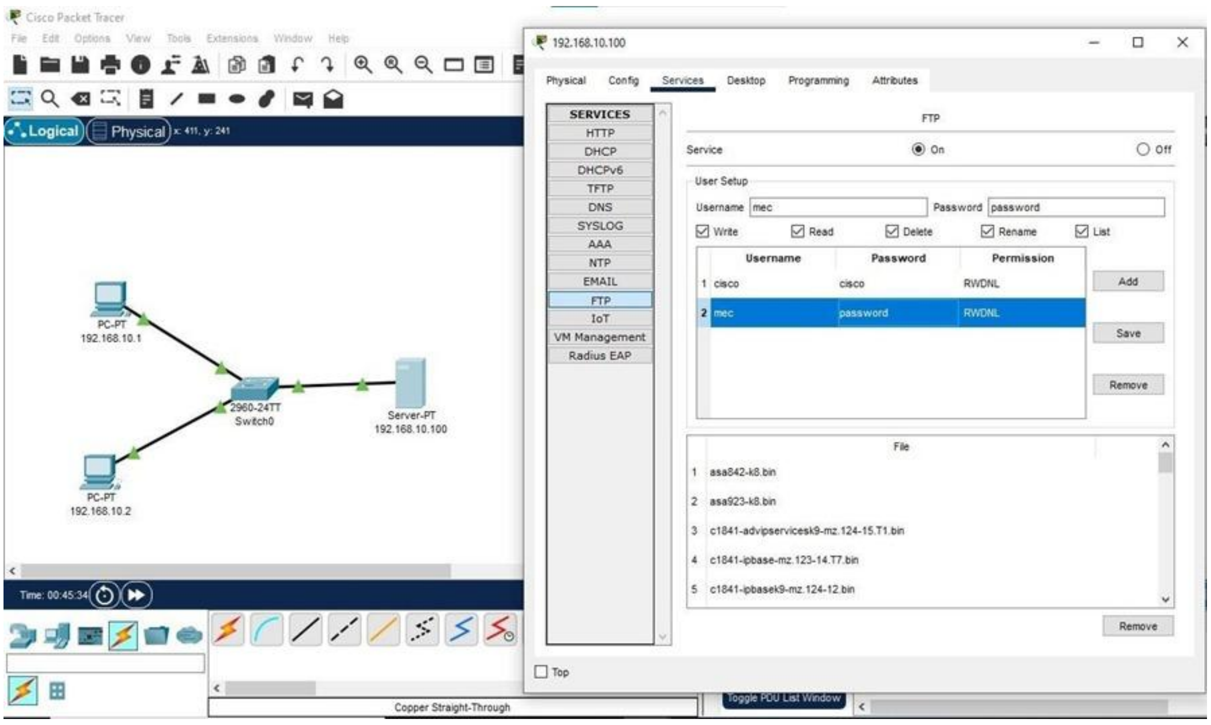
Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	117	100.0	19864	4406	0	0	0
Ethernet	100.0	117	8.2	1638	363	0	0	0
Internet Protocol Version 4	95.7	112	11.3	2240	496	0	0	0
User Datagram Protocol	43.6	51	2.1	408	90	0	0	0
Simple Service Discovery Protocol	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:
Understood Wireshark tool and explored its features like filters, flow graphs, statistics and protocol hierarchy.

OUTPUT:

FTP:



FTP

Service [✓ On] Off

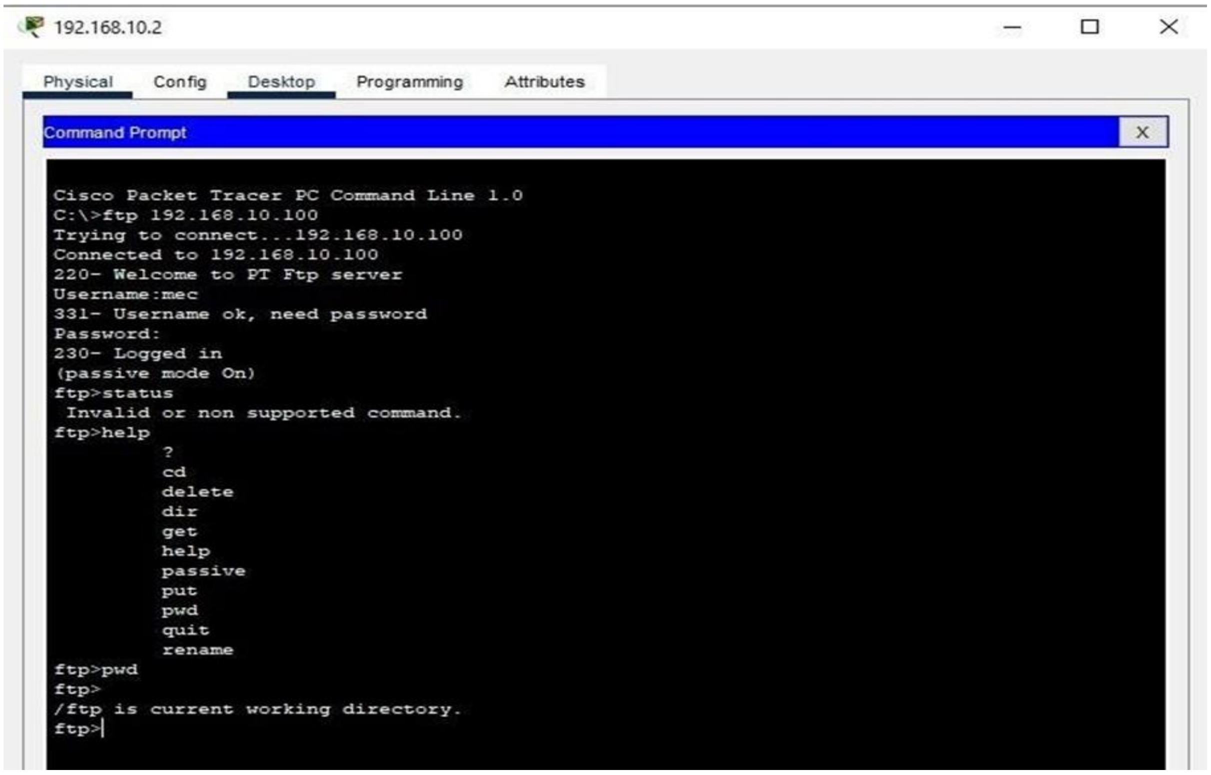
User Setup

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>
```

Name: Abhinand S Date: 02/04/25
Roll no: 22CSA01
Class: CS6A

EXPERIMENT NO: 13
NETWORK WITH MULTIPLE SUBNETS

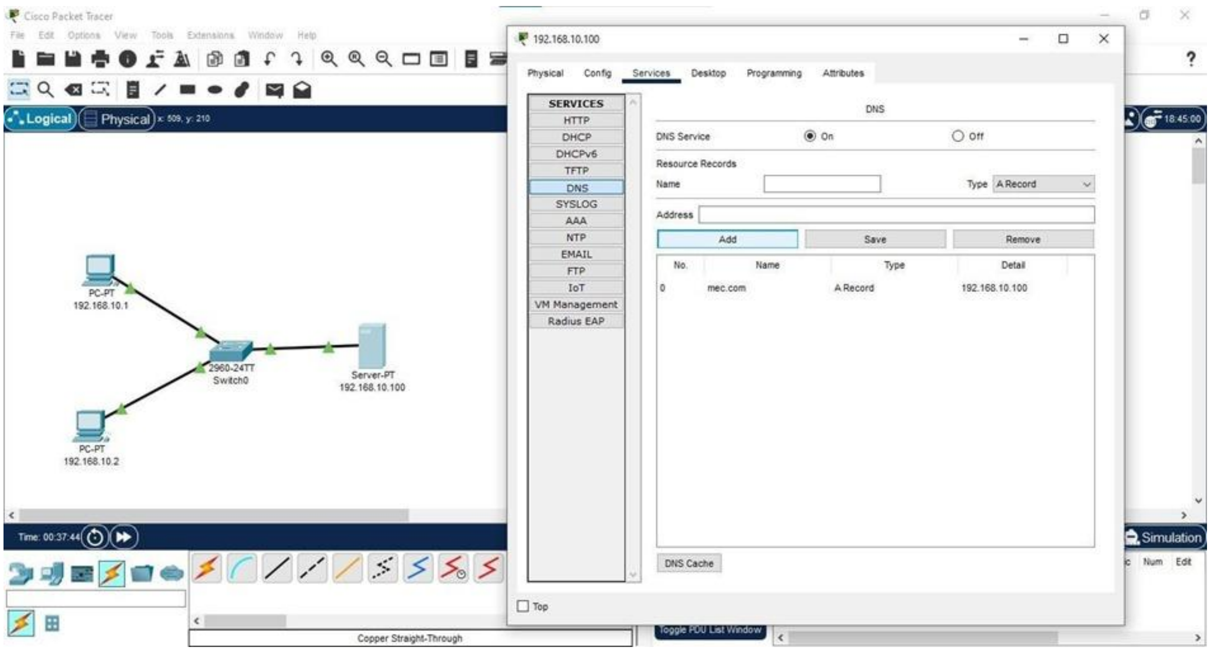
AIM:
Study of Cisco Packet Tracer and configure FTP server, DHCP server and DNS server in a wired network using required network devices.

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:



DNS

DNS Service (● On ○ Off)

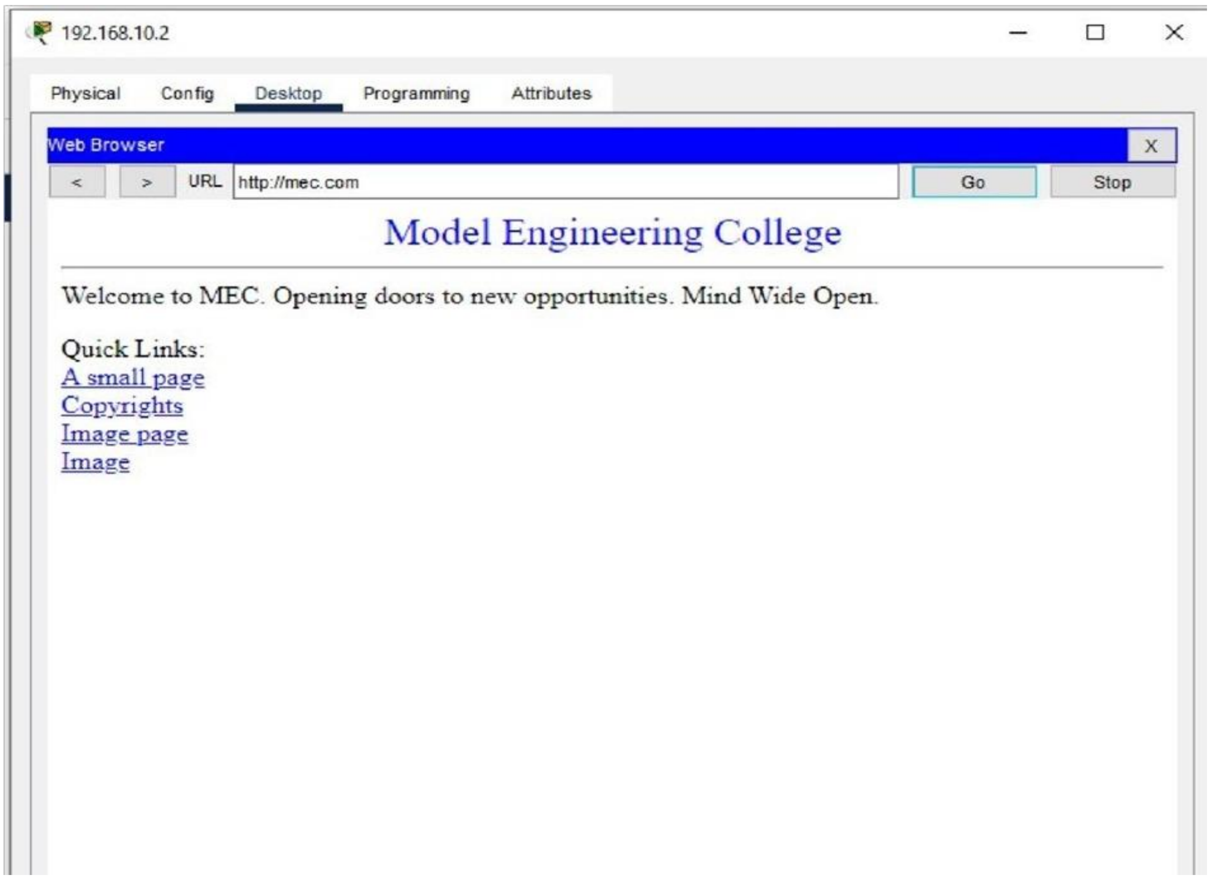
Resource Records

Name

Type A Record

Address

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



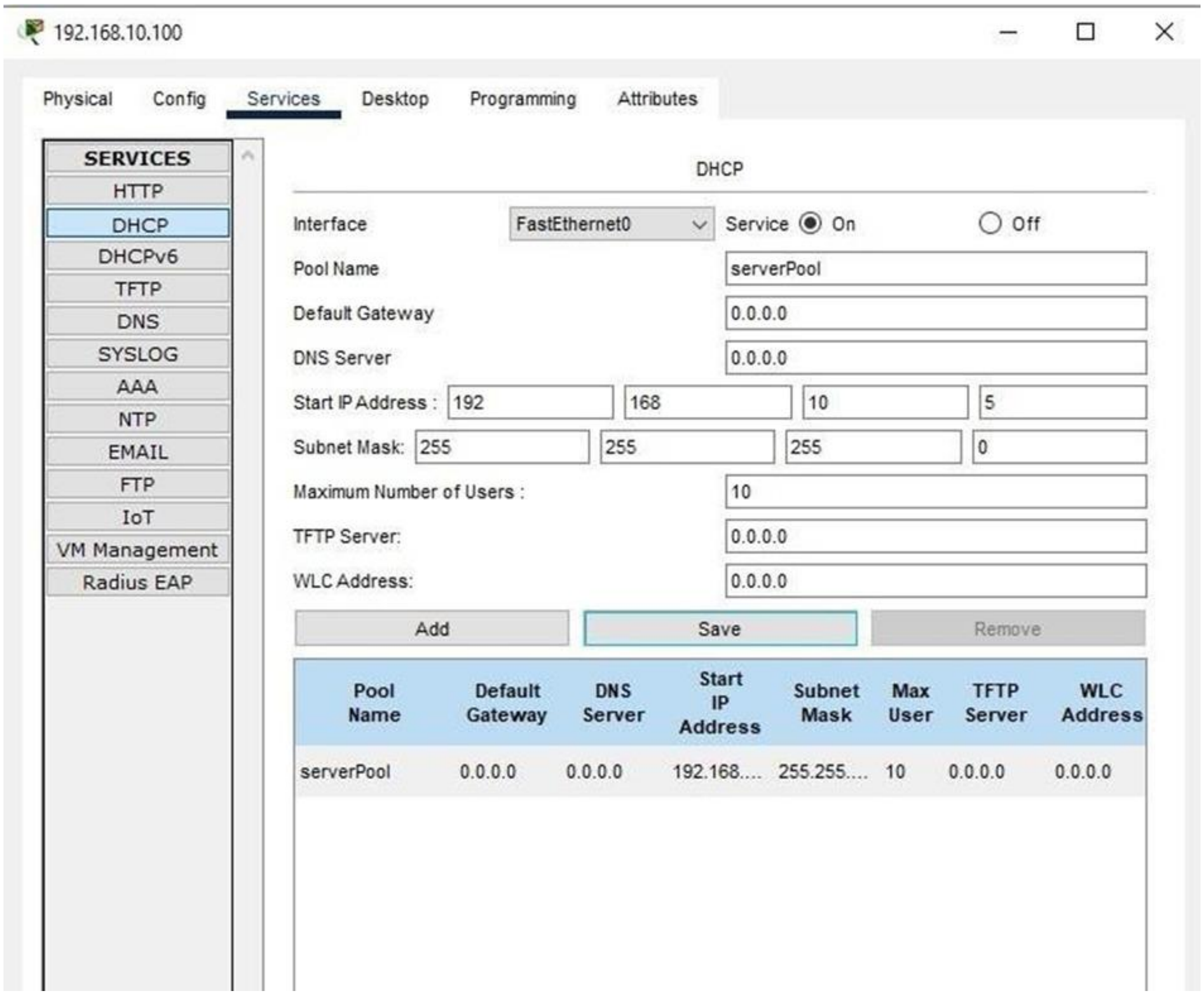
URL http://mec.com

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

Physical | Config | Services | Desktop | Programming | Attributes

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

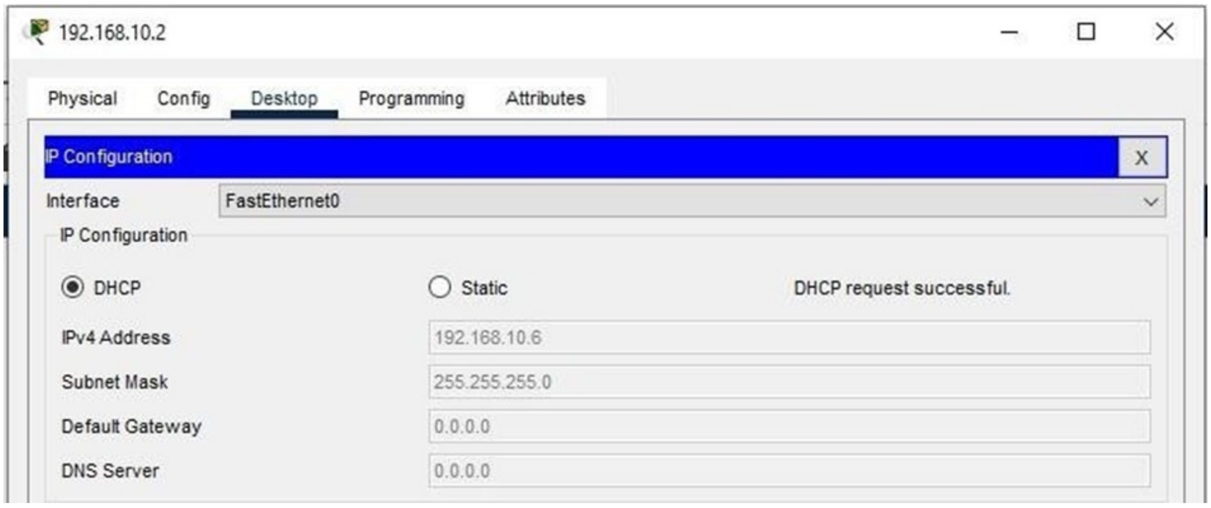
DHCP

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

Add | Save | Remove

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

Physical | Config | Desktop | Programming | Attributes

IP Configuration

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:

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