INDEX

OUTPUT

```
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.20.218.122  netmask 255.255.240.0  broadcast 172.20.223.255
        inet6 fe80::215:5dff:fe35:bdcd  prefixlen 64  scopeid 0x20<link>
        ether 00:15:5d:35:bd:cd  txqueuelen 1000  (Ethernet)
        RX packets 223  bytes 311672 (311.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 95  bytes 7972 (7.9 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 16  bytes 1915 (1.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 16  bytes 1915 (1.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0


$ 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     21675    @70b70142ef729661/bus/systemd-
resolve/bus-api-resolve
unix  2      [ ]         DGRAM                    1384     /run/user/1000/systemd/notify
unix  2      [ ]         DGRAM                    18425    /var/run/chrony/chronyd.sock
unix  3      [ ]         DGRAM      CONNECTED     24590    /run/systemd/notify
unix  2      [ ]         DGRAM                    24604    /run/systemd/journal/syslog
unix  8      [ ]         DGRAM      CONNECTED     24606    /run/systemd/journal/dev-log
unix  8      [ ]         DGRAM      CONNECTED     24608    /run/systemd/journal/socket
unix  3      [ ]         STREAM     CONNECTED     1388     @e431526e7de090ac/bus/systemd/bus-
system


$ traceroute google.com
traceroute to google.com (142.250.196.78), 30 hops max, 60 byte packets
 1  172.20.208.1 (172.20.208.1)  0.295 ms  0.277 ms  0.266 ms
 2  192.168.1.1 (192.168.1.1)  13.161 ms  2.524 ms  2.513 ms
 3  * * *
 4  * * *
 5  130.230.88.202.asianet.co.in (202.88.230.130)  28.193 ms  2980.068 ms  2980.059 ms
 6  142.250.173.164 (142.250.173.164)  2983.093 ms  16.521 ms  17.219 ms
 7  * * *
 8  209.85.248.180 (209.85.248.180)  29.239 ms 142.251.55.232 (142.251.55.232)  32.804 ms
216.239.56.62 (216.239.56.62)  37.006 ms
 9  142.250.236.157 (142.250.236.157)  29.208 ms 172.253.70.166 (172.253.70.166)  42.180 ms
142.251.55.121 (142.251.55.121)  39.454 ms
10  142.250.239.229 (142.250.239.229)  40.885 ms maa03s46-in-f14.1e100.net (142.250.196.78)
42.103 ms  43.123 ms
```

Experiment No.1 **Date: 29-07-2024**

BASIC NETWORKING COMMANDS

AIM

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

NETWORK COMMANDS

- **IFCONFIG** - manages and displays network interface settings.
  ifconfig (interface configuration) is a command-line tool used in Unix-like operating systems, such as Linux and macOS, to configure and manage network interfaces. It provides detailed information about the system's network interfaces, including IP addresses, subnet masks, MAC addresses, and the status of each interface.
  Users can use ifconfig to enable or disable network interfaces, assign IP addresses, configure network parameters, and view the current network configuration of each interface.

- **NETSTAT** - shows active network connections and network interface statistics.
  netstat (network statistics) is a command-line tool used in Unix-like operating systems to display network connections, routing tables, interface statistics, and other network-related information.
  It provides details about active network connections, such as the source and destination IP addresses, port numbers, and the status of each connection (e.g., listening, established, or closed). netstat can also display network interface statistics, such as the number of packets transmitted and received.

- **TRACEROUTE** - shows the path data takes through network routers.
  traceroute is a command-line tool used to trace the path data packets take from one computer to another over a network. It shows each hop along the route, displaying the IP addresses of intermediate routers and the time it takes for data to travel between each.
  This tool helps identify network bottlenecks or failures by showing where delays or packet loss occur along the route.

- **PING** - measures the response time between two networked devices.
  ping is a command-line tool used to test the connectivity between two devices on a network. It sends small data packets (ICMP Echo Request) to a target IP address or domain and waits for a response (ICMP Echo Reply).
  The time it takes for the packet to travel to the target and back (round-trip time) is measured, helping to assess network performance and connectivity.

- **ROUTE** - displays and modifies the IP routing table on a device.
  The route command is used to view and manipulate the IP routing table in a computer's network configuration. It allows users to display, add, or delete routes that determine how data is forwarded between network interfaces. The command is often used for troubleshooting network connectivity and managing routing paths.

```
$ ping google.com
PING google.com (142.250.182.142) 56(84) bytes of data.
64 bytes from maa05s22-in-f14.1e100.net (142.250.182.142): icmp_seq=1 ttl=117 time=17.7 ms
64 bytes from maa05s22-in-f14.1e100.net (142.250.182.142): icmp_seq=2 ttl=117 time=17.1 ms
64 bytes from maa05s22-in-f14.1e100.net (142.250.182.142): icmp_seq=3 ttl=117 time=18.1 ms
64 bytes from maa05s22-in-f14.1e100.net (142.250.182.142): icmp_seq=4 ttl=117 time=18.2 ms
64 bytes from maa05s22-in-f14.1e100.net (142.250.182.142): icmp_seq=5 ttl=117 time=18.0 ms
64 bytes from maa05s22-in-f14.1e100.net (142.250.182.142): icmp_seq=6 ttl=117 time=18.4 ms
64 bytes from maa05s22-in-f14.1e100.net (142.250.182.142): icmp_seq=7 ttl=117 time=17.3 ms
64 bytes from maa05s22-in-f14.1e100.net (142.250.182.142): icmp_seq=8 ttl=117 time=18.0 ms
64 bytes from maa05s22-in-f14.1e100.net (142.250.182.142): icmp_seq=9 ttl=117 time=19.7 ms
64 bytes from maa05s22-in-f14.1e100.net (142.250.182.142): icmp_seq=10 ttl=117 time=17.5 ms

$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         172.20.208.1    0.0.0.0         UG    0      0        0 eth0
172.20.208.0    0.0.0.0         255.255.240.0   U     0      0        0 eth0
```

RESULT

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

PROGRAM

//Server (TCP)

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

int main(){
    char buf[100];
    int k;
    struct sockaddr_in server,client;
    socklen_t server_len,client_len;
    int sockfd,newsockfd;

    sockfd = socket(AF_INET,SOCK_STREAM,0);
    if(sockfd == -1)
        printf("Error in socketcreation\n");

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(8080);
    client_len = sizeof(client);

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

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

    newsockfd = accept(sockfd, (struct sockaddr *)&client,&client_len);
    if (newsockfd == -1)
        printf("Error in temporary socket creation\n");

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

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

Experiment No.2 **Date: 08-08-2024**

TRANSMISSION CONTROL PROTOCOL

AIM

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

ALGORITHM

Server:

1. Create a socket using `socket()` for TCP communication.
2. Initialize the `server` structure with `AF_INET` for IPv4, `INADDR_ANY` for any incoming address, and port 8080.
3. Bind the socket to the server address using `bind()`.
4. Listen for incoming client connections using `listen()`.
5. Accept a connection from the client with `accept()`.
6. Receive data from the client using `recv()` and store it in a buffer.
7. Print the received message to the console.
8. Close the client socket (`newsockfd`) after processing the message.
9. Close the server socket (`sockfd`) to release resources.
10. End the program, completing the client-server communication.

Client:

1. Create a socket using `socket()` for TCP communication.
2. Set up the `server` structure with `AF_INET` for IPv4, `127.0.0.1` for IP, and port 8080.
3. Connect to the server using `connect()` with the specified server address.
4. Prompt the user to enter data and read it using `fgets()`.
5. Send the data to the server using `send()`.
6. Close the socket (`sockfd`) after sending the data.
7. End the program after completing the data transmission.

//**Client (TCP)**

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

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

    int sockfd = socket(AF_INET,SOCK_STREAM,0);
    if(sockfd == -1)
        printf("Error in socket creation!\n");

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_port = htons(8080);

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

    printf("Enter data to be sent: ");
    fgets(buf,sizeof(buf),stdin);

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

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

OUTPUT

//Terminal 1

```
$ gcc -o server tcpserver.c
$ ./server
_
```

//Terminal 2

```
$ gcc -o client tcpclient.c
$ ./client
Enter data to be sent: Hello
```

//Terminal 1

```
$ gcc -o server tcpserver.c
$ ./server
Message from client: Hello
```

RESULT

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

PROGRAM

//Server (UDP)

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

int main(){
    char buf[100];
    int k;
    struct sockaddr_in server, client;
    socklen_t client_len;

    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd == -1)
        printf("Error in socket creation!\n");

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(8080);
    client_len = sizeof(client);

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

    k = recvfrom(sockfd, buf, sizeof(buf), 0, (struct sockaddr *)&client, &client_len);
    if(k == -1)
        printf("Error in recieving!\n");

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

Experiment No.3 **Date: 08-08-2024**

USER DATAGRAM PROTOCOL

AIM

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

ALGORITHM

Server:

1. Create a socket using `socket()` for UDP communication.
2. Initialize the `server` structure with `AF_INET` for IPv4, `INADDR_ANY` for any incoming address, and port 8080.
3. Bind the socket to the server address using `bind()`.
4. Define `client_len` to hold the length of the client address structure.
5. Receive data from a client using `recvfrom()` and store it in a buffer.
6. Print the received message to the console.
7. Close the socket (`sockfd`) to release resources.
8. End the program, completing the UDP communication.

Client:

1. Create a socket using `socket()` for UDP communication.
2. Initialize the `server` structure with `AF_INET` for IPv4, `127.0.0.1` as IP, and port 8080.
3. Prompt the user to enter a message and read it into a buffer.
4. Send the message to the server using `sendto()` with the server's address.
5. Close the socket (`sockfd`) to release resources.
6. End the program after the message is sent.

//Client (UDP)

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

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

    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if(sockfd == -1)
        printf("Error in socket creation!\n");

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_port = htons(8080);

    printf("Enter message to be sent: ");
    fgets(buf, 100, stdin);
    k = sendto(sockfd, buf, strlen(buf), 0, (struct sockaddr *)&server, sizeof(server));
    if(k == -1)
        printf("Error in sending!\n");

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

OUTPUT

```
//Terminal 1
$ gcc -o server udpserver.c
$ ./server
_

//Terminal 2
$ gcc -o client udpclient.c
$ ./client
Enter data to be sent: Hi

//Terminal 1
$ gcc -o server easyudpserver.c
$ ./server
Message from client: Hi
```

RESULT

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

PROGRAM

//Server (Stop and Wait)

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

#define BUFFER_SIZE 100
#define PORT 8080

int main() {
    int sockfd;
    struct sockaddr_in serverAddr, clientAddr;
    char buffer[BUFFER_SIZE];
    socklen_t addr_size;

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Error in socket creation");
        exit(EXIT_FAILURE);
    }

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

    if (bind(sockfd, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) < 0) {
        perror("Error in binding");
        close(sockfd);
        exit(EXIT_FAILURE);
    }

    printf("Receiver is waiting for frames...\n");
    while (1) {
        addr_size = sizeof(clientAddr);
        memset(buffer, 0, BUFFER_SIZE);
        int recv_len = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, (struct sockaddr
*)&clientAddr, &addr_size);
        if (recv_len < 0) {
            perror("Error receiving data");
            close(sockfd);
            exit(EXIT_FAILURE);
        }

        buffer[recv_len] = '\0';
        printf("Received: %s\n", buffer);

        int frame_number;
        sscanf(buffer, "Frame %d", &frame_number);

        snprintf(buffer, BUFFER_SIZE, "Ack for frame %d", frame_number);
```

Experiment No.4 **Date: 29-08-2024**

STOP AND WAIT PROTOCOL

AIM

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

ALGORITHM

Server:

1. Create a UDP socket using socket().
2. Initialize the `server` structure with `AF_INET` for IPv4, `INADDR_ANY` for any incoming address, and port 8080.
3. Bind the socket to the server address using `bind()`.
4. Create a loop to continuously receive frames from the client.
5. Clear the buffer for each frame. Receive a frame from the sender.
   Extract the frame number from the received data.
6. Prepare an acknowledgment message that includes the frame number. Send the acknowledgment back to the sender.
7. Repeat steps 3-4 for each incoming frame.
8. Close the socket when the communication ends.

Client:

1. Create a socket using `socket()` for UDP communication.
2. Initialize the `server` structure with `AF_INET` for IPv4, `127.0.0.1` as IP, and port 8080.
3. Prompt the user for the total number of frames to send to set the frame count.
4. For each frame, format the frame message and send it to the receiver. Clear the buffer to prepare for acknowledgment.
5. Wait for an acknowledgment from the receiver and print the details.
6. Introduce a 1-second delay to simulate Stop-and-Wait.
7. Repeat steps 3–5 until all frames are sent and acknowledged.
8. Close the socket after completing transmission.

```
       if (sendto(sockfd, buffer, strlen(buffer), 0, (struct sockaddr *)&clientAddr, 
addr_size) < 0) {
            perror("Error sending acknowledgment");
            close(sockfd);
            exit(EXIT_FAILURE);
        }
        printf("Ack sent for frame %d\n", frame_number);
    }

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

//**Client (Stop and Wait)**

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

#define BUFFER_SIZE 100
#define PORT 8080
#define IP_ADDRESS "127.0.0.1"

int main() {
    int sockfd;
    struct sockaddr_in receiverAddr;
    char buffer[BUFFER_SIZE];
    int noframes, i = 0;

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Error in socket creation");
        exit(EXIT_FAILURE);
    }

    receiverAddr.sin_family = AF_INET;
    receiverAddr.sin_port = htons(PORT);
    receiverAddr.sin_addr.s_addr = inet_addr(IP_ADDRESS);

    printf("Enter the number of frames: ");
    scanf("%d", &noframes);

    while (noframes > 0) {

        snprintf(buffer, BUFFER_SIZE, "Frame %d", i);
        if (sendto(sockfd, buffer, strlen(buffer), 0, (struct sockaddr *)&receiverAddr, 
sizeof(receiverAddr)) < 0) {
            perror("Error sending frame");
            close(sockfd);
            exit(EXIT_FAILURE);
        }
        printf("Sent frame %d\n", i);
        memset(buffer, 0, BUFFER_SIZE);
        int recv_len = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, NULL, NULL);
```

```c
        if (recv_len < 0) {
            perror("Error receiving acknowledgment");
            close(sockfd);
            exit(EXIT_FAILURE);
        }

        buffer[recv_len] = '\0';
        printf("Ack received: %s\n", buffer);

        noframes--;
        i++;

        sleep(1);
    }
    printf("End of Stop-and-Wait protocol\n");
    close(sockfd);
    return 0;
}
```

OUTPUT

```
//Terminal 1
$ gcc snw_server.c -o snw_server
$ ./snw_server
Receiver is waiting for frames...
Received: Frame 0
Ack sent for frame 0
Received: Frame 1
Ack sent for frame 1
Received: Frame 2
Ack sent for frame 2
Received: Frame 3
Ack sent for frame 3

//Terminal 2
$ gcc snw_client.c -o snw_client
$ ./snw_client
Enter the number of frames: 4
Sent frame 0
Ack received: Ack for frame 0
Sent frame 1
Ack received: Ack for frame 1
Sent frame 2
Ack received: Ack for frame 2
Sent frame 3
Ack received: Ack for frame 3
End of Stop-and-Wait protocol
```

RESULT

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

PROGRAM

//Server (Go Back N)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <fcntl.h>
int main()
{
    int s_sock, c_sock;
    s_sock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server, other;
    memset(&server, 0, sizeof(server));
    memset(&other, 0, sizeof(other));
    server.sin_family = AF_INET;
    server.sin_port = htons(9009);
    server.sin_addr.s_addr = INADDR_ANY;
    socklen_t add;

    if (bind(s_sock, (struct sockaddr *)&server, sizeof(server)) == -1)
    {
        printf("Binding failed\n");
        return 0;
    }
    printf("\tServer Up\n Go back n (n=3) used to send 10 messages \n\n");
    listen(s_sock, 10);
    add = sizeof(other);
    c_sock = accept(s_sock, (struct sockaddr *)&other, &add);
    time_t t1, t2;
    char msg[50] = "server message :";
    char buff[50];
    int flag = 0;
    fd_set set1, set2, set3;
    struct timeval timeout1, timeout2, timeout3;
    int rv1, rv2, rv3;
    int i = -1;

    qq:
        i = i + 1;
        bzero(buff, sizeof(buff));
        char buff2[60];
        bzero(buff2, sizeof(buff2));
        strcpy(buff2, "server message :");
        buff2[strlen(buff2)] = i + '0';
        buff2[strlen(buff2)] = '\0';
        printf("Message sent to client :%s \n", buff2);
        write(c_sock, buff2, sizeof(buff2));
        usleep(1000);
        i = i + 1;
        bzero(buff2, sizeof(buff2));
```

Experiment No.5 **Date: 26-09-2024**

GO BACK N

AIM

To write a program to simulate the Go Back N

ALGORITHM

Server:

1. Create a client socket using the socket() function with domain AF_INET,type SOCK_STREAM, and protocol 0.

2. Initialize the client address structure (client) with zeros using memset().

3. Set the address family (AF_INET), port number (htons(9009)), and IP address (inet_addr("127.0.0.1")) in the client structure.

4. Connect the client socket to the server using the connect() function and the client structure as the socket address. If the connection fails, print an error message and exit.

5. Print a message indicating that the client is ready.

6. Initialize message buffers: msg1 for the acknowledgement message, msg2 for constructing acknowledgement messages, and buff for receiving data from the server.

7. Initialize flags: flag and flg.

8. Start a loop from 0 to 9 to receive messages and send acknowledgements.

a. Clear the buff and msg2 buffers.

b. If i is 8 and flag is 1, simulate a loss by reading from the server without storing the data.

c. Read the message from the server into the buff buffer.

d. Check if the received message is in order by comparing the last character of the message with the expected value (i + '0').

e. If the message is out of order, print a discard message and decrement by 1.

f. If the message is in order:

- Print the received message and the corresponding index.
- Print a message indicating that the acknowledgement is sent.
- Construct the acknowledgement message in msg2 by appending the acknowledgement prefix (msg1) and the index value (i + '0').
- Send the acknowledgement message to the server using the write() function.

9. Close the client socket.

10. End the program.

```
    strcpy(buff2, msg);
    buff2[strlen(msg)] = i + '0';
    printf("Message sent to client :%s \n", buff2);
    write(c_sock, buff2, sizeof(buff2));
    i = i + 1;
    usleep(1000);
qqq:
    bzero(buff2, sizeof(buff2));
    strcpy(buff2, msg);
    buff2[strlen(msg)] = i + '0';
    printf("Message sent to client :%s \n", buff2);
    write(c_sock, buff2, sizeof(buff2));
    FD_ZERO(&set1);
    FD_SET(c_sock, &set1);
    timeout1.tv_sec = 2;
    timeout1.tv_usec = 0;

    rv1 = select(c_sock + 1, &set1, NULL, NULL, &timeout1);
    if (rv1 == -1)
        perror("select error ");
    else if (rv1 == 0)
    {
        printf("Going back from %d:timeout \n", i);
        i = i - 3;

        goto qq;
    }
    else
    {
        read(c_sock, buff, sizeof(buff));
        printf("Message from Client: %s\n", buff);
        i++;
        if (i <= 9)
            goto qqq;
    }
qq2:
    FD_ZERO(&set2);
    FD_SET(c_sock, &set2);
    timeout2.tv_sec = 3;
    timeout2.tv_usec = 0;
    rv2 = select(c_sock + 1, &set2, NULL, NULL, &timeout2);
    if (rv2 == -1)
        perror("select error ");
    else if (rv2 == 0){

            printf("Going back from %d:timeout on last 2\n", i - 1);
            i = i - 2;
            bzero(buff2, sizeof(buff2));
            strcpy(buff2, msg);
            buff2[strlen(buff2)] = i + '0';
            write(c_sock, buff2, sizeof(buff2));
            usleep(1000);
            bzero(buff2, sizeof(buff2));
            i++;
            strcpy(buff2, msg);
            buff2[strlen(buff2)] = i + '0';
```

Client algorithm:

1. Include the necessary header files.

2. Declare a variable for the client socket descriptor: c_sock.

3. Create a socket using the socket() function with domain AF_INET, type SOCK_STREAM, andprotocol 0.

4. Initialize the client address structure (client) and set the port, IP address and family.

5. Connect the client socket to the server using the connect() function. If the connection fails, print an error message and exit.

6. Print a message indicating that the client is ready.

7. Initialize message buffers: msg1 for the acknowledgement message, msg2 for constructing acknowledgement messages, and buff for receiving data from the server.

8. Initialize flags: flag and flg.

9. Enter a loop from 0 to 9 to receive messages and send acknowledgements.

10. Clear the message and buffer.

11. If i is 8 and flag is 1, simulate a loss by reading from the server without storing the data.

12. Read the message from the server into the buff buffer.

13. Check if the received message is in order by comparing the last character of the message with the expected value.

14. If the message is out of order, print a discard message and decrement i by 1.

15. If the message is in order, print the received message and the corresponding index.

16. Print a message indicating that the acknowledgement is sent.

17. Construct the acknowledgement message in msg2 using the acknowledgement prefix and the index value.

18. Send the acknowledgement message to the server using the write() function.

19. Close the client socket.

20. End the program

```
                write(c_sock, buff2, sizeof(buff2));
                goto qq2;
        }
        else{
            read(c_sock, buff, sizeof(buff));
            printf("Message from Client: %s\n", buff);
            bzero(buff, sizeof(buff));
            read(c_sock, buff, sizeof(buff));
            printf("Message from Client: %s\n", buff);
        }
        close(c_sock);
        close(s_sock);
        return 0;
}
```

//Client(Go Back N)

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

int main(){
    int c_sock;
    c_sock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in client;
    memset(&client, 0, sizeof(client));
    client.sin_family = AF_INET;
    client.sin_port = htons(9009);
    client.sin_addr.s_addr = inet_addr("127.0.0.1");
    if(connect(c_sock, (struct sockaddr *)&client, sizeof(client))==-1){
            printf("Connection failed");
            return 0;
    }
    printf("\nClient -with individual acknowledgement scheme\n\n");
    char msg1[50] = "acknowledgement of :";
    char msg2[50];
    char buff[100];
    int flag = 1, flg = 1;
    for (int i = 0; i <= 9; i++){
            flg = 1;
            bzero(buff, sizeof(buff));
            bzero(msg2, sizeof(msg2));
            if (i == 8 && flag == 1){
                    printf("here\n");
                    flag = 0;
                    read(c_sock, buff, sizeof(buff));
            }
            int n = read(c_sock, buff, sizeof(buff));
            if (buff[strlen(buff) - 1] != i + '0'){
                printf("Discarded as out of order \n");
                i--;
            }
```

```
        else{
            printf("Message received from server:%s \t%d\n",buff, i);
            printf("Acknowledgement sent for message \n");
            strcpy(msg2, msg1);
            msg2[strlen(msg2)] = i + '0';
            write(c_sock, msg2, sizeof(msg2));
         }
    }
    close(c_sock);
    return 0;
}
```

OUTPUT

```
//Terminal 1
$ gcc -o server gbnserver.c
$ ./server
        Server Up
 Go back n (n=3) used to send 10 messages

//Terminal 2
$ gcc -o client gbnclient.c
$ ./client

Client -with individual acknowledgement scheme

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

```
//Terminal 1
$ gcc -o server gbnserver.c
$ ./server
        Server Up
 Go back n (n=3) used to send 10 messages

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

RESULT

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

PROGRAM

//Server (Selective Repeat)

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

void rsendd(int ch, int c_sock) {
    char buff2[60];
    bzero(buff2, sizeof(buff2));
    strcpy(buff2, "reserver message :");
    buff2[strlen(buff2)] = (ch) + '0';
    buff2[strlen(buff2)] = '\0';
    printf("Resending Message to client :%s \n", buff2);
    write(c_sock, buff2, sizeof(buff2));
    usleep(1000);
}
int main() {
    int s_sock, c_sock;
    s_sock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server, other;
    memset(&server, 0, sizeof(server));
    memset(&other, 0, sizeof(other));
    server.sin_family = AF_INET;
    server.sin_port = htons(9009);
    server.sin_addr.s_addr = INADDR_ANY;
    socklen_t add;

    if (bind(s_sock, (struct sockaddr *)&server, sizeof(server)) == -1) {
        printf("Binding failed\n");
        return 0;
    }

    printf("Server Up\nSelective repeat scheme\n\n");
    listen(s_sock, 10);
    add = sizeof(other);
    c_sock = accept(s_sock, (struct sockaddr *)&other, &add);

    time_t t1, t2;
    char msg[50] = "server message :";
    char buff[50];
    int flag = 0;
    fd_set set1, set2, set3;
    struct timeval timeout1, timeout2, timeout3;
    int rv1, rv2, rv3;
    int tot = 0;
    int ok[20];

    memset(ok, 0, sizeof(ok));
```

Experiment No.6 **Date: 26-09-2024**

SELECTIVE REPEAT

AIM

To write a program to simulate the Selective Repeat protocol.

ALGORITHM

Server:

1. Initialize Socket:

a. Create and bind a UDP socket to the server's IP and port.

2. Set Initial Frame:

a. Initialize expected_frame to 0.

3. Receive Frames:

a. Loop until expected_frame reaches MAX_FRAMES:

4. Receive frame from the sender

a. If frame == expected_frame:

Send ACK, increment expected_frame.

b. If frame > expected_frame:

Send NACK with expected_frame.

c. If frame < expected_frame:

Send duplicate ACK for frame.

5. Cleanup:

a. Close the socket.

6. End

```
    while (tot < 9) {
        int toti = tot;
        for (int j = (0 + toti); j < (3 + toti); j++) {
            bzero(buff, sizeof(buff));
            char buff2[60];
            bzero(buff2, sizeof(buff2));
            strcpy(buff2, "server message :");
            buff2[strlen(buff2)] = (j) + '0';
            buff2[strlen(buff2)] = '\0';
            printf("Message sent to client :%s \t%d\t%d\n", buff2, tot, j);
            write(c_sock, buff2, sizeof(buff2));
            usleep(1000);
        }
        for (int k = 0 + toti; k < (toti + 3); k++)
        {
        qq: FD_ZERO(&set1);
            FD_SET(c_sock, &set1);
            timeout1.tv_sec = 2;
            timeout1.tv_usec = 0;

            rv1 = select(c_sock + 1, &set1, NULL, NULL, &timeout1);
            if (rv1 == -1)
                perror("select error ");
            else if (rv1 == 0) {
                printf("Timeout for message :%d \n", k);
                rsendd(k, c_sock);
                goto qq;
            }
            else {
                read(c_sock, buff, sizeof(buff));
                printf("Message from Client: %s\n", buff);
                if (buff[0] == 'n')
                {
                    printf(" corrupt message acknowledgement (msg %d) \n",
buff[strlen(buff) - 1] - '0');
                    rsendd((buff[strlen(buff) - 1] - '0'), c_sock);
                    goto qq;
                }
                else
                    tot++;
            }
        }
    }
    close(c_sock);
    close(s_sock);
    return 0;
}
```

Client algorithm:

1. Initialize Socket

a. Create a UDP socket and configure it to communicate with the server.

2. Set Initial Frame Variables

a. Set `base` to 0, `next_seq_num` to 0, and `nack_received` to `false`.

3. Send Frames

a. Loop until all frames are acknowledged (`base < MAX_FRAMES`):

i. If no NACK received (`nack_received == false`):

1. Send frames from `base` to `base + WINDOW_SIZE` (up to `MAX_FRAMES`).

2. Increment `next_seq_num` for each frame sent.

ii. If NACK received (`nack_received == true`):

1. Resend the specific frame indicated by `nack`.

2. Reset `nack_received` to `false`.

4. Receive Acknowledgment

a. Wait for `ACK` or `NACK` from the server.

b. If ACK received for frame ≥ `base`:

i. Advance `base` to `ack + 1`.

c. If NACK received for frame < `base`:

i. Set `nack` to the frame number and mark `nack_received` as `true`.

5. Cleanup

a. Close the socket.

6. End

//Client (Selective Repeat)

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

int isfaulty() {
    int d = rand() % 4;
    return (d > 2);
}
int main() {
    srand(time(0));
    int c_sock;
    c_sock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in client;
    memset(&client, 0, sizeof(client));
    client.sin_family = AF_INET;
    client.sin_port = htons(9009);
    client.sin_addr.s_addr = inet_addr("127.0.0.1");

    if (connect(c_sock, (struct sockaddr *)&client, sizeof(client)) == -1) {
        printf("Connection failed");
        return 0;
    }
    printf("\nClient -with individual acknowledgement scheme\n\n");
    char msg1[50] = "acknowledgement of ";
    char msg3[50] = "negative ack ";
    char msg2[50];
    char buff[100];
    int count = -1, flag = 1;
    while (count < 8) {
        bzero(buff, sizeof(buff));
        bzero(msg2, sizeof(msg2));

        if (count == 7 && flag == 1) {
            printf("here\n");
            flag = 0;
            read(c_sock, buff, sizeof(buff));
            continue;
        }

        int n = read(c_sock, buff, sizeof(buff));
        char i = buff[strlen(buff) - 1];
        printf("Message received from server : %s \n", buff);
        int isfault = isfaulty();
        printf("corruption status : %d \n", isfault);
        printf("Response/acknowledgement sent for message \n");

        if (isfault)
            strcpy(msg2, msg3);
        else
        {
```

```
            strcpy(msg2, msg1);
            count++;
        }
        msg2[strlen(msg2)] = i;
        write(c_sock, msg2, sizeof(msg2));
    }
    close(c_sock);
    return 0;
}
```

OUTPUT

```
//Terminal 1
$ gcc -o server sr_server.c
$ ./server
Server Up
Selective repeat scheme

Message sent to client :server message :0       0       0
Message sent to client :server message :1       0       1
Message sent to client :server message :2       0       2
Message from Client: negative ack 0
 corrupt message acknowledgement (msg 0)
Resending Message to client :reserver message :0
Message from Client: acknowledgement of 1
Message from Client: acknowledgement of 2
Message from Client: acknowledgement of 0
Message sent to client :server message :3       3       3
Message sent to client :server message :4       3       4
Message sent to client :server message :5       3       5
Message from Client: acknowledgement of 3
Message from Client: acknowledgement of 4
Message from Client: negative ack 5
 corrupt message acknowledgement (msg 5)
Resending Message to client :reserver message :5
Message from Client: acknowledgement of 5
Message sent to client :server message :6       6       6
Message sent to client :server message :7       6       7
Message sent to client :server message :8       6       8
Message from Client: acknowledgement of 6
Message from Client: acknowledgement of 7
Timeout for message :8
Resending Message to client :reserver message :8
Message from Client: acknowledgement of 8
```

```
//Terminal 2
$ gcc -o client sr_client.c
$ ./client

Client -with individual acknowledgement scheme

Message received from server : server message :0
corruption status : 1
Response/acknowledgement sent for message
Message received from server : server message :1
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :2
corruption status : 0
Response/acknowledgement sent for message
Message received from server : reserver message :0
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :3
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :4
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :5
corruption status : 1
Response/acknowledgement sent for message
Message received from server : reserver message :5
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :6
corruption status : 0
Response/acknowledgement sent for message
Message received from server : server message :7
corruption status : 0
Response/acknowledgement sent for message
here
Message received from server : reserver message :8
corruption status : 0
Response/acknowledgement sent for message
```

RESULT

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

PROGRAM

//Distance Vector Routing

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

struct Link {
    int hop, dest, wt;
};

struct Network {
    int H, L;
    struct Link *link;
};

int main() {
    int H, L, S, i, j;

    printf("Distance Vector Routing using Bellman Ford Algorithm:\n");
    printf("Enter number of hops: ");
    scanf("%d", &H);

    printf("Enter number of links: ");
    scanf("%d", &L);

    printf("Enter the source node (0 to %d): ", H - 1);
    scanf("%d", &S);

    struct Network *n = (struct Network *)malloc(sizeof(struct Network));
    n->H = H;
    n->L = L;
    n->link = (struct Link *)malloc(n->L * sizeof(struct Link));

    for (i = 0; i < L; i++) {
        printf("\nLink %d: enter source, destination and weight: \n", i + 1);
        scanf("%d", &n->link[i].hop);
        scanf("%d", &n->link[i].dest);
        scanf("%d", &n->link[i].wt);
    }

    int *dist = (int *)malloc(H * sizeof(int));
    for (i = 0; i < H; i++)
        dist[i] = INT_MAX;

    dist[S] = 0;

    for (i = 1; i < H; i++) {
        for (j = 0; j < L; j++) {
            int u = n->link[j].hop;
            int v = n->link[j].dest;
            int wt = n->link[j].wt;

            if (dist[u] != INT_MAX && dist[u] + wt < dist[v]) {
                dist[v] = dist[u] + wt;
            }
        }
    }
```

Experiment No.7 **Date: 10-10-2024**

DISTANCE VECTOR ROUTING

AIM

To simulate Distance Vector Routing protocol.

ALGORITHM

1. Start

2. Input initialization:

a. Prompt the user to enter the number of hops(H). and the number of links(L).

b. Create a network structure to hold the links.

c. Allocate memory for the links.

3. Link input:

a. For each link from 1 to L:

i. Prompt user to enter the source hop, destination hop and weight of the link.

ii. Store these values in the link array of the network structure.

4. Distance initialization:

a. Create an array dist[] to store distances from the source node to all other nodes.

b. Initialize all distances in dist[] to INT_MAX(infinity) to represent that all nodes are initially unreachable.

c. Set the distance from source node to itself to 0.

5. Relaxation process:

a. For each hop(H-1) times:

i. For each link L:

1. Retrieve the source(u), destination(v), weight(wt) of the link.

2. If the distance to the source(dist[u]) plus the weight(wt) is less than the distance to the destination(dist[v]), update distance to destination(dist[v] = dist[u] + wt)

6. Negative cycle detection:

a. For each link(L):

i. Retrieve the source(u), destination(v), weight(wt) of the link.

ii. If the distance to the source(dist[u]) plus the weight(wt) is less than the distance to the destination(dist[v]), print a message indicating that network contains a negative weight cycle.

7. Output result:

a. Print the hop number and the corresponding distance from source for each hop.

8. Stop

```
for (i = 0; i < L; i++) {
        int u = n->link[i].hop;
        int v = n->link[i].dest;
        int wt = n->link[i].wt;
        if (dist[u] != INT_MAX && dist[u] + wt < dist[v]) {
            printf("Network contains negative weight cycle\n");
            free(dist);
            free(n->link);
            free(n);
            return 0;
        }
    }
    printf("\nHop\tDistance from source\n");
    for (i = 0; i < H; i++) {
        printf("%d \t %d\n", i, dist[i]);
    }
    free(dist);
    free(n->link);
    free(n);
    return 0;
}
```

OUTPUT

```
$ gcc dvr.c
$ ./a.out
Distance Vector Routing using Bellman Ford Algorithm:
Enter number of hops: 5
Enter number of links: 6
Enter the source node (0 to 4): 0

Link 1: enter source, destination and weight:
0 1 2

Link 2: enter source, destination and weight:
0 2 4

Link 3: enter source, destination and weight:
1 3 2

Link 4: enter source, destination and weight:
2 4 3

Link 5: enter source, destination and weight:
2 3 4

Link 6: enter source, destination and weight:
4 3 -5

Hop     Distance from source
0        0
1        2
2        4
3        2
4        7
```

RESULT

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

PROGRAM

//Link State Routing

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

int main() {
    int H, i, j, adj[50][50], d;
    printf("LINK STATE ROUTING PROTOCOL:\nEnter the number of hops: ");
    scanf("%d", &H);

    printf("Enter the adjacency matrix:\n");
    for (i = 0; i < H; i++)
        for (j = 0; j < H; j++)
            scanf("%d", &adj[i][j]);

    int dist[H], visited[H], round, v;
    for (i = 0; i < H; i++) {
        dist[i] = INT_MAX;
        visited[i] = 0;
    }

    dist[0] = 0;
    for (round = 0; round < H - 1; round++) {
        int min = INT_MAX, min_index;

        for (v = 0; v < H; v++) {
            if (visited[v] == 0 && dist[v] < min) {
                min = dist[v];
                min_index = v;
            }
        }

        visited[min_index] = 1;

        for (d = 0; d < H; d++) {
            if (!visited[d] && adj[min_index][d] && dist[min_index] != INT_MAX &&
dist[min_index] + adj[min_index][d] < dist[d]) {
                dist[d] = dist[min_index] + adj[min_index][d];
            }
        }
    }

    printf("Vertex\tDistance from Source\n");
    for (i = 0; i < H; i++) {
        printf("%d\t\t%d\n", i, dist[i]);
    }

    return 0;
}
```

Experiment No.8 **Date: 10-10-2024**

LINK STATE ROUTING

AIM

To simulate link state routing protocol.

ALGORITHM

1. Start

2. Input initialization:

a. Prompt the user to enter the number of hops(H).

b. Initialize an adjacency matrix adj to represent the graph.

3. Input adjacency matrix:

a. For each node i from 0 to H-1:

i. For each node j from 0 to H-1:

1. Input the weight of the edge from node i to node j into adj[i][j].

4. Distance and visited initialization:

a. Create an array dist[] of size H to store the shortest distances from the source(initially set all values to INT_MAX).

b. Create an array visited[] of size H to track visited nodes(initialize all values to 0).

c. Set the distance from source node to itself to 0.

5. Main loop for Dijkstra's Algorithm:

a. For node from 0 to H-2:

i. Initialize min to INT_MAX and min_index to -1

ii. For each node(v from 0 to H-1):

1. If node is not visited and its distance is less than min, update min and min_index.

iii. Mark the node at min_index as visited i.e., visited[min_index] = 1.

iv. For each node d from 0 to H-1:

1. If node is not visited and there is an edge from min_index to d and distance to min_index is not INT_MAX:

a. Update distance to node d if a shorter path is found i.e., dist[d] = dist[min_index] + adj[min_index].

6. Output results:

a. Print the vertex and the corresponding distance from the source.

7. Stop

OUTPUT

```
$ gcc lsr.c
$ ./a.out
LINK STATE ROUTING PROTOCOL:
Enter the number of hops: 5
Enter the adjacency matrix:
0 10 3 0 0
0 0 1 2 0
0 4 0 8 2
0 0 0 0 7
0 0 0 9 0
Vertex  Distance from Source
0               0
1               7
2               3
3               9
4               5
```

RESULT

The program for Link State Routing was executed and output was verified successfully.

PROGRAM

```
//Leaky Bucket Algorithm
#include<stdio.h>
#include<stdlib.h>

struct packet {
    int time;
    int size;
};

int main () {
    int i, n, k = 0;
    int bucket_size, current_bucket, output_rate;
    printf ("Enter the number of packets: ");
    scanf ("%d", &n);

    struct packet p[n];

    printf ("\nEnter the packets in the order of their arrival time: \n");
    for (i = 0; i < n; i++) {
        printf ("Enter the time and size of packet %d: ", i+1);
        scanf ("%d %d", &p[i].time, &p[i].size);
    }

    printf ("\nEnter the bucket size: ");
    scanf ("%d", &bucket_size);
    printf ("Enter the output rate: ");
    scanf ("%d", &output_rate);

    int max_time = p[n - 1].time;
    i = 0;
    current_bucket = 0;

    while (i <= max_time || current_bucket != 0) {
        printf("\nAt time %d", i);

        if (k < n && p[k].time == i) {  /
            if (bucket_size >= current_bucket + p[k].size) {
                current_bucket = current_bucket + p[k].size;
                printf("\n%d byte packet is inserted", p[k].size);
            } else {
                printf("\n%d byte packet is discarded", p[k].size);
            }
            k++;  // Move to next packet
        }

       if (current_bucket == 0) {
            printf("\nNo packets to transmit");
        } else {
            if (current_bucket >= output_rate) {
                current_bucket = current_bucket - output_rate;
                printf("\n%d bytes transferred", output_rate);
            } else {
                printf("\n%d bytes transferred", current_bucket);
                current_bucket = 0;
            }
        }
```

Experiment No.9 **Date: 17-10-2024**

LEAKY BUCKET ALGORITHM

AIM

To write a program to implement Leaky Bucket Algorithm.

ALGORITHM

1.Input:

a. Read number of packets n, packet arrival times and sizes.
b. Read bucket_size and output_rate.

2.Initialize:

a. Set max_time as the arrival time of the last packet.
b. Set current_time = 0, current_bucket = 0, and k = 0.

3.Simulate:

a. While current_time <= max_time or current_bucket > 0:
   1. If a packet arrives at current_time, insert or discard it based on bucket space.
   2. If the bucket is not empty, transfer up to output_rate bytes.
   3. Print the status of the bucket and the transmission.

4.End:

Continue until all packets are processed and the bucket is empty.

```
        printf("\nPackets in the bucket: %d byte(s)\n", current_bucket);
        i++;
    }

    return 0;
}
```

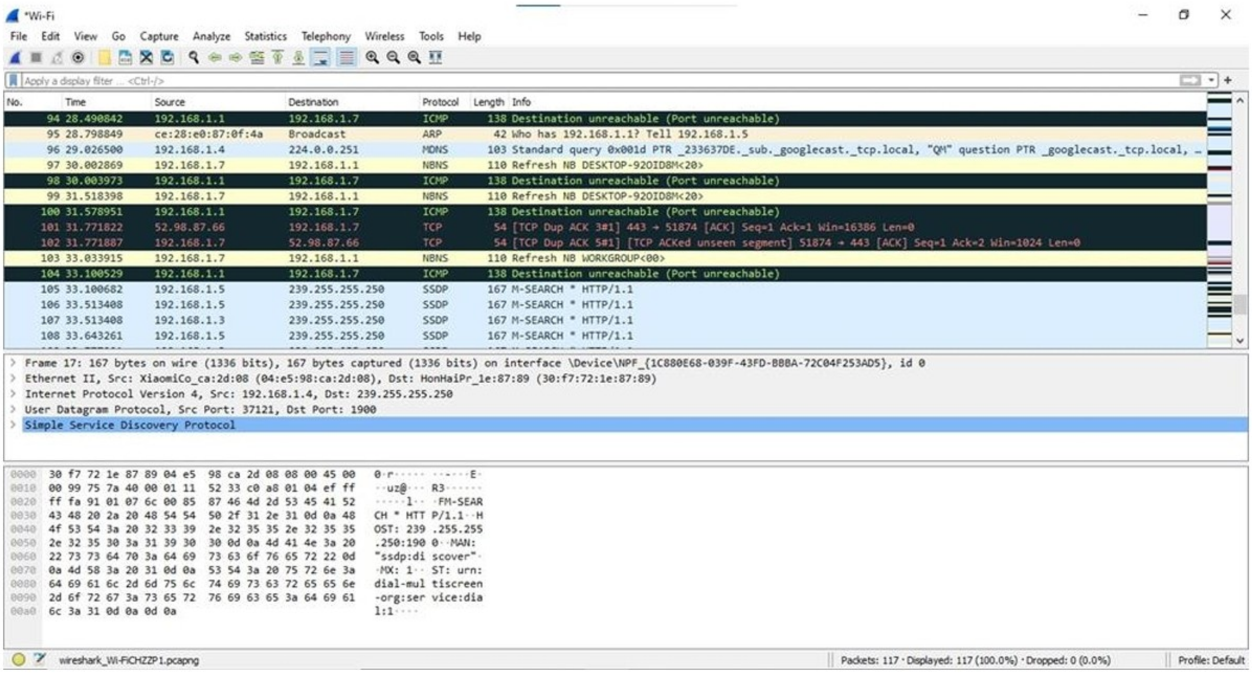
OUTPUT

```
$ gcc leakybucket.c
$ ./a.out
Enter the number of packets: 3

Enter the packets in the order of their arrival time:
Enter the time and size of packet 1: 2 20
Enter the time and size of packet 2: 3 40
Enter the time and size of packet 3: 5 50

Enter the bucket size: 100
Enter the output rate: 20

At time 0
No packets to transmit
Packets in the bucket: 0 byte(s)

At time 1
No packets to transmit
Packets in the bucket: 0 byte(s)

At time 2
20 byte packet is inserted
20 bytes transferred
Packets in the bucket: 0 byte(s)

At time 3
40 byte packet is inserted
20 bytes transferred
Packets in the bucket: 20 byte(s)

At time 4
20 bytes transferred
Packets in the bucket: 0 byte(s)

At time 5
50 byte packet is inserted
20 bytes transferred
Packets in the bucket: 30 byte(s)

At time 6
20 bytes transferred
Packets in the bucket: 10 byte(s)

At time 7
10 bytes transferred
Packets in the bucket: 0 byte(s)
```

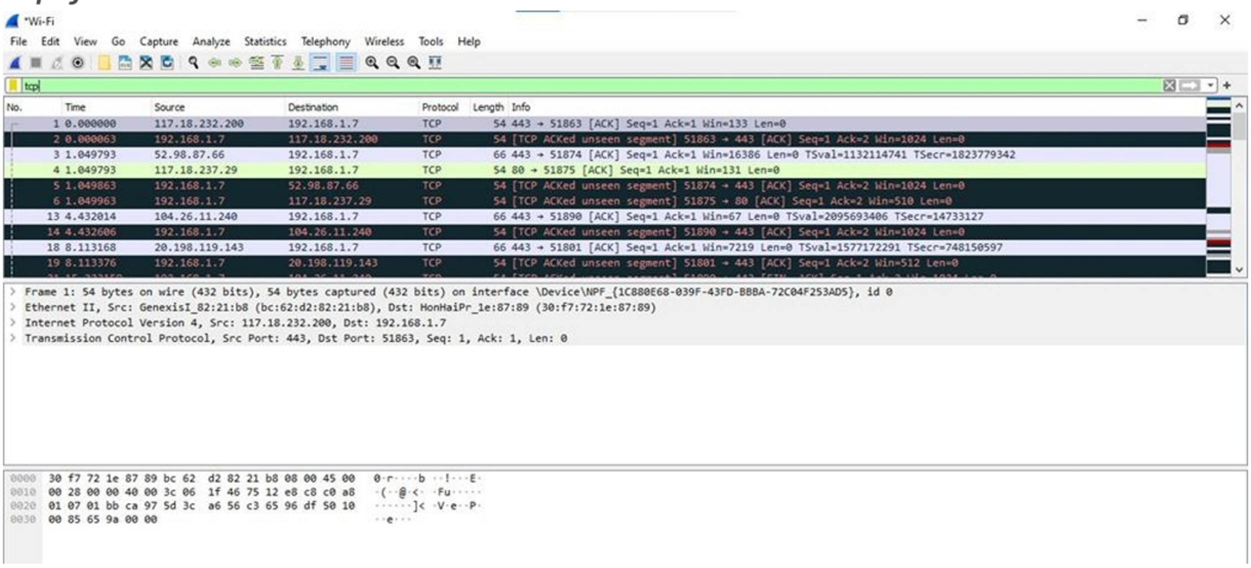
RESULT

Successfully implemented Leaky Bucket Algorithm.

OUTPUT:



Tcp filter



Experiment No.10 **Date: 07-11-2024**

WIRESHARK

AIM

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

STEPS

1.Open wireshark tool.

2.Select capture then interfaces.

3.Choose the interface to capture packets on.

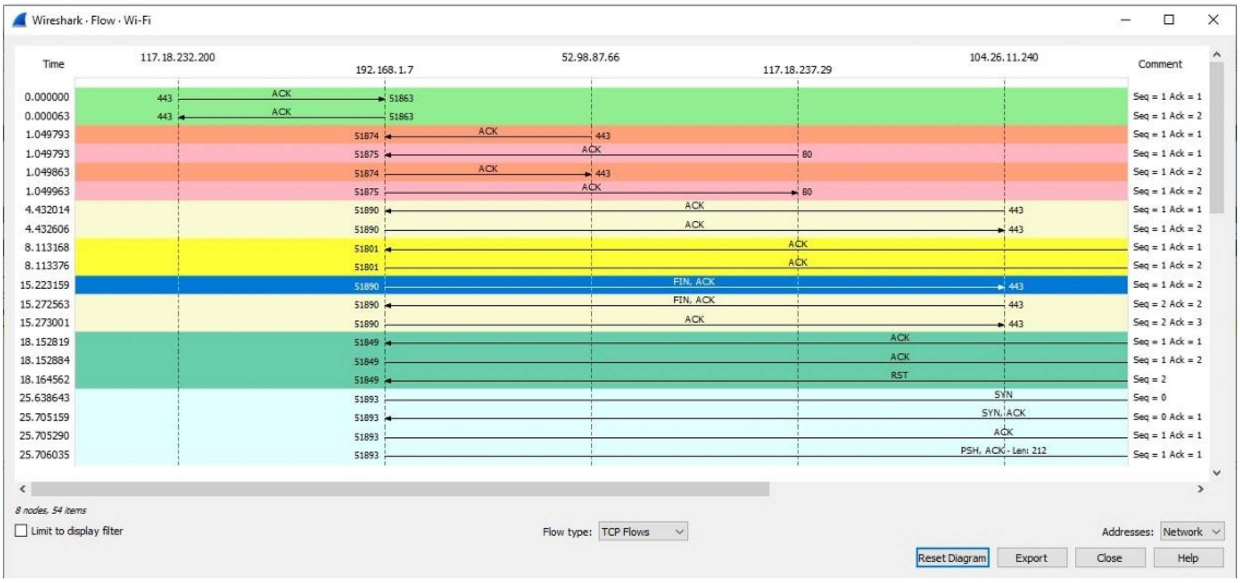
4.Click start to begin capturing.

5.Reproduce the problem.

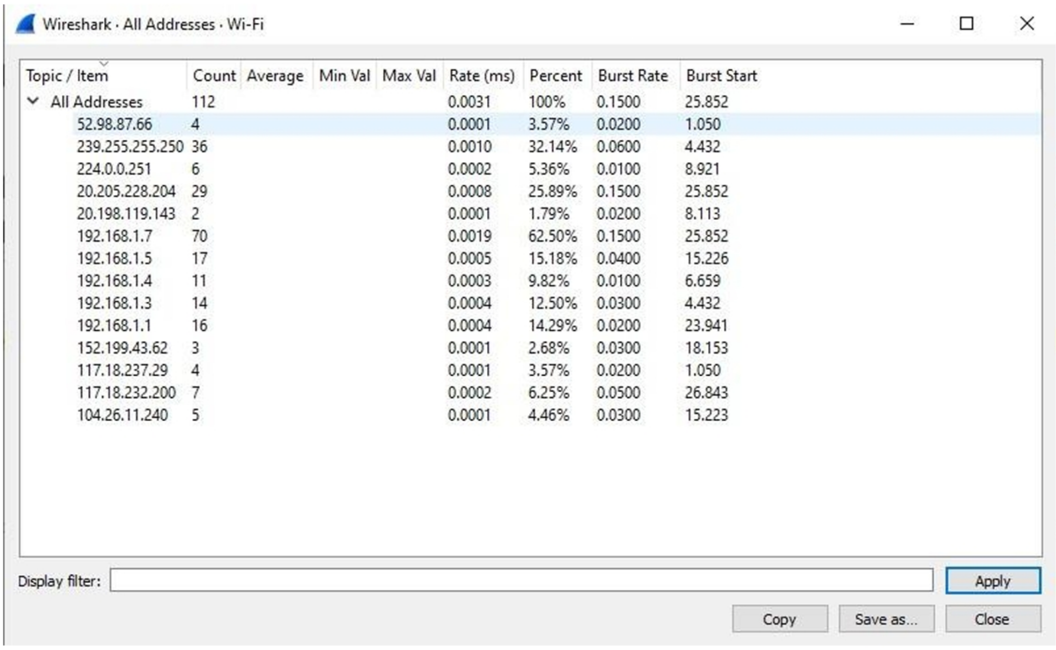
6.Click stop when the problem has been reproduces.

7.Save the packet trace

Flow graphs



Statistics



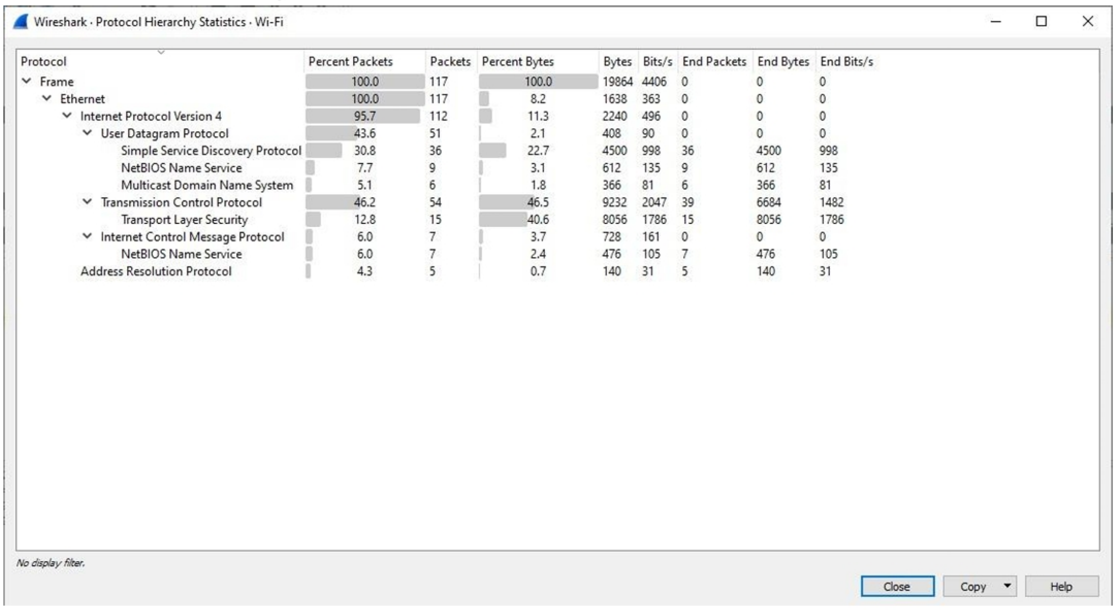
Wireshark · All Addresses · Wi-Fi

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

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.

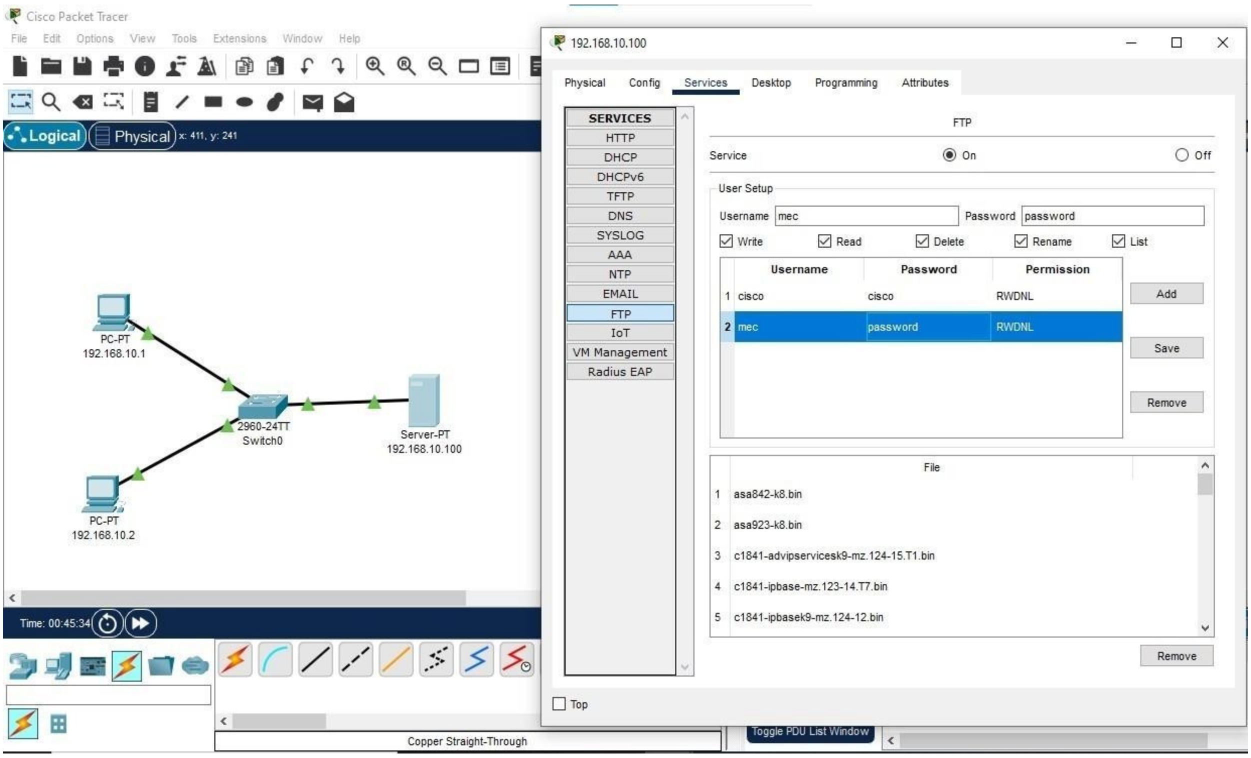
Close Copy Help

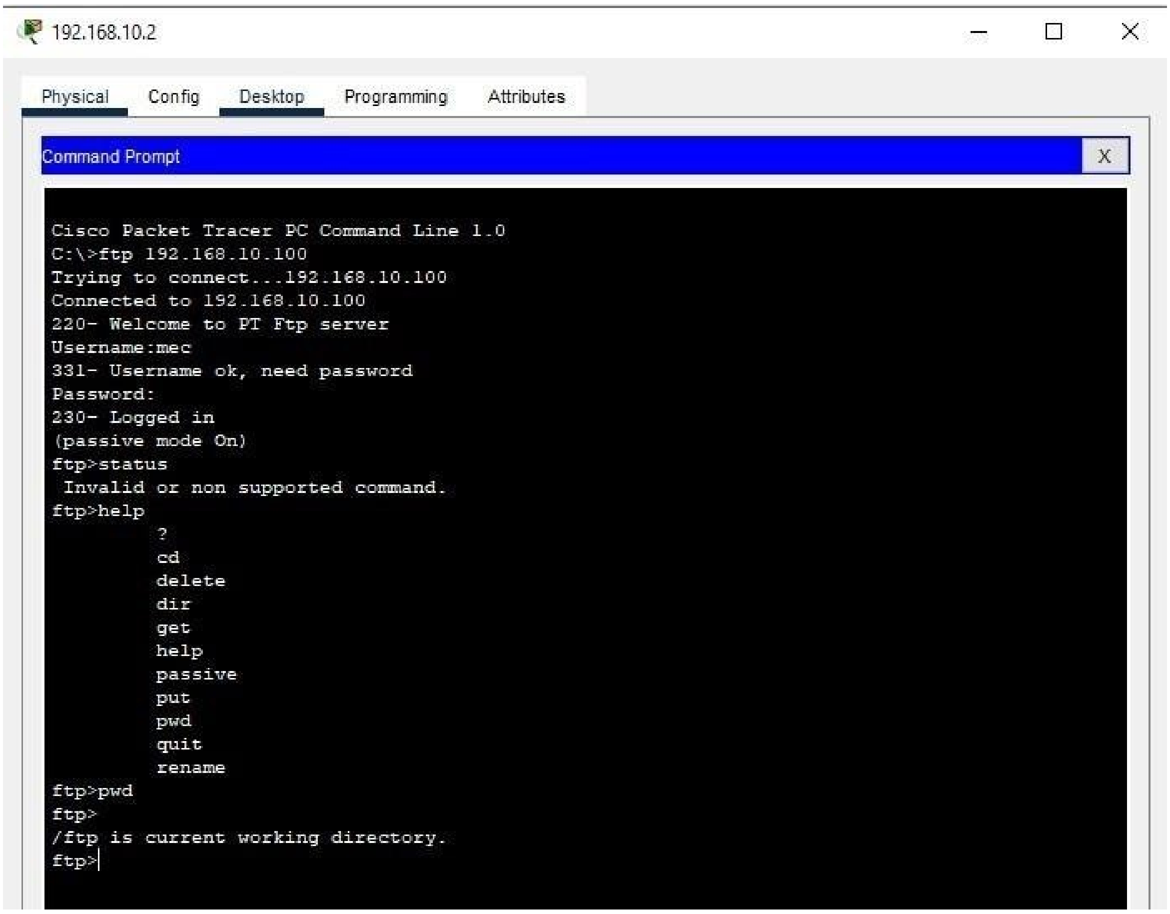
RESULT

The experiment was executed successfully..

OUTPUT

FTP:


Cisco Packet Tracer
Logical
Physical
PC-PT
192.168.10.1
2960-24TT
Switch0
Server-PT
192.168.10.100
PC-PT
192.168.10.2
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
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
Add
Save
Remove
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
Remove
Top
Copper Straight-Through
Toggle PDU List Window


192.168.10.2
Physical
Config
Desktop
Programming
Attributes
Command Prompt
X
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>

Experiment No.11 **Date: 07-11-2024**

NETWORK WITH MULTIPLE SUBNETS

AIM

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

STEPS

1.Build a Simple Network in the Logical Topology Workspace

a. Launch Packet Tracer.
  i. Launch Packet Tracer on your PC or laptop.

b. Build the topolgy.
  i. Add network devices to the workspace.
  ii. Change display names of the network devices.
  iii.Add physical cabling between the devices on the workspace.

2.Configure the workspace.

a.Configure the wireless router.

  i. Create the wireless network on the wireless router.
  ii. Click on the Save Settings tab.

b.Configure the laptop.
  i. Configure the laptop to access the wireless network.

c.Configure the PC.
  i. Configure the PC for wired network.

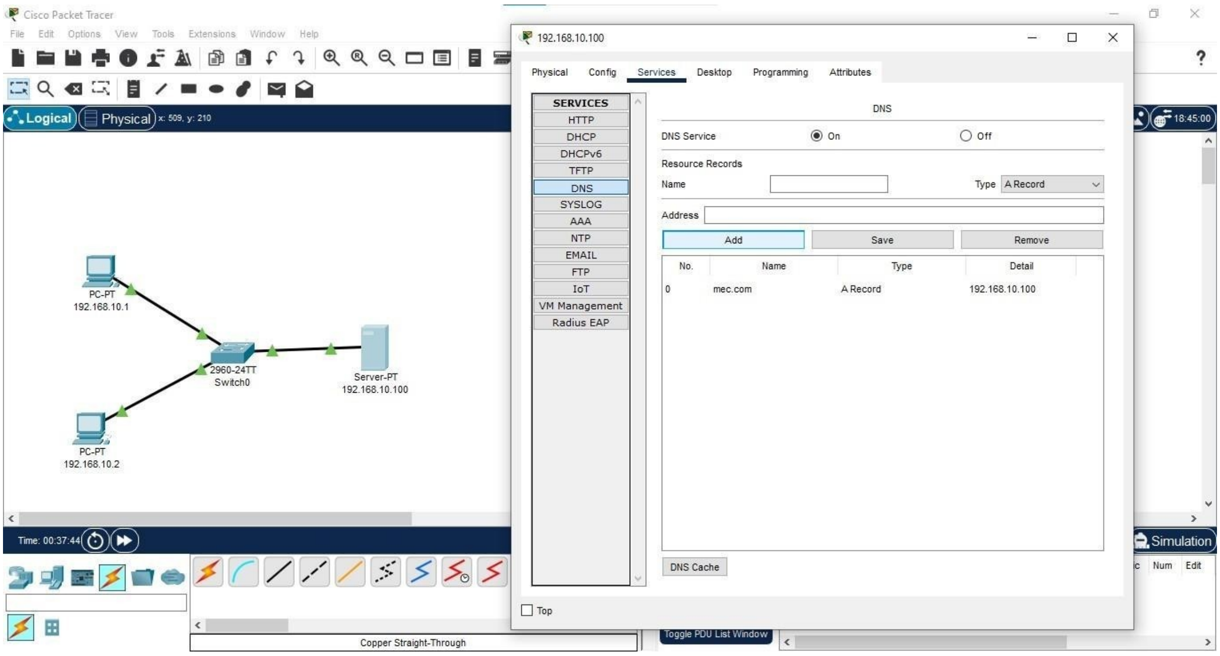
d. Configure the Internet Cloud

  i. Install network modules if necessary.
  ii. Identify the from and to ports.
  iii.Identify the type of provider.

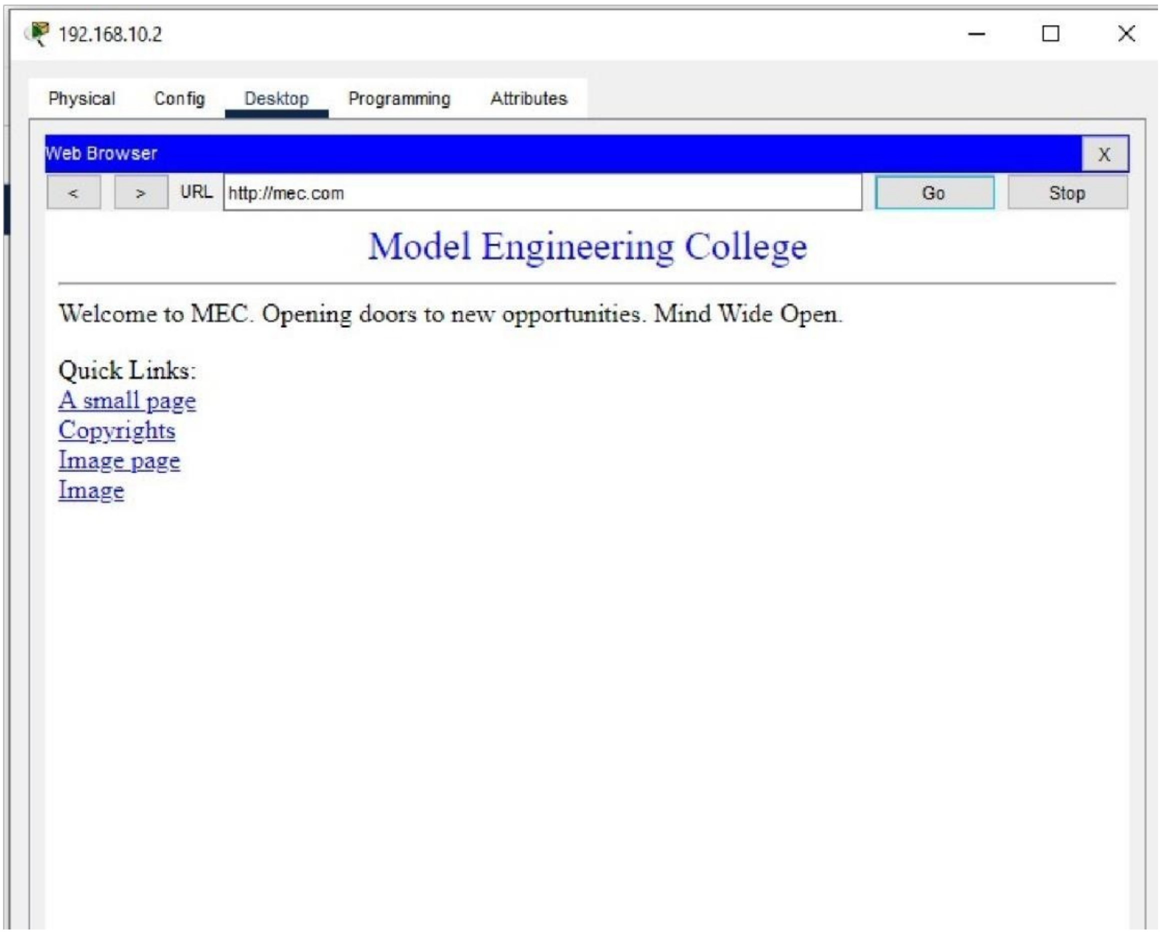
e.Configure the cisco.com server's

  i. Configure the cisco.com server as the DHCP server.
  ii. Configure the Cisco.com server as a DNS server to provide domain name to IPv4 address resolution.
  iii. Configure the Cisco.com server Global settings.
  iv.Configure the Cisco.com server FastEthernet0 Interface settings

DNS :



192.168.10.100
Physical
Config
Services
Desktop
Programming
Attributes
SERVICES
HTTP
DHCP
DHCPv6
TFTP
DNS
SYSLOG
AAA
NTP
EMAIL
FTP
IoT
VM Management
Radius EAP
DNS
DNS Service
On
Off
Resource Records
Name
Type
A Record
Address
Add
Save
Remove
No.
Name
Type
Detail
0
mec.com
A Record
192.168.10.100
DNS Cache
PC-PT
192.168.10.1
2960-24TT
Switch0
Server-PT
192.168.10.100
PC-PT
192.168.10.2
Copper Straight-Through



192.168.10.2
Physical
Config
Desktop
Programming
Attributes
Web Browser
URL
http://mec.com
Go
Stop
Model Engineering College
Welcome to MEC. Opening doors to new opportunities. Mind Wide Open.
Quick Links:
A small page
Copyrights
Image page
Image

3.Verify connectivity

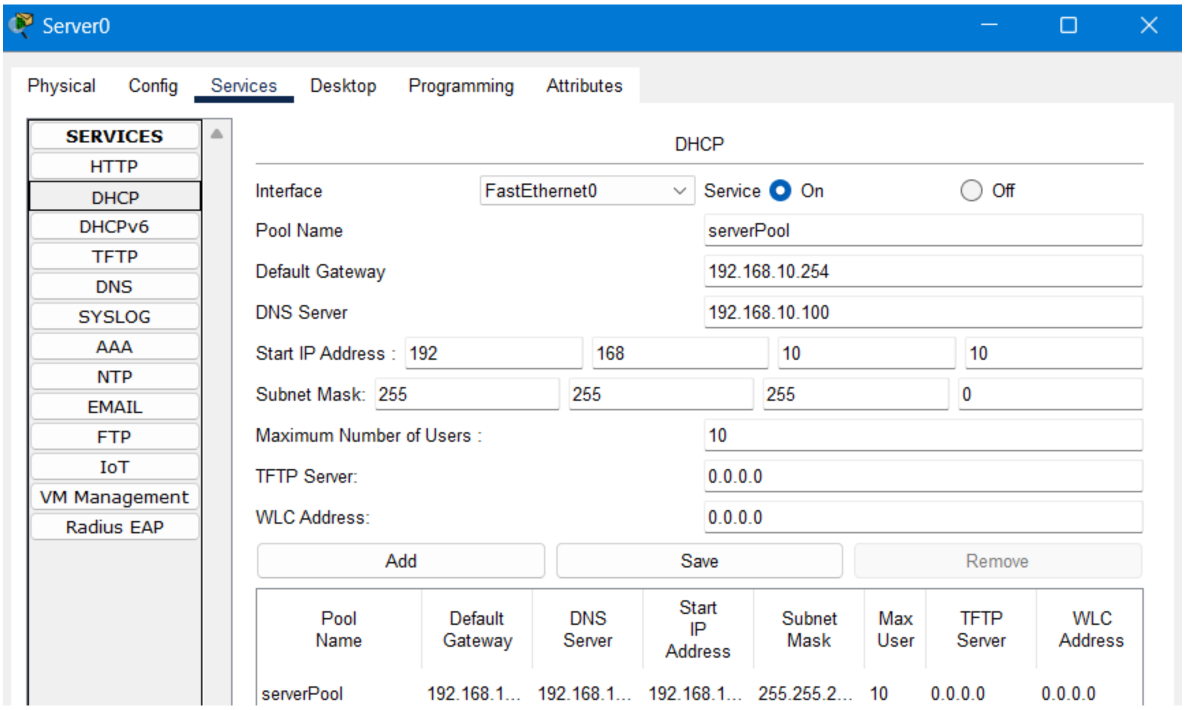
a.Refresh the IPV4 settings on the PC.

i.Verify that the PC is receiving IPV4 configuration information from DHCP.
ii.Test connectivity to the Cisco.com server from the PC.

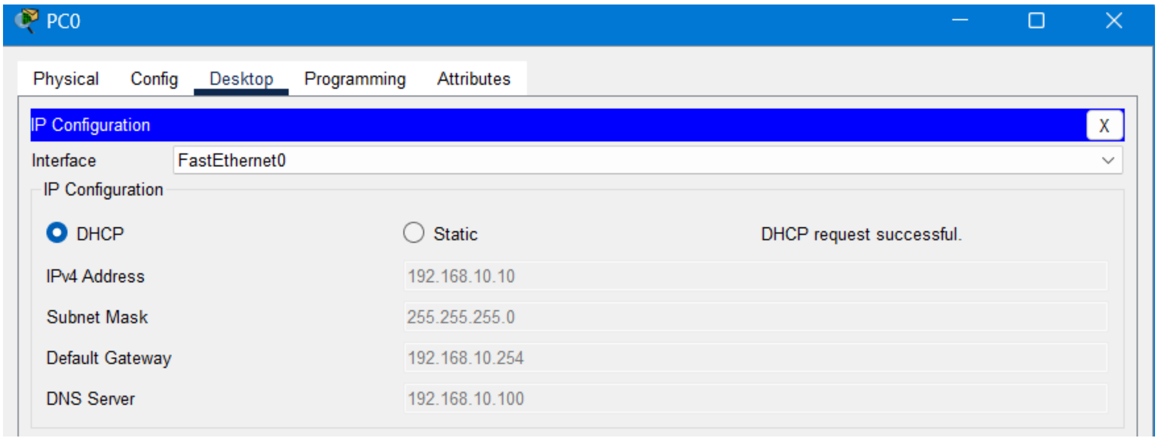
4.Save the File and Close Packet Tracer.

a.Save the File as a Packet Tracer Activity File (*.pkt)
b.Close Packet Tracer

DHCP :



Server0
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
192.168.10.254
DNS Server
192.168.10.100
Start IP Address :
192
168
10
10
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
192.168.1...
192.168.1...
192.168.1...
255.255.2...
10
0.0.0.0
0.0.0.0



PC0
Physical
Config
Desktop
Programming
Attributes
IP Configuration
X
Interface
FastEthernet0
IP Configuration
DHCP
Static
DHCP request successful.
IPv4 Address
192.168.10.10
Subnet Mask
255.255.255.0
Default Gateway
192.168.10.254
DNS Server
192.168.10.100

RESULT

The experiment was executed successfully..

PROGRAM

//Link-State.tcl

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

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

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

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

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

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

set null1 [new Agent/Null]
$ns attach-agent $n3 $null1
$ns connect $udp0 $null0
$ns connect $udp1 $null1
$ns at .1 "$cbr1 start"
$ns at .2 "$cbr0 start"
$ns at 45.0 "$cbr1 stop"
$ns at 45.1 "$cbr0 stop"
$ns at 50.0 "finish"
$ns run

AWK Program for LS
BEGIN {
    print "Performance evaluation"
    send = 0
    recv = 0
    dropped = 0
    rout = 0
}
```

Experiment No.12 **Date: 07-11-2024**

NS2 SIMULATOR

AIM

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

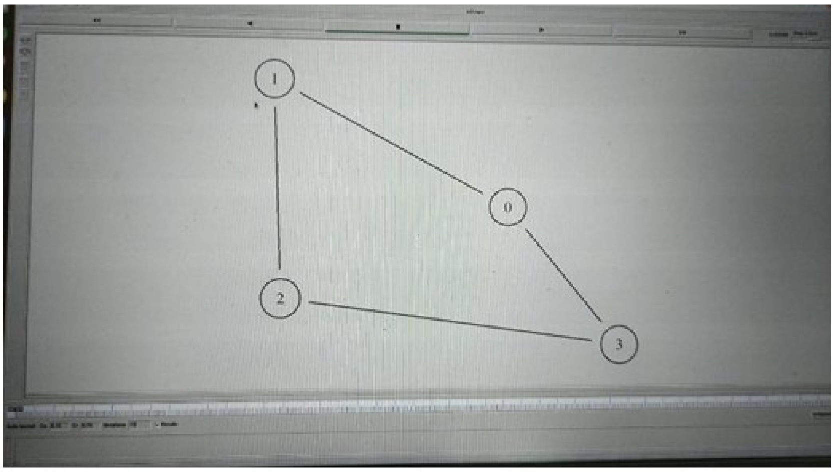
ALGORITHM

LSR :

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

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

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

OUTPUT



LSR

RESULT

The experiment was executed successfully.