COLLEGE OF ENGINEERING

(ESTABLISHED BY IHRD GOVT. OF KERALA)

CSL 332 NETWORKING LAB

RECORD

NAME...

BRANCH...

SEMESTER.............................. ROLLNO..........................

Certified that this bonafide work is done by

Shri / Smt....

Year & Month :.......................... Date:....................

STAFF IN CHARGE: **HEAD OF DEPARTMENT:**

INTERNAL EXAMINER: **EXTERNAL EXAMINER:**

INDEX

SL NO.	PROGRAM NAME	DATE	PAGE NO.	REMARKS
1.	Basics of network configuration and networking commands			
2.	System calls for network programming in Linux			
3.	Client-server communication using socket programming and TCP			
4.	Client-server communication using socket programming and UDP			
5.	Simulate sliding window flow control protocols			
6.	Distance Vector Routing protocol			
7.	Simple Mail Transfer protocol			
8.	File Transfer protocol			
9.	Leaky bucket congestion control			
10.	Understanding the Wireshark tool			
11.	Design and configure a network			
12.	Study of NS2 simulator			

Basics of network configuration files and networking commands

Aim

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

Theory

The important network configuration files in Linux operating systems are:

1. /etc/hosts

This file is used to resolve hostnames on small networks with no DNS server. This text file contains a mapping of an IP address to the corresponding host name in each line. This file also contains a line specifying the IP address of the loopback device i.e, 127.0.0.1 is mapped to localhost.
A typical hosts file is as shown
127.0.0.1 localhost
127.0.1.1 anil-300E4Z-300E5Z-300E7Z

2. /etc/resolv.conf

This configuration file contains the IP addresses of DNS servers and the search domain.

3. /etc/sysconfig/network

This configuration file specifies routing and host information for all network interfaces. It contains directives that are global specific. For example if NETWORKING=yes, then /etc/init.d/network activates network devices.

4. /etc/nsswitch.conf

This file includes database search entries. The directive specifies which database is to be searched first.

The important Linux networking commands are:

1. ifconfig

This command gives the configuration of all interfaces in the system. It can be run with an interface name to get the details of the interface.

ifconfig wlan0

Link encap:Ethernet HWaddr b8:03:05:ad:6b:23

inet addr:192.168.43.15 Bcast:192.168.43.255 Mask:255.255.255.0

inet6 addr: 2405:204:d206:d3b1:ba03:5ff:fead:6b23/64 Scope:Global inet6 addr: fe80::ba03:5ff:fead:6b23/64 Scope:Link

inet6 addr: 2405:204:d206:d3b1:21ee:5665:de59:bd4e/64 Scope:Global UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:827087 errors:0 dropped:0 overruns:0 frame:0 TX packets:433391 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:1000

RX bytes:1117797710 (1.1 GB) TX bytes:53252386 (53.2 MB)

2. netstat

This command command gives network status information.

3. ping

This is the most commonly used command for checking connectivity.

A healthy connection is determined by a steady stream of replies with consistent times. Packet loss is shown by discontinuity of sequence numbers. Large scale packet loss indicates problem along the path.

Result

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

System calls for network programming in Linux

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

Theory
Basic Socket Functions
The following are some of the most basic function names:
socket (...)
bind (...)
connect (...)
select (...)
listen (...)
accept (...)
recv (...)
send (...)
shutdown (...)
close (...)

socket(Socket-Desc, Domain, Type, Protocol)
This function is used to create a new socket.
Socket-Desc is a unique identifier (like a file handle) for the socket created.
Domain is the address family, such as AF_INET (address family Internet), when ports are used for communication.
Type is the way in which data will be transmitted, such as in a continuous, consecutive, connection-oriented, reliable stream of bytes (SOCK_STREAM) in independent individual, connectionless, unreliable packets (SOCK_DGRAM) in the form of raw bytes (SOCK_RAW).
Protocol is the low-level protocol used for data transmission. This is TCP for stream sockets and UDP for datagram sockets.

bind(Socket-Desc, Packed-Structure-Address)
This function associates the server socket descriptor with information specified in the second parameter, especially the port on which the server will be listening for requests from clients.
Socket-Desc is the unique socket descriptor obtained from the socket function.

Packed-Structure-Address is a collection of information, such as the address family, the server port, any client address indicator stored in a particular format that can be understood by the bind function.

connect(Socket-Desc, Packed-Structure-Address)
The connect function is used by a client to establish connection to a server.
Socket-Desc is the client socket descriptor.
Packed-Structure-Address is a collection of information about the server to connect to, such as the address family, server port, and server IP address, stored in a particular format that can be understood by the connect function.

select(Socket-Desc/s)
The select function is used to check the status of a socket descriptor to determine, for example, whether it is ready for reading or writing, or whether it has an error condition pending.

listen(Socket-Desc, QueueSize)
This function is used by a connection-oriented server to specify the number of client requests that are allowed at a time and are queued for service.
Socket-Desc is the server socket descriptor on which the server is listening for requests from clients.
QueueSize specifies the number of client requests that can be queued until they can be serviced by the server on that socket. If there are more client requests than the number specified, then the client making the request might receive "connection refused" errors.

accept (Client-Socket-Desc, Server-Socket-Desc)
The accept function is used by a connection-oriented server to wait for and accept a client connection request.
Client-Socket-Desc is the new socket descriptor that is created by the function to represent the client whose request the server has accepted and will be processing.
Server-Socket-Desc is the server socket descriptor on which the server listens and accepts client requests for processing.

recv (Socket-Desc, Data-Buffer, Data-Buffer-length, Flags)
This function is used to receive data.

Socket-Desc is the socket descriptor that receives the data.
Data-Buffer specifies where the data received is stored.
Data-Buffer-Length is the size of the buffer, in bytes.
Flags are optional; they affect the way data is received.

send (Socket-Desc, Data-Buffer, Flags)

This function is used to send data.
Socket-Desc is the socket descriptor used to send data.
Data-Buffer specifies where the data to be sent is stored.
Flags are optional; they affect the way data is sent.

shutdown (Socket-Desc, How)

This function is used before the close function in case of a connection-oriented socket because there might be information ready to be sent or received.
Socket-Desc specifies the socket descriptor to be closed.
If the How value is 0, receiving bytes is disabled in this socket. If it is 1, sending bytes is disabled on this socket. If it is 2, both sending and receiving bytes is disabled on this socket.

close (Socket-Desc)

This function is used to close the socket and free the Socket-Desc.
Socket-Desc is the socket descriptor representing the socket to be closed.
This function does not seem to work in Perl, but it is a socket system call in C under UNIX.

Result

Familiarized and understood the use and functioning of system calls used for network programming in Linux.

Client-server communication using socket programming and TCP

Aim

Client sends a string to the server using tcp protocol. The server receives the string and replies with another string.

Algorithm

Client

1. Create socket
2. Connect the socket to the server
3. Read the string to be reversed from the standard input and send it to the server

Read the
matrices from the standard input and send it to server using socket

4. Read the reversed string from the socket and display it on the standard output

Read product
matrix from the socket and display it on the standard output

5. Close the socket

Server

1. Create listening socket
2. bind IP address and port number to the socket
3. listen for incoming requests on the listening socket
4. accept the incoming request
5. connection socket is created when accept returns
6. Read the string using the connection socket from the client
7. Reverse the string
8. Send the string to the client using the connection socket
9. close the connection socket
10. close the listening socket

Result: Program executed successfully and result obtained

PROGRAM NO: DATE: PAGE NO :

Code:

Client:

```
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define CHECK(expr)                                   \
        if ((expr) < 0)                               \
        return -1;                               \
        else                                          \
        printf("%s;\n", #expr);

int main(void) {
        int sockfd;
        struct sockaddr_in saddr;
        char buf[2000] = {'\0'};
        CHECK(sockfd = socket(AF_INET, SOCK_STREAM, 0))
        saddr.sin_family = AF_INET;
        saddr.sin_port = htons(3333);
        saddr.sin_addr.s_addr = INADDR_ANY;
        CHECK(connect(sockfd, (struct sockaddr *)&saddr, sizeof(saddr)))
        for (;;) {
        bzero(buf, sizeof(buf));
        scanf("%s", buf);
        if (!strcmp(buf, "x"))
        break;
        CHECK(send(sockfd, buf, sizeof(buf), 0))
        CHECK(recv(sockfd, buf, sizeof(buf), 0))
        printf("SERVER: %s\n", buf);
        } close(sockfd); return 0; }
```

PROGRAM NO: DATE: PAGE NO :

Server:

```
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define CHECK(expr)                              \
        if ((expr) < 0)                          \
        return -1;                          \
        else                                     \
        printf("%s;\n", #expr);

int main(void) {
        int sockfd;
        struct sockaddr_in saddr;
        char buf[2000] = {'\0'};
        CHECK(sockfd = socket(AF_INET, SOCK_STREAM, 0))
        saddr.sin_family = AF_INET;
        saddr.sin_port = htons(3333);
        saddr.sin_addr.s_addr = INADDR_ANY;
        CHECK(connect(sockfd, (struct sockaddr *)&saddr, sizeof(saddr)))
        for (;;) {
        bzero(buf, sizeof(buf));
        scanf("%s", buf);
        if (!strcmp(buf, "x"))
        break;
        CHECK(send(sockfd, buf, sizeof(buf), 0))
        CHECK(recv(sockfd, buf, sizeof(buf), 0))
        printf("SERVER: %s\n", buf);
        }
        close(sockfd); return 0; }
```

Output:

Terminal 1 (Server)

```
ubuntu@ubuntu:~/Desktop$ gcc tcp-server.c
ubuntu@ubuntu:~/Desktop$ ./a.out

Client : Hi
Server : Hello

Client : Nothing
Server : Fine
ubuntu@ubuntu:~/Desktop$
```

Terminal 2 (Client)

```
ubuntu@ubuntu:~/Desktop$ gcc tcp-client.c
ubuntu@ubuntu:~/Desktop$ ./a.out

Client : Hi

Server : Hello
```

Client-server communication using socket programming and UDP

Aim
Client sends a string to the server using UDP protocol. The server receives the string and replies with another string.

Algorithm:

Client
1.Create a UDP socket.
2.Bind the socket with the proper IP (Internet Protocol) address and the port number
3.Wait for the datagram packet from the client.
4.Process the datagram and send the reply.
5.Finish.

Server
1.Create a UDP socket.
2.Send a message to the server.
3.Wait for the reply from the server.
4.Process the packet.
5.Finish.

Result:
Program executed successfully and output obtained

PROGRAM NO: DATE: PAGE NO :

Client Code:

```
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define CHECK(expr)                              \
        if ((expr) < 0)                          \
        return -1;                               \
        else                                     \
        printf("%s;\n", #expr);

int main(void) {
        int sockfd;
        struct sockaddr_in saddr;
        socklen_t saddrsz = sizeof(saddr);
        char buf[2000] = {'\0'};
        CHECK(sockfd = socket(AF_INET, SOCK_DGRAM, 0))
        saddr.sin_family = AF_INET;
        saddr.sin_port = htons(3333);
        saddr.sin_addr.s_addr = INADDR_ANY;
        for (;;) {
        bzero(buf, sizeof(buf));
        scanf("%s", buf);
        if (!strcmp(buf, "x"))
        break;
        CHECK(sendto(sockfd, buf, sizeof(buf), MSG_CONFIRM,
                (struct sockaddr *)&saddr, sizeof(saddr)));
        bzero(buf, sizeof(buf));
        CHECK(recvfrom(sockfd, (char *)buf, sizeof(buf), MSG_WAITALL,
                (struct sockaddr *)&saddr, &saddrsz));
        printf("SERVER: %s\n", buf);
```

PROGRAM NO: DATE: PAGE NO :

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

Server Code:

```
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define CHECK(expr)                                         \
        if ((expr) < 0)                                 \
        return -1;                                  \
        else                                        \
        printf("%s;\n", #expr);

int main(void) {
        int sockfd;
        struct sockaddr_in saddr, caddr;
        socklen_t caddrsz = sizeof(caddr);
        char buf[2000] = {'\0'};
        CHECK(sockfd = socket(AF_INET, SOCK_DGRAM, 0))
        setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &(int){1},
sizeof(int));
        saddr.sin_family = AF_INET;
        saddr.sin_port = htons(3333);
        saddr.sin_addr.s_addr = INADDR_ANY;
        CHECK(bind(sockfd, (struct sockaddr *)&saddr, sizeof(saddr)))
        for (;;) {
        bzero(buf, sizeof(buf));
        CHECK(recvfrom(sockfd, (char *)buf, sizeof(buf), MSG_WAITALL,
```

PROGRAM NO: DATE: PAGE NO :

```
            (struct sockaddr *)&caddr, &caddrsz));
        printf("CLIENT: %s\n", buf);
        bzero(buf, sizeof(buf));
        scanf("%s", buf);
        if (!strcmp(buf, "x"))
        break;
        CHECK(sendto(sockfd, buf, sizeof(buf), MSG_CONFIRM,
            (struct sockaddr *)&caddr, sizeof(caddr)));
        }
        close(sockfd);
        return 0;
}
```

Output:

Terminal 1 (Server)

```
ubuntu@ubuntu:~/Desktop$ gcc udp-server.c
ubuntu@ubuntu:~/Desktop$ ./a.out
Client : Hello
Server : Hi
Client : Nothing
Server : Fine
ubuntu@ubuntu:~/Desktop$
```

Terminal 2 (Client)

```
ubuntu@ubuntu:~/Desktop$ gcc udp-client.c
ubuntu@ubuntu:~/Desktop$ ./a.out
Client : Hello
Server : Hi
Client : Nothing
Server : Fine
Client : end
ubuntu@ubuntu:~/Desktop$
```

Simulate sliding window flow control protocol

Aim: To write programs to simulate sliding window flow control protocol

Algorithm (Stop-and Wait)

Server:
1. Start
2. Establish UDP connection with client
3. Receive total number of frames from client
4. Repeat
a. Receive frame P from client
b. If P = -99, go to Step 5
c. Send acknowledgement to client (1 for positive, -1 for negative)
5. Stop

Client:
1. Start
2. Establish UDP connection with server
3. Send total number of frames N to server
4. For i = 1 to N, do
a. Let ACK = -1
b. Repeat
i. Send frame to server
ii. Receive acknowledgement from server
iii. If ACK == -1, print “Resending”
iv. Else, go to Step 5
5. Stop.

Result: Successfully implemented sliding window flow control protocols.

PROGRAM NO: DATE: PAGE NO :

Code:

Server:

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
int main() {
    printf("\nWaiting for client...\n");
    struct sockaddr_in servaddr, cliaddr;
    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(8080);
    bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr));
    int len = sizeof(cliaddr);
    int frames[100], n;
    recvfrom(sockfd, &n, sizeof(n), 0, (struct sockaddr *)&cliaddr, &len);
    printf("\nClient connected successfuly.\n");
    printf("\nWaiting for total number of frames...\n");
    recvfrom(sockfd, &n, sizeof(n), 0, (struct sockaddr *)&cliaddr, &len);
    int p, ack;
    while (1) {
        recvfrom(sockfd, &p, sizeof(n), 0, (struct sockaddr *)&cliaddr, &len);
        if (p == -99)
            return 0;
        printf("\nReceived frame-%d ", p);
        printf("\nEnter 1 for +ve ack and -1 for -ve ack.\n");
        scanf("%d", &ack);
        sendto(sockfd, &ack, sizeof(n), 0, (struct sockaddr *)&cliaddr, sizeof(cliaddr));
    }
    return 0;
}
```

PROGRAM NO: DATE: PAGE NO :

Client:

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
int main() {
    printf("\nSearching for server...\n");
    struct sockaddr_in servaddr;
    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(8080);
    servaddr.sin_addr.s_addr = INADDR_ANY;
    int n = -1;
    sendto(sockfd, &n, sizeof(n), 0, (struct sockaddr *)&servaddr, sizeof(servaddr));
    printf("\nServer connected successfully.\n");
    printf("\nEnter the total number of frames: ");
    scanf("%d", &n);
    sendto(sockfd, &n, sizeof(n), 0, (struct sockaddr *)&servaddr, sizeof(servaddr));
    int len, ack;
    for (int i = 1; i <= n; i++) {
        ack = -1;
        do {
            printf("\nSending frames-%d ", i);
            sendto(sockfd, &i, sizeof(n), 0, (struct sockaddr *)&servaddr,
                   sizeof(servaddr));
            printf("\nWaiting for ACK...\n");
            recvfrom(sockfd, &ack, sizeof(n), 0, (struct sockaddr *)&servaddr, &len);
            if (ack == -1) {
                printf("\nNegative ack recieved.. resending...\n");
            }
        } while (ack == -1);
    }
    n = -99;
    sendto(sockfd, &n, sizeof(n), 0, (const struct sockaddr *)&servaddr,
           sizeof(servaddr));
    printf("\nSuccessfully sent all the frames.\n");
    close(sockfd);
    return 0;
}
```

Output:

Terminal 1 (Server)

```
Waiting for client...

Client connected successfuly.

Waiting for total number of frames...

Received frame-1
Enter 1 for +ve ack and -1 for -ve ack.
1

Received frame-2
Enter 1 for +ve ack and -1 for -ve ack.
1

Received frame-3
Enter 1 for +ve ack and -1 for -ve ack.
-1

Received frame-3
Enter 1 for +ve ack and -1 for -ve ack.
1
```

```
Sending frames-1
Waiting for ACK...

Sending frames-2
Waiting for ACK...

Sending frames-3
Waiting for ACK...

Negative ack recieved.. resending...

Sending frames-3
Waiting for ACK...

Successfully sent all the frames.
```

PROGRAM NO: DATE: PAGE NO :

Terminal 2 (Client)

```
Searching for server...

Server connected successfully.

Enter the total number of frames: 3
```

```
Sending frames-1
Waiting for ACK...

Sending frames-2
Waiting for ACK...

Sending frames-3
Waiting for ACK...

Negative ack recieved.. resending...

Sending frames-3
Waiting for ACK...

Successfully sent all the frames.
```

PROGRAM NO: DATE: PAGE NO :

Distance Vector Routing protocol

Aim: To implement and simulate algorithm for Distance vector routing protocol.

Algorithm:

1. Start

2. Read the number of nodes from user into N

3. Construct a N x N routing table matrix

4. Read distance values into matrix from user

5. For x = 0 to N-1, do

a. For i = 0 to N-1, do

i. For j = 0 to N-1, do

1. For k = 0 to N-1, do

a. If table[i][j] > table[i][k] + table[k][j], then

i. table[i][j] = table[i][k] + table[k][j]

b. End If

2. End loop

ii. End loop

b. End loop

6. End loop

7. Print the new matrix

8. Stop

Result: Program executed successfully and intended output obtained.

PROGRAM NO: DATE: PAGE NO :

Program:

```
#include <stdio.h>
void main() {
    int number_of_nodes, i, j, k, x;
    printf("Enter the number of nodes: ");
    scanf("%d", &number_of_nodes);
    int routing_table[number_of_nodes][number_of_nodes];
    printf("Enter the routing table:\n");
    for (int i = 0; i < number_of_nodes; i++)
        for (int j = 0; j < number_of_nodes; j++) {
            printf("[%d][%d]: ", i, j);
            scanf("%d", &routing_table[i][j]);
        }
    for (int x = 0; x < number_of_nodes; x++)
        for (int i = 0; i < number_of_nodes; i++)
            for (int j = 0; j < number_of_nodes; j++)
                for (int k = 0; k < number_of_nodes; k++)
                    if (routing_table[i][j] > routing_table[i][k] + routing_table[k][j])
                        routing_table[i][j] = routing_table[i][k] + routing_table[k][j];
    printf("\nDistance Vector Table:\n");
    for (int i = 0; i < number_of_nodes; i++) {
        for (int j = 0; j < number_of_nodes; j++)
            printf("%d\t", routing_table[i][j]);
        printf("\n");
    }
}
```

PROGRAM NO: DATE: PAGE NO :

Output:

```
Enter the number of nodes: 4
Enter the routing table:
[0][0]: 0
[0][1]: 3
[0][2]: 2
[0][3]: 5
[1][0]: 3
[1][1]: 0
[1][2]: 8
[1][3]: 4
[2][0]: 2
[2][1]: 8
[2][2]: 0
[2][3]: 1
[3][0]: 5
[3][1]: 4
[3][2]: 1
[3][3]: 0

Distance Vector Table:
0       3       2       3
3       0       5       4
2       5       0       1
3       4       1       0
```

Simple Mail Transfer Protocol

AIM

To implement a subset of simple mail transfer protocol (SMTP) using UDP.

ALGORITHM

SMTP Client

1. Create the client UDP socket.
2. Send the message “SMTP REQUEST FROM CLIENT” to the server. This is done so that the server understands the address of the client.
3. Read the first message from the server using client socket and print it.
4. The first command HELO<”Client's mail server address”> is sent by the client.
5. Read the second message from the server and print it.
6. The second command MAIL FROM:<”email address of the sender”> is sent by the client.
7. Read the third message from the server and print it.
8. The third command RCPT TO:<”email address of the receiver”> is sent by the client.
9. Read the fourth message from the server and print it.
10. The fourth command DATA is sent by the client.
11. Read the fifth message from the server and print it.
12. Write the messages to the server and end with “.”
13. Read the sixth message from the server and print it.
14. The fifth command QUIT is sent by the client.
15. Read the seventh message from the server and print it.

SMTP Server

1. Create the server UDP socket.
2. Read the message from the client and gets the client's address.
3. Send the first command to the client. 220 “server name”.
4. Read the first message from the client and print it.

5. Send the second command to the client. 250 Hello “client name”.
6. Read the second message from client and print it.
7. Send the third command to the client. 250 “client email address “.........................Sender ok.
8. Read the third message from client and print it.
9. Send the fourth command to the client 250 “server email address”........................Recepient ok.
10. Read the fourth message from client and print it.
11. Send the fifth command to the client 354 Enter mail, end with “.” on a line by itself.
12. Read the email text from the client till a “.” is reached.
13. Send the sixth command to the client 250 Message accepted for delivery.
14. Read the fifth message from the client and print it.
15. Send the seventh command to the client

Result: Program executed successfully and result obtained

PROGRAM NO: DATE: PAGE NO :

Code:

Client

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <stdlib.h>
#include <ctype.h>

#define MAXLINE 100
int main(int argc, char *argv[])
{
        int n;
        int sock_fd;
        int i = 0;
        struct sockaddr_in servaddr;
        char buf[MAXLINE + 1];

        char address_buf[MAXLINE], message_buf[MAXLINE];
        char *str_ptr, *buf_ptr, *str;
        if (argc != 3)
        {
        fprintf(stderr, "Command is :./client address port\n");
        exit(1);
        }
        if ((sock_fd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
        {
        printf("Cannot create socket\n");
```

PROGRAM NO: DATE: PAGE NO :

```
        exit(1);
        }
        bzero((char *)&servaddr, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_port = htons(atoi(argv[2]));
        inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
        sprintf(buf, "SMTP REQUEST FROM CLIENT\n");
        n = sendto(sock_fd, buf, strlen(buf), 0, (struct sockaddr *)&servaddr,
sizeof(servaddr));
        if (n < 0)
        {
        perror("ERROR");
        exit(1);
        }
        if ((n = recvfrom(sock_fd, buf, MAXLINE, 0, NULL, NULL)) == -1)
        {
        perror("UDP read error");
        exit(1);
        }
        buf[n] = '\0';
        printf("S:%s", buf);
        sprintf(buf, "HELLO name_of_client_mail_server\n");
        n = sendto(sock_fd, buf, strlen(buf), 0, (struct sockaddr *)&servaddr,
sizeof(servaddr));
        if ((n = recvfrom(sock_fd, buf, MAXLINE, 0, NULL, NULL)) == -1)
        {
        perror("UDP read error");
        exit(1);
        }
        buf[n] = '\0';
        printf("S:%s", buf);
        printf("please enter the email address of the sender:");
```

PROGRAM NO: DATE: PAGE NO :

```
	fgets(address_buf, sizeof(address_buf), stdin);
	address_buf[strlen(address_buf) - 1] = '\0';

	sprintf(buf, "MAIL FROM :<%s>\n", address_buf);

	sendto(sock_fd, buf, sizeof(buf), 0, (struct sockaddr *)&servaddr,
sizeof(servaddr));
	if ((n = recvfrom(sock_fd, buf, MAXLINE, 0, NULL, NULL)) == -1)
	{
	perror("UDP read error");
	exit(1);
	}
	buf[n] = '\0';
	printf("S:%s", buf);
	printf("please enter the email address of the receiver:");
	fgets(address_buf, sizeof(address_buf), stdin);
	address_buf[strlen(address_buf) - 1] = '\0';
	sprintf(buf, "RCPT TO : <%s>\n", address_buf);
	sendto(sock_fd, buf, strlen(buf), 0, (struct sockaddr *)&servaddr,
sizeof(servaddr));
	if ((n = recvfrom(sock_fd, buf, MAXLINE, 0, NULL, NULL)) == -1)
	{
	perror("UDP read error");
	exit(1);
	}
	buf[n] = '\0';
	printf("S:%s", buf);
	sprintf(buf, "DATA\n");
	sendto(sock_fd, buf, strlen(buf), 0, (struct sockaddr *)&servaddr,
sizeof(servaddr));
	if ((n = recvfrom(sock_fd, buf, MAXLINE, 0, NULL, NULL)) == -1)
	{
```

PROGRAM NO: DATE: PAGE NO :

```
        perror("UDP read error");
        exit(1);
        }
        buf[n] = '\0';
        printf("S:%s", buf);
        do
        {
        fgets(message_buf, sizeof(message_buf), stdin);
        sprintf(buf, "%s", message_buf);
        sendto(sock_fd, buf, strlen(buf), 0, (struct sockaddr *)&servaddr,
sizeof(servaddr));
        message_buf[strlen(message_buf) - 1] = '\0';
        str = message_buf;
        while (isspace(*str++))
        ;
        if (strcmp(--str, ".") == 0)
        break;
        } while (1);

        if ((n = recvfrom(sock_fd, buf, MAXLINE, 0, NULL, NULL)) == -1)
        {
        perror("UDP read error");
        exit(1);
        }

        buf[n] = '\0';
        sprintf(buf, "QUIT\n");
        printf("S:%s", buf);
        sendto(sock_fd, buf, strlen(buf), 0, (struct sockaddr *)&servaddr,
sizeof(servaddr));
        if ((n = recvfrom(sock_fd, buf, MAXLINE, 0, NULL, NULL)) == -1)
        {
        perror("UDP read error");
        exit(1);
        }
```

PROGRAM NO: DATE: PAGE NO :

```
        buf[n] = '\0';
        printf("S:%s", buf);
}
```

Server

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <stdlib.h>
#include <ctype.h>

#define MAXLINE 100
int main(int argc, char *argv[])
{
        int n, sock_fd;
        struct sockaddr_in servaddr, cliaddr;
        char mesg[MAXLINE + 1];
        socklen_t len;
        char *str_ptr, *buf_ptr, *str;
        len = sizeof(cliaddr);
        if ((sock_fd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
        {
        printf("cannot create socket\n");
        exit(1);
        }
        bzero((char *)&servaddr, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_port = htons(atoi(argv[1]));
```

PROGRAM NO: DATE: PAGE NO :

```
        servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
        if (bind(sock_fd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0)
        {
        perror("bind failed:");
        exit(1);
        }

        if ((n = recvfrom(sock_fd, mesg, MAXLINE, 0, (struct sockaddr
*)&cliaddr, &len)) == -1)
        {
        perror("size not received:");
        exit(1);
        }
        mesg[n] = '\0';
        printf("mesg:%s\n", mesg);
        sprintf(mesg, "220 name_of_server_mail_server\n");
        sendto(sock_fd, mesg, MAXLINE, 0, (struct sockaddr *)&cliaddr,
sizeof(cliaddr));
        n = recvfrom(sock_fd, mesg, MAXLINE, 0, (struct sockaddr
*)&cliaddr, &len);
        mesg[n] = '\0';
        printf("C:%s\n", mesg);
        str_ptr = strdup(mesg);
        buf_ptr = strsep(&str_ptr, " ");
        sprintf(mesg, "250 Hello %s", str_ptr);
        free(buf_ptr);
        sendto(sock_fd, mesg, MAXLINE, 0, (struct sockaddr *)&cliaddr,
sizeof(cliaddr));
        n = recvfrom(sock_fd, mesg, MAXLINE, 0, (struct sockaddr
*)&cliaddr, &len);
        mesg[n] = '\0';
        printf("C:%s", mesg);
```

```
        str_ptr = strdup(mesg);
        buf_ptr = strsep(&str_ptr, ":");
        str_ptr[strlen(str_ptr) - 1] = '\0';

        sprintf(mesg, "250 Hello %s.......sender ok\n", str_ptr);
        free(buf_ptr);
        sendto(sock_fd, mesg, MAXLINE, 0, (struct sockaddr *)&cliaddr,
sizeof(cliaddr));
        n = recvfrom(sock_fd, mesg, MAXLINE, 0, (struct sockaddr
*)&cliaddr, &len);
        mesg[n] = '\0';
        printf("C:%s", mesg);
        str_ptr = strdup(mesg);
        buf_ptr = strsep(&str_ptr, ":");
        str_ptr[strlen(str_ptr) - 1] = '\0';
        sprintf(mesg, "250 Hello %s.......Recepient ok\n", str_ptr);
        free(buf_ptr);
        sendto(sock_fd, mesg, MAXLINE, 0, (struct sockaddr *)&cliaddr,
sizeof(cliaddr));
        n = recvfrom(sock_fd, mesg, MAXLINE, 0, (struct sockaddr
*)&cliaddr, &len);
        mesg[n] = '\0';
        printf("C:%s\n", mesg);
        sprintf(mesg, "354 Enter mail,end with \".\" on a line by itself \n");
        sendto(sock_fd, mesg, MAXLINE, 0, (struct sockaddr *)&cliaddr,
sizeof(cliaddr));
        while (1)
        {
        n = recvfrom(sock_fd, mesg, MAXLINE, 0, (struct sockaddr
*)&cliaddr, &len);
        mesg[n] = '\0';
        printf("C:%s\n", mesg);
```

```
        mesg[strlen(mesg) - 1] = '\0';

        str = mesg;
        while (isspace(*str++))
        ;
        if (strcmp(--str, ".") == 0)
        break;
        sprintf(mesg, "250 messages accepted for delivery \n");
        sendto(sock_fd, mesg, MAXLINE, 0, (struct sockaddr *)&cliaddr,
sizeof(cliaddr));
        n = recvfrom(sock_fd, mesg, MAXLINE, 0, (struct sockaddr
*)&cliaddr, &len);
        mesg[n] = '\0';
        printf("C:%s\n", mesg);
        sprintf(mesg, "221 servers mail server closing connection\n");
        sendto(sock_fd, mesg, MAXLINE, 0, (struct sockaddr *)&cliaddr,
sizeof(cliaddr));
        }
}
```

Output

./client 127.0.0.1 6500
S:220 name_of_server_mail_server
S:250 Hello name_of_client_mail_server
please enter the email address of the sender:aromal@katha.today
S:250 Hello <aromal@katha.today>.......sender ok
please enter the email address of the receiver:ashishpeter@gmail.com
S:250 Hello <ashishpeter@gmail.com>.......Recepient ok
S:354 Enter mail,end with "." on a line by itself
Hello Ashish! How Are You ?
.
S:QUIT
S:221 servers mail server closing connection

PROGRAM NO: DATE: PAGE NO :

```
./server 6500
mesg:SMTP REQUEST FROM CLIENT
C:HELLO name_of_client_mail_server
C:MAIL FROM :<aromal@katha.today>
C:RCPT TO : <ashishpeter@gmail.com>
C:DATA
C:Hello Ashish! How Are You ?
C:.
C:QUIT
```

PROGRAM NO: DATE: PAGE NO :

File Transfer Protocol

Aim: Implement file transfer over UDP

Algorithm:

1. The server starts and waits for the filename.
2. The client sends a filename.
3. The server receives the filename.
   If file is present,
   server starts reading file
   and continues to send a buffer filled with
   file contents until file-end is reached.
4. End is marked by EOF.
5. File is received as buffers until EOF is received.
6. If Not present, a file not found is sent.

Result: Program executed successfully and result obtained.

PROGRAM NO: DATE: PAGE NO :

Program:

Client:

```
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define BUFSZ 32
#define FLAG 0
#define CHECK(expr)                     \
        if ((expr) < 0)                 \
        return -1;                      \
        else                            \
        printf("%s;\n", #expr);

int recvFile(char *buf, int s) {
        char ch;
        for (int i = 0; i < s; i++)
        if ((ch = buf[i]) == EOF)
        return 1;
        else
        printf("%c", ch);
        return 1;
}

int main() {
        int sockfd;
        struct sockaddr_in saddr;
        socklen_t addrlen = sizeof(saddr);
        saddr.sin_family = AF_INET;
```

PROGRAM NO: DATE: PAGE NO :

```
        saddr.sin_port = htons(3333);
        saddr.sin_addr.s_addr = INADDR_ANY;
        char buf[BUFSZ] = {'\0'};
        CHECK(sockfd = socket(AF_INET, SOCK_DGRAM, 0));
        while (1) {
        printf("\nPlease enter file name to receive: ");
        scanf("%s", buf);
        CHECK(sendto(sockfd, buf, BUFSZ, FLAG, (struct sockaddr
*)&saddr,
                addrlen));
        printf("\n---------Data Received---------\n");
        while (1) {
        bzero(buf, sizeof(buf));
        CHECK(recvfrom(sockfd, buf, BUFSZ, FLAG, (struct sockaddr
*)&saddr,
                        &addrlen));
        if (recvFile(buf, BUFSZ))
                break;
        }
        printf("\n-------------------------------\n");
        }
        return 0;
}
```

PROGRAM NO: DATE: PAGE NO :

Server:

```
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define BUFSZ 32
#define FLAG 0
#define CHECK(expr)                                  \
        if ((expr) < 0)                              \
        return -1;                                   \
        else                                         \
        printf("%s;\n", #expr);

int sendFile(FILE *fp, char *buf, int s) {
        int i = 0;
        if (fp == NULL) {
        strcpy(buf, "File Not Found!");
        return 1;
        }
        while (!feof(fp))
        buf[i++] = fgetc(fp);
        return 1;
}

int main() {
        int sockfd;
        struct sockaddr_in caddr;
        socklen_t addrlen = sizeof(caddr);
        caddr.sin_family = AF_INET;
        caddr.sin_port = htons(3333);
        caddr.sin_addr.s_addr = INADDR_ANY;
```

PROGRAM NO: DATE: PAGE NO :

```
        char buf[BUFSZ] = {'\0'};
        FILE *fp;
        CHECK(sockfd = socket(AF_INET, SOCK_DGRAM, 0));
        setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &(int){1},
sizeof(int));
        CHECK(bind(sockfd, (struct sockaddr *)&caddr, sizeof(caddr)));
        for (;;) {
        printf("Waiting for filename...\n");
        bzero(buf, sizeof(buf));
        CHECK(recvfrom(sockfd, buf, BUFSZ, FLAG, (struct sockaddr
*)&caddr,
                        &addrlen));
        printf("File Name Received: %s", buf);
        ((fp = fopen(buf, "r")) == NULL) ? printf(" (ERR)\n")
                                : printf(" (OPN)\n");
        for (;;) {
        if (sendFile(fp, buf, BUFSZ)) {
                CHECK(sendto(sockfd, buf, BUFSZ, FLAG,
                        (struct sockaddr *)&caddr, addrlen));
                break;
        }
        bzero(buf, sizeof(buf));
        }
        if (fp != NULL)
        fclose(fp);
        }
        return 0;
}
```

PROGRAM NO: DATE: PAGE NO :

File1: contents of file1

File2: contents of file2

Output:
./server
Waiting for filename
Filename received: file1

./client
Please enter file name to receive: file1
---------Data Received---------
Contents of file1
—---------------------------------

Leaky bucket congestion control

AIM: Implement congestion control using a leaky bucket algorithm

Algorithm:

1. Start
2. Let STORE = 0
3. Read bucket size (BUCKETSIZE), outgoing rate (OUTGOING) and number of
inputs (N) from user
4. While N ≠ 0, do
a. Read packet size to INCOMING
b. If INCOMING ≤ (BUCKETSIZE - STORE), then
i. STORE = STORE + INCOMING
ii. Print STORE
c. Else, then
a. Print INCOMING - (BUCKETSIZE - STORE) as “Dropped number of packets”
b. STORE = BUCKETSIZE
c. Print STORE
d. STORE = STORE - OUTGOING
e. If STORE < 0, then
a. STORE = 0
f. Print STORE
g. Decrement N by 1
5. Stop

Result:

Successfully implemented congestion control using Leaky Bucket algorithm in C.

PROGRAM NO: DATE: PAGE NO :

Code:

```
#include<stdio.h>
int main() {
int incoming, outgoing, buck_size, n, store = 0;
printf("Enter bucket size, outgoing rate and number of inputs: ");
scanf("%d %d %d", & buck_size, & outgoing, & n);
while (n != 0) {
printf("Enter incoming packet size: ");
scanf("%d", & incoming);
printf("Incoming packet size is %d\n", incoming);
if (incoming <= (buck_size - store)) {
store += incoming;
printf("Bucket buffer size is %d out of %d\n", store,
buck_size);
} else {
printf("Dropped %d no of packets\n",
incoming - (buck_size - store));
store = buck_size;
printf("Bucket buffer size is %d out of %d\n", store,
buck_size);
}
store = store - outgoing;
if (store < 0)
store = 0;
printf("After outgoing, %d packets left out of %d
in buffer\n", store, buck_size);
n--;
}
}
```

Output:

Enter bucket size, outgoing rate and number of inputs: 25 10 4
Enter incoming packet size: 20
Incoming packet size is 20
Bucket buffer size is 20 out of 25
After outgoing, 10 packets left out of 25 in buffer
Enter incoming packet size: 20
Incoming packet size is 20
Dropped 5 no of packets
Bucket buffer size is 25 out of 25
After outgoing, 15 packets left out of 25 in buffer
Enter incoming packet size: 5
Incoming packet size is 5
Bucket buffer size is 20 out of 25
After outgoing, 10 packets left out of 25 in buffer
Enter incoming packet size: 10
Incoming packet size is 10
Bucket buffer size is 20 out of 25
After outgoing, 10 packets left out of 25 in buffer

PROGRAM NO: DATE: PAGE NO :

Understanding the Wireshark tool

Aim:

To learn and use Wireshark to observe TCP and UDP connections live.

Description:

Wireshark is a free network protocol analyzer tool. It is also a packet sniffer since it
can be used to observer messages exchanged between protocol entities.

Wireshark interface has 4 major components:

a) Command menus

b) Packet listing window

c) Packet header details window

d) Packet contents window

Method:

a) Observing TCP connection

Example website used: Computer Networks Research Group website

IP address: 128.119.245.12

Steps:

Filter connections by typing "ip.addr == 128.119.245.12" into the filter toolbar

Open web browser and go to http://128.119.245.12

Observe the newly appeared entries on Wireshark

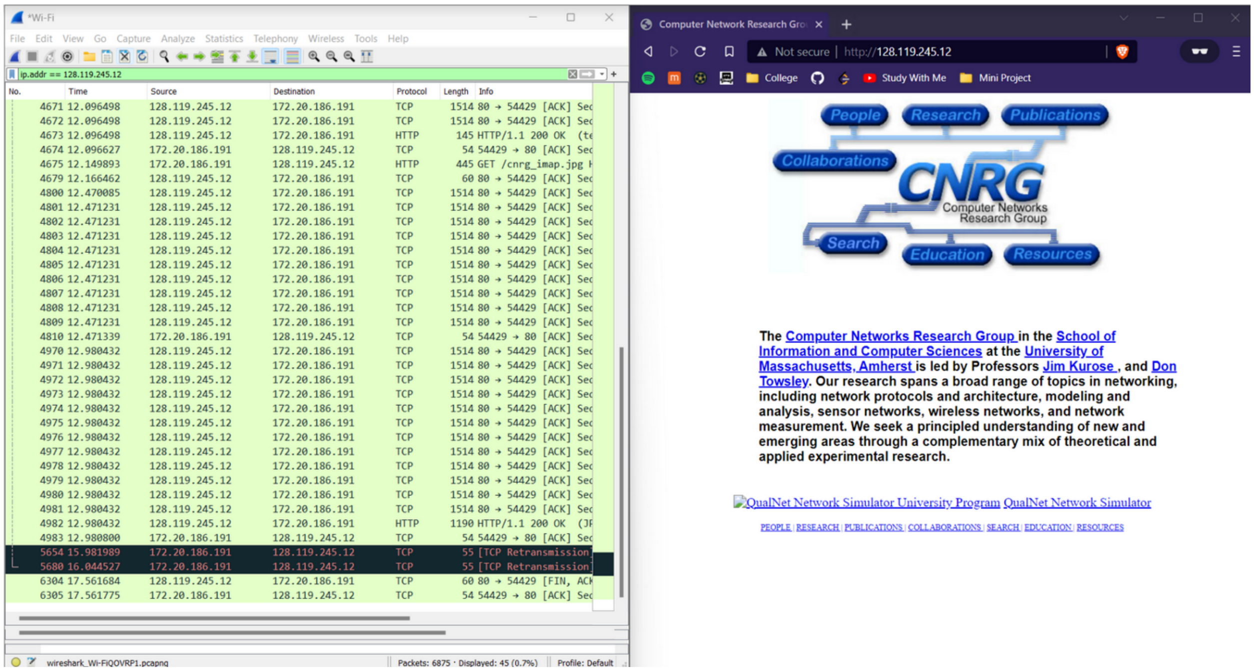
b) Observing UDP connection
Example website used: Google Meet
IP address: 142.250.82.0 to 142.250.82.255
Steps:
Filter connections by typing “ip.addr == 142.250.82.0/24”
Go to Google Meet and start a new meeting
Observe the newly appeared entries on Wireshark

Result:
Successfully familiarized with Wireshark tool. Used Wireshark to observe TCP and
UDP connections.

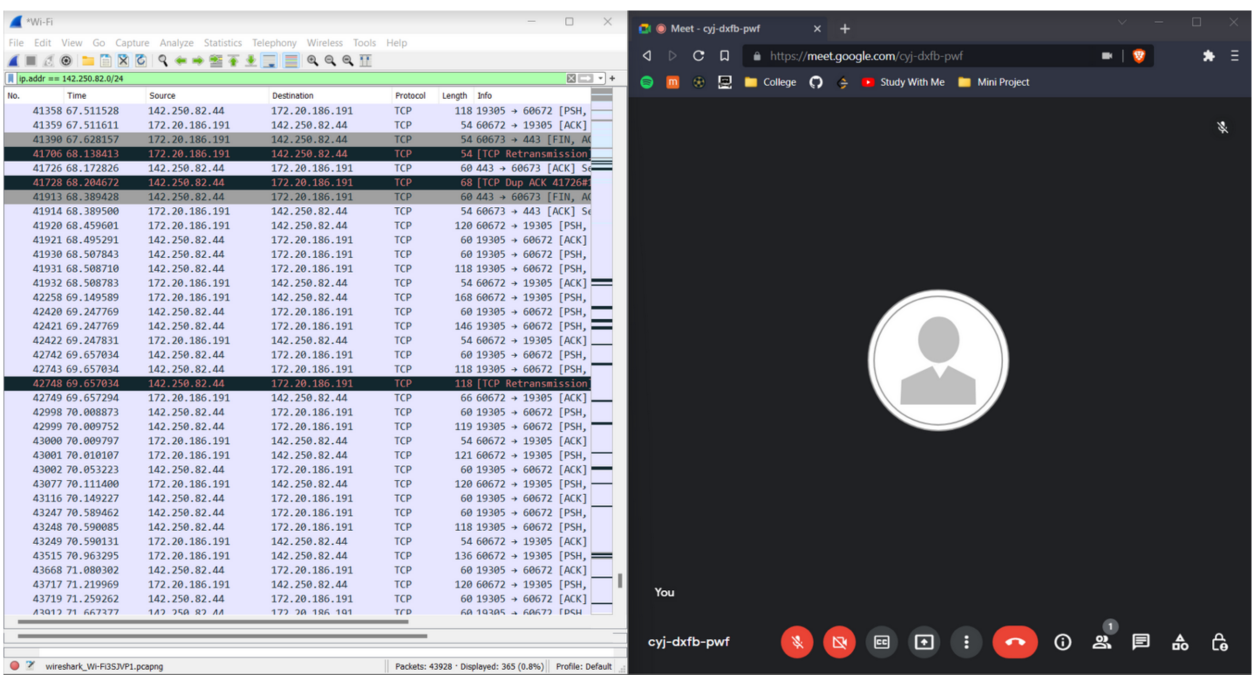
PROGRAM NO: DATE: PAGE NO :

Output:

a) Observing TCP connection



b) Observing UDP connection



PROGRAM NO: DATE: PAGE NO :

Design and configure a network

Aim:

Design and configure a network with multiple subnets with wired LANs using
required network devices. Configure commonly used services in the network.

Result:

Successfully designed and configured a network with multiple subnets using network
Devices.

Procedure & Output:

Router 1:

```
Router>enable
Router#config terminal
Router(config)#interface fa0/0
Router(config-if)#ip address 192.168.10.1 255.255.255.0
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#interface s0/0/0
Router(config-if)#ip address 222.222.22.1 255.255.255.0
Router(config-if)#clock rate 64000
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#exit
Router#copy running-config startup-config
Router>enable
Router#config terminal
Router(config)#router rip
Router(config-router)#network 192.168.10.0
Router(config-router)#network 222.222.22.0
Router(config-router)#exit
Router(config)#exit
Router#copy running-config startup-config
```

Router 2:

```
Router>enable
Router#config terminal
Router(config)#interface fa0/0
Router(config-if)#ip address 192.168.20.1 255.255.255.0
Router(config-if)#no shutdown
Router(config-if)#exit
```

PROGRAM NO: DATE: PAGE NO :

```
Router(config)#interface s0/0/0
Router(config-if)#ip address 222.222.22.2 255.255.255.0
Router(config-if)#clock rate 64000
Router(config-if)#no shutdown
Router(config-if)#exit
Router(config)#exit
Router#copy running-config startup-config
Router>enable
Router#config terminal
Router(config)#router rip
Router(config-router)#network 192.168.20.0
Router(config-router)#network 222.222.22.0
Router(config-router)#exit
Router(config)#exit
Router#copy running-config startup-config
```

Study of NS2 simulator

AIM: To study about NS2 simulator in detail.

THEORY:

Network Simulator (Version 2), widely known as NS2, is simply an event driven simulation tool that
has proved useful in studying the dynamic nature of communication networks. Simulation of wired as
well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done
using NS2. In general, NS2 provides users with a way of specifying such network protocols and
simulating their corresponding behaviors. Due to its flexibility and modular nature, NS2 has gained
constant popularity in the networking research community since its birth in 1989. Ever since, several
revolutions and revisions have marked the growing maturity of the tool, thanks to substantial
contributions from the players in the field. Among these are the University of California and Cornell
University who developed the REAL network simulator,1 the foundation which NS is based on. Since
1995 the Defense Advanced Research Projects Agency (DARPA) supported development of NS
through the Virtual Inter Network Testbed (VINT) project . Currently the National Science
Foundation (NSF) has joined the ride in development. Last but not the least, the group of Researchers
and developers in the community are constantly working to keep NS2 strong and versatile.

BASIC ARCHITECTURE:

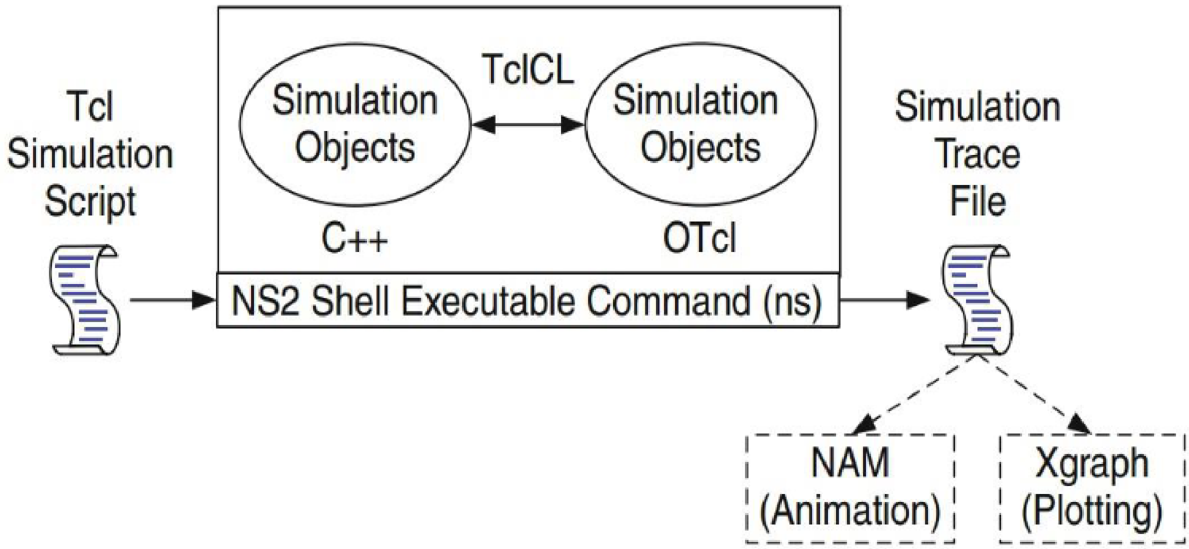


Fig. 2.1. Basic architecture of NS.

Figure 2.1 shows the basic architecture of NS2. NS2 provides users with an executable
command ns which takes on input argument, the name of a Tcl simulation scripting file. Users are
feeding the name of a Tcl simulation script (which sets up a simulation) as an input argument of an
NS2 executable command ns.
In most cases, a simulation trace file is created, and is used to plot graph and/or to create
animation. NS2 consists of two key languages: C++ and Object-oriented Tool Command Language
(OTcl). While the C++ defines the internal mechanism (i.e., a backend) of the simulation objects, the
OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete
events (i.e., a frontend).

PROGRAM NO: DATE: PAGE NO :

The C++ and the OTcl are linked together using TclCL. Mapped to a C++ object, variables in
the OTcl domains are sometimes referred to as handles. Conceptually, a handle (e.g., n as a Node
handle) is just a string (e.g.,_o10) in the OTcl domain, and does not contain any functionality. Instead,
the functionality (e.g., receiving a packet) is defined in the mapped C++ object (e.g., of class
Connector). In the OTcl domain, a handle acts as a frontend which interacts with users and other OTcl objects. It may defines its own procedures and variables to facilitate the interaction. Note that the
member procedures and variables in the OTcl domain are called instance procedures (instprocs) and
instance variables (instvars), respectively. Before proceeding further, the readers are encouraged to
learn C++ and OTcl languages. We refer the readers to [14] for the detail of C++, while a brief tutorial
of Tcl and OTcl tutorial are given in Appendices A.1 and A.2, respectively.
NS2 provides a large number of built-in C++ objects. It is advisable to use these C++ objects
to set up a simulation using a Tcl simulation script. However, advance users may find these objects
insufficient. They need to develop their own C++ objects, and use a OTcl configuration interface to
put together these objects. After simulation, NS2 outputs either text-based or animation-based
simulation results. To interpret these results graphically and interactively, tools such as NAM
(Network AniMator) and XGraph are used. To analyze a particular behaviour of the network, users
can extract a relevant subset of text-based data and transform it to a more conceivable presentation.

PROGRAM NO: DATE: PAGE NO :

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

Tcl scripting
Tcl is a general purpose scripting language. [Interpreter]
• Tcl runs on most of the platforms such as Unix, Windows, and Mac.
• The strength of Tcl is its simplicity.
• It is not necessary to declare a data type for variable prior to the usage.

Basics of TCL
Syntax: command arg1 arg2 arg3
Hello World!
puts stdout{Hello, World!} Hello, World!
Variables Command Substitution
set a 5 set len [string length foobar]
set b $a set len [expr [string length foobar] + 9]
Wired TCL Script Components
Create the event scheduler
Open new files & turn on the tracing
Create the nodes

PROGRAM NO: DATE: PAGE NO :

Setup the links
Configure the traffic type (e.g., TCP, UDP, etc)
Set the time of traffic generation (e.g., CBR, FTP)
Terminate the simulation

NS Simulator Preliminaries.
1. Initialization and termination aspects of the ns simulator.
2. Definition of network nodes, links, queues and topology.
3. Definition of agents and of applications.
4. The nam visualization tool.
5. Tracing and random variables.

Initialization and Termination of TCL Script in NS-2
An ns simulation starts with the command
set ns [new Simulator]
Which is thus the first line in the tcl script. This line declares a new variable as using the set command,
you can call this variable as you wish, In general people declares it as ns because it is an instance of
the Simulator class, so an object the code[new Simulator] is indeed the installation of the class
Simulator using the reserved word new.
In order to have output files with data on the simulation (trace files) or files used for visualization
(nam files), we need to create the files using —open command:
#Open the Trace file
set tracefile1 [open out.tr w]
$ns trace-all $tracefile1
#Open the NAM trace file
set namfile [open out.nam w]
$ns namtrace-all $namfile

PROGRAM NO: DATE: PAGE NO :

The above creates a dta trace file called out.tr and a nam visualization trace file called out.nam.
Within the tcl script, these files are not called explicitly by their names, but instead by pointers that are
declared above and called —tracefile1 and —namfile respectively. Remark that they begins with a #
symbol. The second line open the file —out.tr to be used for writing, declared with the letter —w. The
third line uses a simulator method called trace-all that have as parameter the name of the file where the
traces will go.
Define a "finish" procedure
Proc finish { } {
global ns tracefile1 namfile
$ns flush-trace
Close $tracefile1
Close $namfile
Exec nam out.nam &
Exit 0
}
Definition of a network of links and nodes
The way to define a node is
set n0 [$ns node]
Once we define several nodes, we can define the links that connect them. An example of a definition
of a link is:
$ns duplex-link $n0 $n2 10Mb 10ms DropTail
Which means that $n0 and $n2 are connected using a bi-directional link that has 10ms of propagation
delay and a capacity of 10Mb per sec for each direction.

Result: Network Simulator 2 is studied in detail.