COMPUTER NETWORKS LAB

III – I SEMESTER

COMPUTER SCIENCE AND ENGINEERING

(R20)

List of Experiments:

1. Study of Network devices in detail and connect the computers in Local Area Network.

2. Write a Program to implement the data link layer farming methods such as i) Character stuffing ii) bit stuffing.

3. Write a Program to implement data link layer framing method checksum.

4. Write a program for Hamming Code generation for error detection and correction.

5. Write a Program to implement on a data set of characters the three CRC polynomials – CRC 12, CRC 16 and CRC CCIP.

6. Write a Program to implement Sliding window protocol for Goback N.

7. Write a Program to implement Sliding window protocol for Selective repeat.

8. Write a Program to implement Stop and Wait Protocol.

9. Write a program for congestion control using leaky bucket algorithm.

10. Write a Program to implement Dijkstra‘s algorithm to compute the Shortest path through a graph.

11. Write a Program to implement Distance vector routing algorithm by obtaining routing table at each node (Take an example subnet graph with weights indicating delay between nodes).

12. Write a Program to implement Broadcast tree by taking subnet of hosts.

13. Wireshark

    i. Packet Capture Using Wire shark

    ii. Starting Wire shark

    iii. Viewing Captured Traffic

    iv. Analysis and Statistics & Filters.

14. How to run Nmap scan

15. Operating System Detection using Nmap

16. Do the following using NS2 Simulator

i. NS2 Simulator-Introduction

ii. Simulate to Find the Number of Packets Dropped

iii. Simulate to Find the Number of Packets Dropped by TCP/UDP

iv. Simulate to Find the Number of Packets Dropped due to Congestion

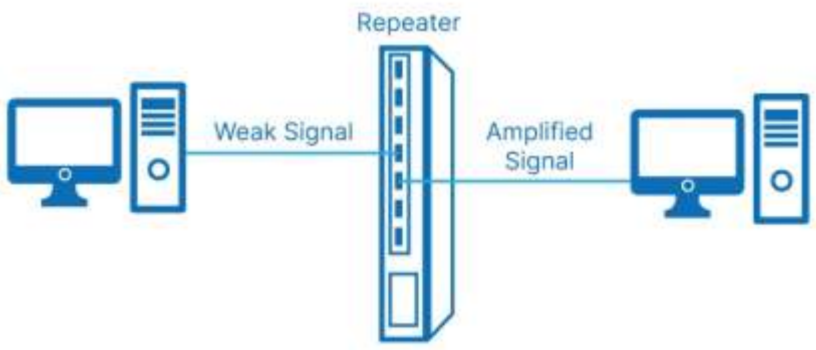
v. Simulate to Compare Data Rate& Throughput.

EXPERIMENT – 1

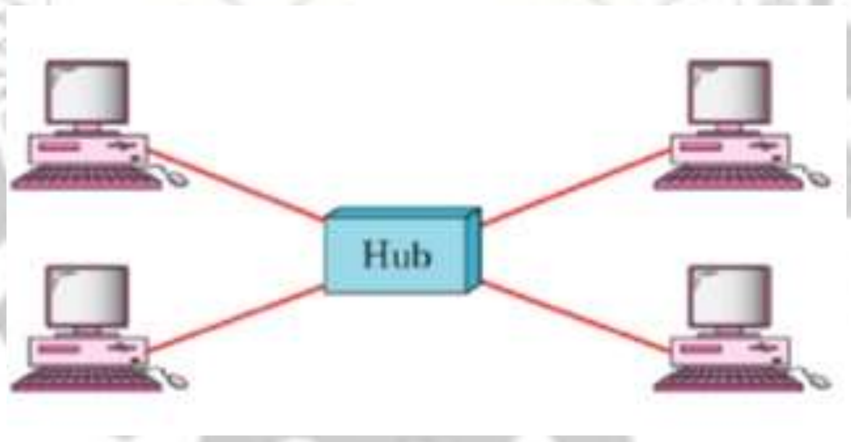
Aim: Study of the following Network Devices in detail connect the computers in Local Area Network.

- Repeater
- Hub
- Switch
- Bridge
- Router
- Gateway

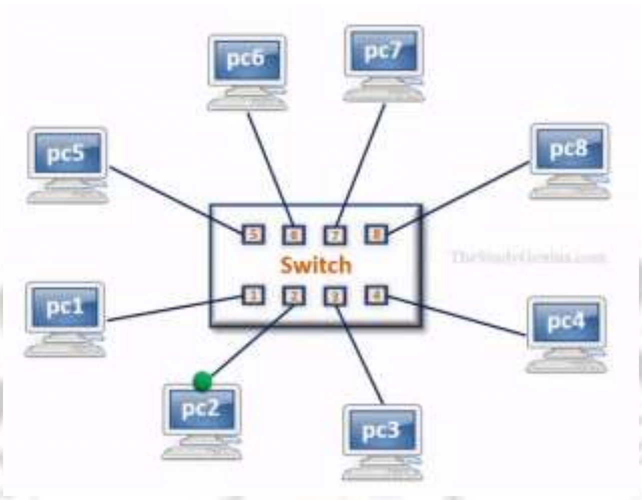
1. **Repeater:**
   Repeaters are defined as a networking device that is used to amplify and generate the incoming signal. Repeaters work at the "physical layer" of the OSI model. The main aim of using a repeater us to increase the networking distance by increasing the strength and quality of signals. Repeater have two ports, so cannot be used to connect for more than two devices.



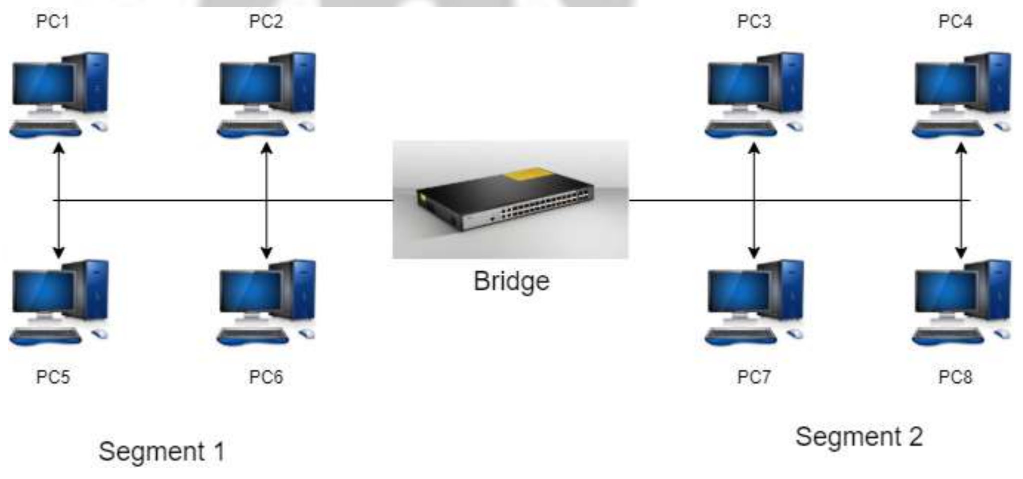
2. **Hub:**
   A hub is a basic networking device that connects multiple computers or network devices in a local area network (LAN). It operates at the physical layer (Layer 1) of the OSI model. Hub supports half-duplex transmission. It can detect collisions in the network and send the jamming signal to each part.



3. **Switch:**
   A switch is a networking device that connects multiple computers and devices within a local area network (LAN). It operates primarily at the data link layer (Layer 2) of the OSI model but can also operate at higher layers (Layer 3) in some cases.

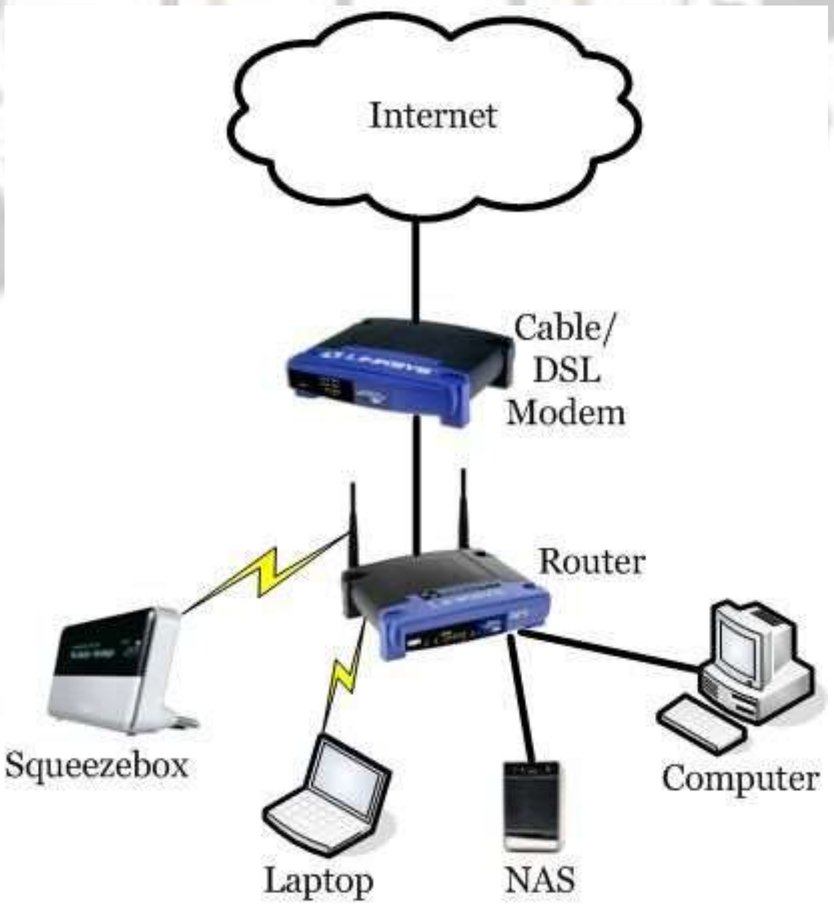


4. **Bridge:**
A bridge is a device that connects two or more network segments allowing them to communicate as if they were a single network. Bridges operate at the data link layer (Layer 2) of the OSI model. They filter and forward data between segments based on MAC addresses.



5. Router:

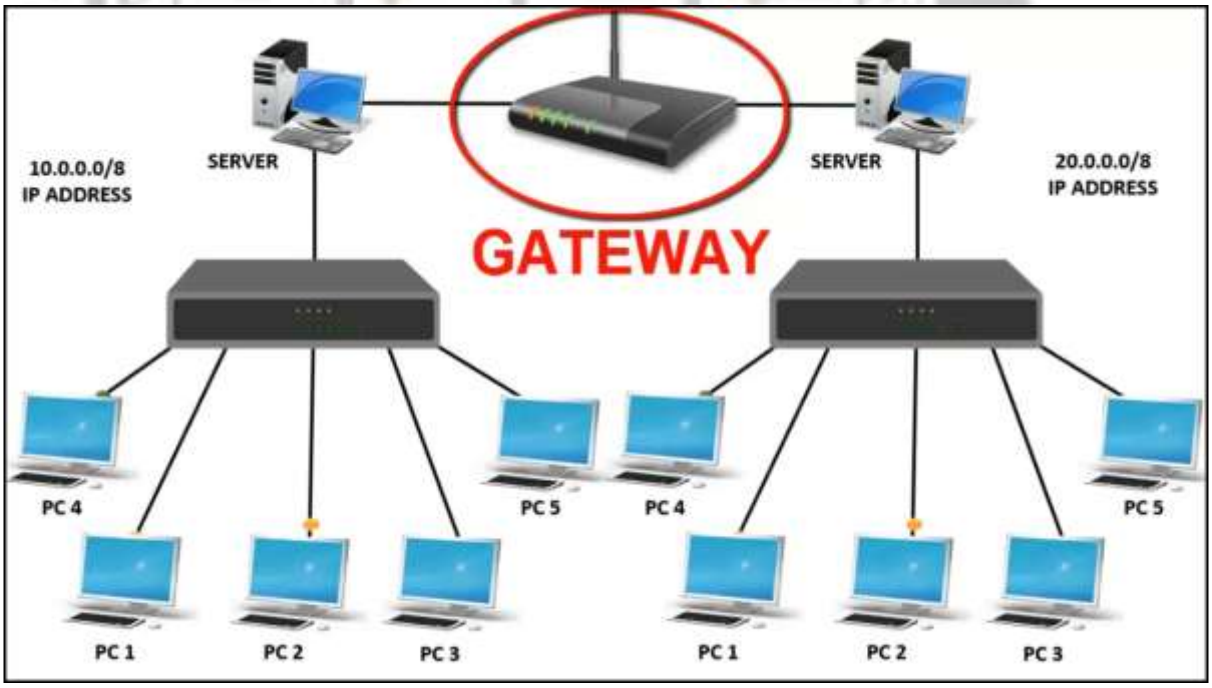
A router is an electronic device that interconnects two or more computer networks and also forwards data packets between them. Routers operate at the network layer (Layer 3) of the OSI model, making decisions based on IP addresses. Routers determine the best path for data packets to travel from the source to the destination, using routing tables and protocols.



6. Gateway:

Gateways in computer networks are devices that serve as a “gate” between two different networks, often with different protocols. They facilitate communication and data transfer between these networks.

Key functions are protocol translation, traffic management, data routing etc., Gateways are used in Home, Enterprise, IoT Networks, Cloud Computing, VPNs etc.



EXPERIMENT – 2

Aim: Write a Program to implement the data link layer farming methods such as

i)Character stuffing ii) bit stuffing.

1.Character Stuffing

Algorithm:

Step 1: Start

Step 2: Append DLE STX at the beginning of the string.

Step 3: Check the data, if the character ‘DLE’ is present in the string (example DOODLE) then insert ‘DLE’ in the string (ex: DOODLEDLE)

Step 4: Transmit DLEETX at the end of the string.

Step 5: Display the string

Step 6: Stop

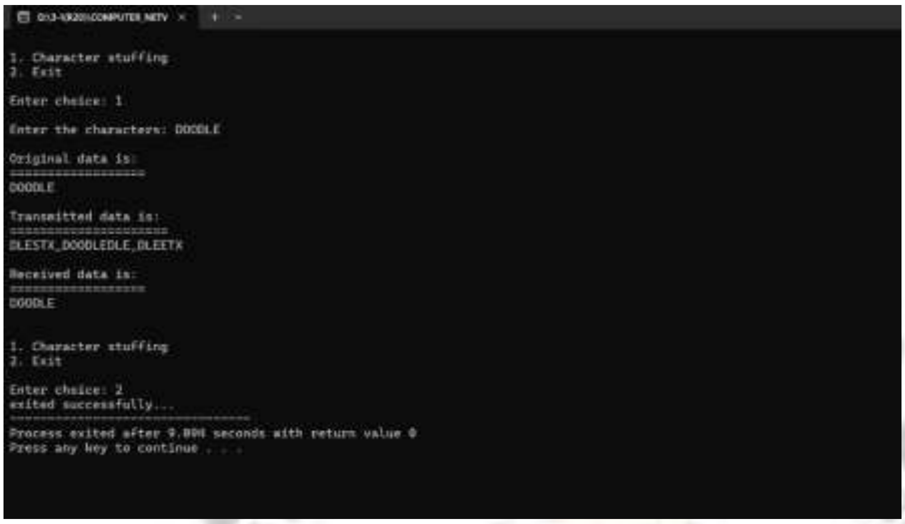
Source Code:

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

void charc(void);

int main()
{
    int choice;
    while(1)
    {
        printf("\n1. Character stuffing");
        printf("\n2. Exit");
        printf("\n\nEnter choice: ");
        scanf("%d", &choice);
        if(choice > 2)
            printf("\nInvalid option....so, please reenter
       (1 or 2): ");
        switch(choice)
        {
            case 1:
                charc();
                break;
            case 2:
              printf("exited successfully...");
                exit(0);
        }
    }
    return 0;
}

void charc(void) {
    char c[50], d[50], t[50];
    int i, m, j;
    printf("\nEnter the characters: ");
    scanf("%s", c);
    m = strlen(c);
    printf("\nOriginal data is: ");
    printf("\n===================\n");
    printf("%s", c);
```

```
for(i = 0, j = 0; i < m; i++, j++)
{
   if(c[i] == 'D' && c[i+1] == 'L' && c[i+2] == 'E')
   {
      d[j] = 'D';
      j++;
      d[j] = 'L';
      j++;
      d[j] = 'E';
      j++;
      m = m + 3;
   }
   d[j] = c[i];
}
d[j] = '\0'; // null terminate the string

   printf("\n\nTransmitted data is: ");
printf("\n========================\n");
printf("DLESTX_");
printf("%s", d);
```

```
    printf("_DLEETX");


    for(i = 0, j = 0; i < m; i++, j++)
    {
        if(d[i] == 'D' && d[i+1] == 'L' && d[i+2] == 'E'
&& d[i+3] == 'D' && d[i+4] == 'L' && d[i+5] == 'E')
            i = i + 3;
        t[j] = d[i];
    }
    t[j] = '\0'; // null terminate the string


    printf("\n\nReceived data is: ");
    printf("\n====================\n");
    for(i = 0; i < j; i++)
    {
        printf("%c", t[i]);
    }
    printf("\n\n");
}
```

Output:



```
1. Character stuffing
2. Exit

Enter choice: 1

Enter the characters: DOODLE

Original data is:
===================
DOODLE

Transmitted data is:
======================
DLESTX_DOODLEDLE_DLEETX

Received data is:
===================
DOODLE


1. Character stuffing
2. Exit

Enter choice: 2
exited successfully...
--------------------------------
Process exited after 9.808 seconds with return value 0
Press any key to continue . . .
```

2. Bit Stuffing

Algorithm:

Step 1: Start

Step 2: Initialize the array for transmitted stream with the special bit pattern 0111 1110 which indicates the beginning of the frame.

Step 3: Get the bit stream to be transmitted in to the array.

Step 4: Check for five consecutive ones and if they occur, stuff a bit ‘0’.

Step 5: Display the data transmitted as it appears on the data line after appending 0111 1110 at the end.

Step 6: For de−stuffing, copy the transmitted data to another array after detecting the stuffed bits.

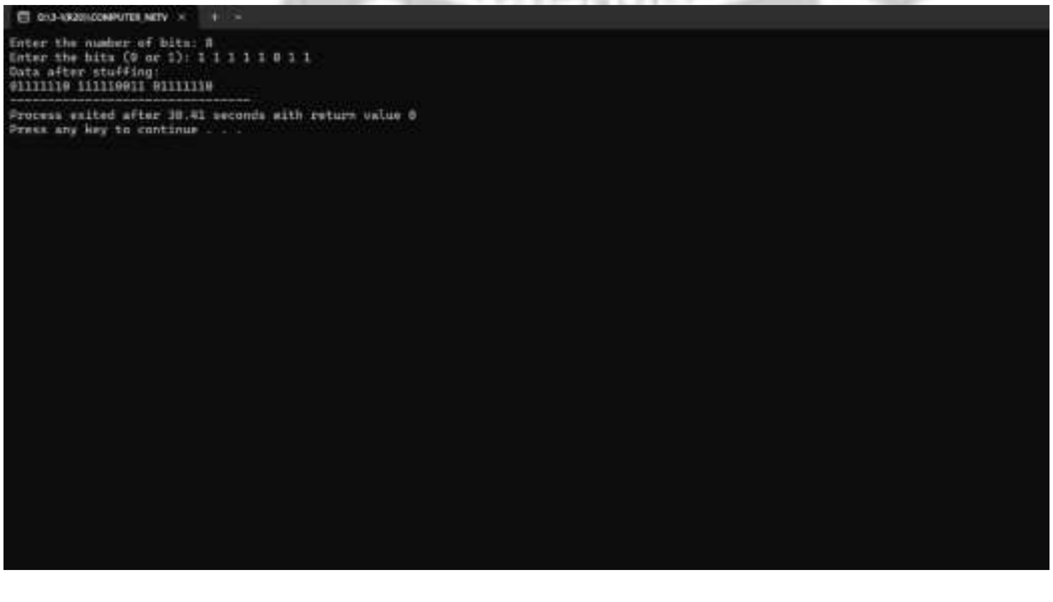
Step 7: Display the received bit stream.

Step 8: Stop

Source Code:

```
#include <stdio.h>

int main() {
    int N;
    int arr[30]; // Array to hold the input bits
    int brr[60]; // Array to hold the stuffed bits
    int i, j = 0, count;

    // Take number of bits from user
    printf("Enter the number of bits: ");
    scanf("%d", &N);

    // Take the input bits from user
    printf("Enter the bits (0 or 1): ");
    for (i = 0; i < N; i++) {
        scanf("%d", &arr[i]);
    }

    // Loop to perform bit stuffing
    for (i = 0; i < N; i++) {
        // If the current bit is a set bit
        if (arr[i] == 1) {
            count = 1; // Count of consecutive 1s
            brr[j++] = arr[i]; // Store the current bit

            // Loop to check for next bits
            while (i + 1 < N && arr[i + 1] == 1 && count <
5) {
                i++; // Move to the next bit
                brr[j++] = arr[i]; // Store the bit
                count++;

                // If 5 consecutive set bits are found, insert
                a 0 bit
                if (count == 5) {
                    brr[j++] = 0; // Stuff a 0
                }
            }
        } else {
            brr[j++] = arr[i]; // Otherwise, just store the bit
        }
    }
    // Print the stuffed bits
    printf("Data after stuffing: \n");
    printf("01111110 ");
    for (i = 0; i < j; i++) {
        printf("%d", brr[i]);
    }

       printf(" 01111110");
    return 0;
}
```

Output:



```
Enter the number of bits: 8
Enter the bits (0 or 1): 1 1 1 1 1 0 1 1
Data after stuffing:
01111110 111110011 01111110
--------------------------------
Process exited after 30.41 seconds with return value 0
Press any key to continue . . .
```

EXPERIMENT – 3

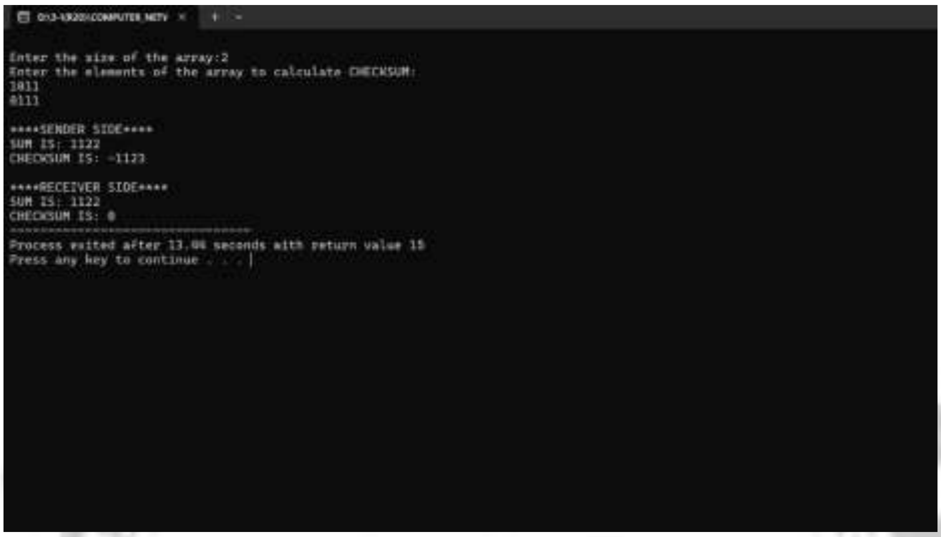
Aim: Write a Program to implement data link layer framing method checksum.

Source Code:

```
#include<stdio.h>
#include<math.h>
int sender(int arr[10],int n) {
        int checksum,sum=0,i;
        printf("\n****SENDER SIDE****\n");
        for(i=0;i<n;i++)
                sum+=arr[i];
        printf("SUM IS: %d",sum);
        checksum=~sum;    //1's complement of sum
        printf("\nCHECKSUM IS: %d",checksum);
        return checksum;
}
void receiver(int arr[10],int n,int sch) {
        int checksum,sum=0,i;
        printf("\n\n****RECEIVER SIDE****\n");
        for(i=0;i<n;i++)
```

```
		sum+=arr[i];
	printf("SUM IS: %d",sum);
	sum=sum+sch;
	checksum=~sum;   //1's complement of sum
	printf("\nCHECKSUM IS: %d",checksum);
}
void main() {
	int n, sch, rch, i;
	printf("\nEnter the size of the array:");
	scanf("%d",&n);
	int arr[n];
	printf("Enter the elements of the array to
calculate CHECKSUM: \n");
	for(i=0; i<n; i++) {
		scanf("%d",&arr[i]);
	}
	sch=sender(arr,n);
	receiver(arr,n,sch);


}
```

Output:



```
Enter the size of the array:2
Enter the elements of the array to calculate CHECKSUM:
1011
0111

****SENDER SIDE****
SUM IS: 1122
CHECKSUM IS: -1123

****RECEIVER SIDE****
SUM IS: 1122
CHECKSUM IS: 0
--------------------------------
Process exited after 13.06 seconds with return value 15
Press any key to continue . . . |
```

EXPERIMENT – 4

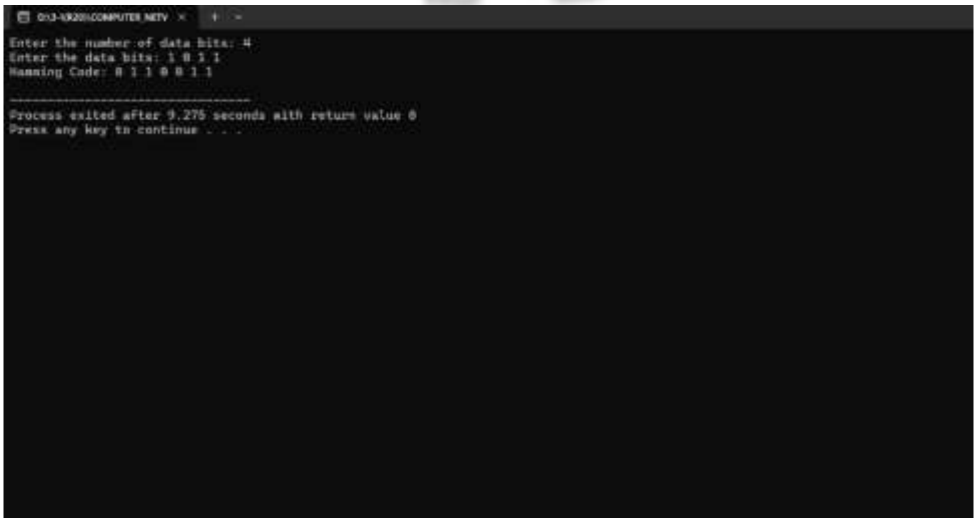
Aim: Write a program for Hamming Code generation for error detection and correction.

Source Code:

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

// Function to calculate the number of parity bits needed
int calculateParityBits(int dataBits) {
   int parityBits = 0;
   while ((1 << parityBits) < (dataBits + parityBits
+ 1)) {
      parityBits++;
   }
   return parityBits;
}

// Function to generate Hamming Code
void generateHammingCode(int data[], int
dataBits, int hammingCode[]) {
   int parityBits = calculateParityBits(dataBits);
   int totalBits = dataBits + parityBits;
   int i, j = 0, k = 0;

   // Initialize hamming code array with data
and parity bits
   for (i = 1; i <= totalBits; i++) {
      if ((i & (i - 1)) == 0) {
          hammingCode[i - 1] = 0; // Place parity bits
at positions 1, 2, 4, 8, ...
      } else {
          hammingCode[i - 1] = data[j++];
      }
   }

   // Calculate parity bits
   for (i = 0; i < parityBits; i++) {
      int parityPos = (1 << i);
      int parity = 0;
      for (j = parityPos; j <= totalBits; j +=
(parityPos << 1)) {
```

```
            for (k = j; k < j + parityPos && k <=
totalBits; k++) {
                parity ^= hammingCode[k - 1];
            }
        }
        hammingCode[parityPos - 1] = parity;
    }
}

int main() {
    int dataBits, i;
    printf("Enter the number of data bits: ");
    scanf("%d", &dataBits);

    int data[dataBits];
    printf("Enter the data bits: ");
    for (i = 0; i < dataBits; i++) {
        scanf("%d", &data[i]);
    }

    int parityBits = calculateParityBits(dataBits);
    int totalBits = dataBits + parityBits;
    int hammingCode[totalBits];

    generateHammingCode(data, dataBits,
hammingCode);

    printf("Hamming Code: ");
    for (i = 0; i < totalBits; i++) {
        printf("%d ", hammingCode[i]);
    }
    printf("\n");

    return 0;
}
```

Output:



```
Enter the number of data bits: 4
Enter the data bits: 1 0 1 1
Hamming Code: 0 1 1 0 0 1 1

--------------------------------
Process exited after 9.275 seconds with return value 0
Press any key to continue . . .
```

EXPERIMENT – 5

Aim: Write a Program to implement on a data set characters the three CRC polynomials – CRC 12, CRC 16 and CRC CCIP.

Source Code:

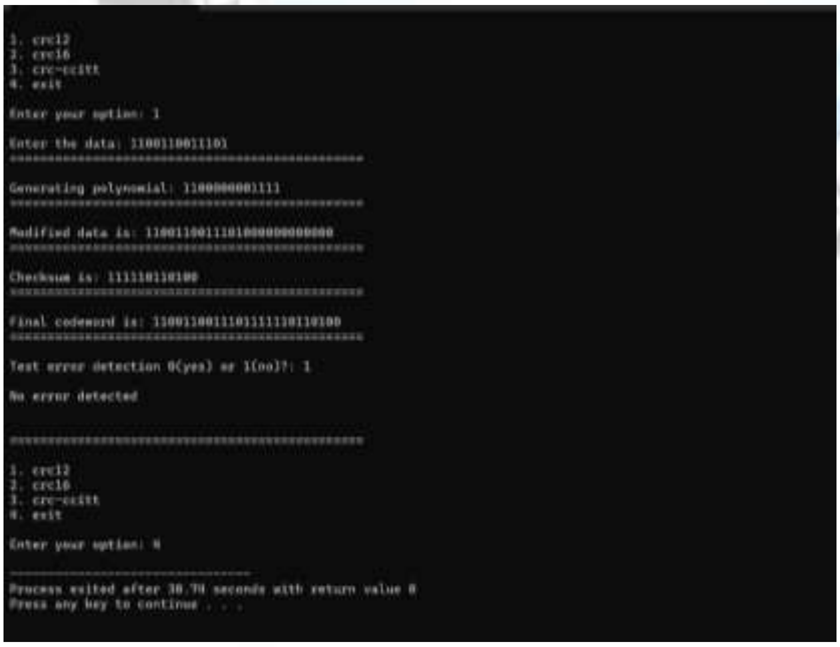
```
#include <stdio.h>
#include <string.h>

#define N strlen(g)

char t[28], cs[28], g[28];
int a, e, c, b;

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

int main() {
    int flag = 0;
    do {
        printf("\n1. crc12\n2. crc16\n3. crc-ccitt\n4.
exit\n\nEnter your option: ");
        scanf("%d", &b);
        switch (b) {
            case 1: strcpy(g, "1100000001111"); break;
            case 2: strcpy(g, "11000000000000101"); break;
            case 3: strcpy(g, "10001000000100001"); break;
            case 4: return 0;
        }
```

```
 printf("\nEnter the data: ");
 scanf("%s", t);

printf("===================================
==================\n");
 printf("\nGenerating polynomial: %s", g);
 a = strlen(t);
 for (e = a; e < a + N - 1; e++)
     t[e] = '0';
 t[e] = '\0';

printf("\n==================================
====================\n");
 printf("\nModified data is: %s", t);

printf("\n==================================
====================\n");
 crc();
 printf("\nChecksum is: %s", cs);
 for (e = a; e < a + N - 1; e++)
     t[e] = cs[e - a];
 t[e] = '\0';
```

```
    printf("\n=====================================
=====================\n");
     printf("\nFinal codeword is: %s", t);

    printf("\n=====================================
=====================\n");
     printf("\nTest error detection 0(yes) or 1(no)?: ");
     scanf("%d", &e);
     if (e == 0) {
         do {
             printf("\nEnter the position where error is to
be inserted: ");
             scanf("%d", &e);
         } while (e == 0 || e > a + N - 1);
         t[e - 1] = (t[e - 1] == '0') ? '1' : '0';

    printf("\n=====================================
=====================\n");
         printf("\nErroneous data: %s\n", t);
     }
     crc();
     for (e = 0; (e < N - 1) && (cs[e] != '1'); e++);
```

```
        if (e < N - 1)
            printf("\nError detected");
        else
            printf("\nNo error detected\n\n");

        printf("\n===========================================================\n");
    } while (flag != 1);
    return 0;
}
```

Output:



```
1. crc12
2. crc16
3. crc-ccitt
4. exit

Enter your option: 1

Enter the data: 1100110011101
==================================================

Generating polynomial: 1100000001111
==================================================

Modified data is: 1100110011101000000000000
==================================================

Checksum is: 111110110100
==================================================

Final codeword is: 1100110011101111110110100
==================================================

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

No error detected


==================================================

1. crc12
2. crc16
3. crc-ccitt
4. exit

Enter your option: 4

--------------------------------
Process exited after 38.74 seconds with return value 0
Press any key to continue . . .
```

EXPERIMENT – 6

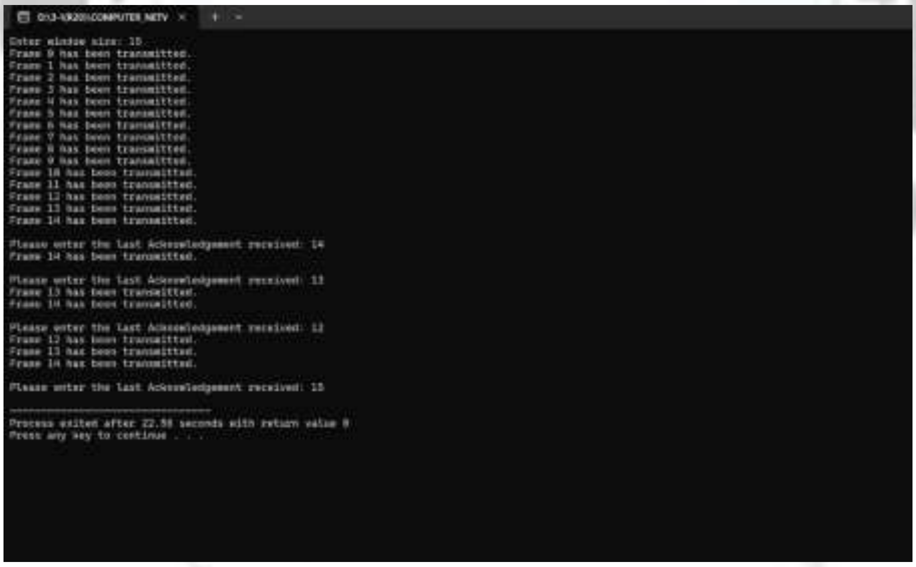
Aim: Write a Program to implement Sliding window protocol for Goback N.

Source Code:

```
#include <stdio.h>

int main() {
    int windowsize, sent = 0, ack, i;
    printf("Enter window size: ");
    scanf("%d", &windowsize);

    while (1) {
        for (i = 0; i < windowsize; i++) {
            printf("Frame %d has been transmitted.\n",
sent);
            sent++;
            if (sent == windowsize)
                break;
        }
        printf("\nPlease enter the last
Acknowledgement received: ");
```

```
            scanf("%d", &ack);
            if (ack == windowsize)
                break;
            else
                sent = ack;
        }
        return 0;
    }
```

Output:



```
Enter window size: 15
Frame 0 has been transmitted.
Frame 1 has been transmitted.
Frame 2 has been transmitted.
Frame 3 has been transmitted.
Frame 4 has been transmitted.
Frame 5 has been transmitted.
Frame 6 has been transmitted.
Frame 7 has been transmitted.
Frame 8 has been transmitted.
Frame 9 has been transmitted.
Frame 10 has been transmitted.
Frame 11 has been transmitted.
Frame 12 has been transmitted.
Frame 13 has been transmitted.
Frame 14 has been transmitted.

Please enter the last Acknowledgement received: 14
Frame 14 has been transmitted.

Please enter the last Acknowledgement received: 13
Frame 13 has been transmitted.
Frame 14 has been transmitted.

Please enter the last Acknowledgement received: 12
Frame 12 has been transmitted.
Frame 13 has been transmitted.
Frame 14 has been transmitted.

Please enter the last Acknowledgement received: 15

--------------------------------
Process exited after 22.58 seconds with return value 0
Press any key to continue . . .
```

EXPERIMENT – 7

Aim: Write a Program to implement Sliding window protocol for Selective repeat.

Algorithm:

Step 1: Start.

Step 2: Get the frame size from the user.

Step 3: Create the frames based on the user request.

Step 4: Send frames to the server from the client side.

Step 5: If your frames reach the server then it will send ACK signal to client otherwise it'll send NACK signal to client.

Step 6: Stop

Source Code:

<u>server.c</u>

```
#include <stdio.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <string.h>

#include <stdlib.h>
```

```
#include <arpa/inet.h>

#define SIZE 4

int main() {
    int sfd, lfd, len, i, j, status;
    char str[20], frame[20], temp[20], ack[20];
    struct sockaddr_in saddr, caddr;

    sfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sfd < 0) {
        perror("Socket Error");
        exit(1);
    }

    bzero(&saddr, sizeof(saddr));
    saddr.sin_family = AF_INET;
    saddr.sin_addr.s_addr = htonl(INADDR_ANY);
    saddr.sin_port = htons(5465);
    if (bind(sfd, (struct sockaddr*)&saddr,
    sizeof(saddr)) < 0) {
        perror("Bind Error");
        exit(1);
    }

    listen(sfd, 5);
    len = sizeof(caddr);
    lfd = accept(sfd, (struct sockaddr*)&caddr,
&len);
    if (lfd < 0) {
        perror("Accept Error");
        exit(1);
    }

    printf("Enter the text :");
    scanf("%s", str);

    i = 0;
    while (i < strlen(str)) {
        memset(frame, 0, 20);
        strncpy(frame, str + i, SIZE);
```

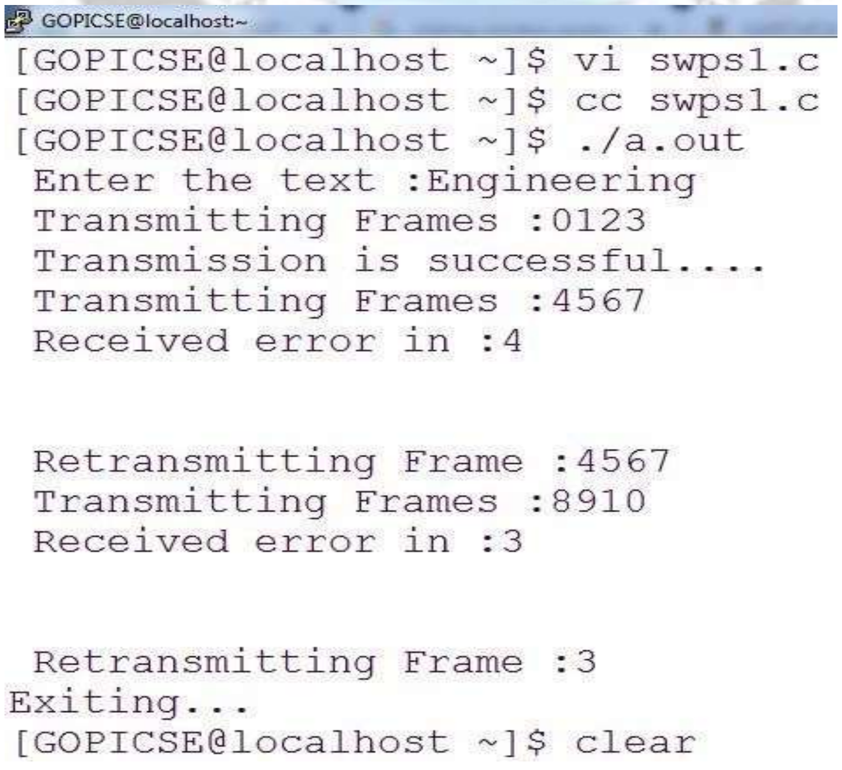
```
printf("Transmitting Frames :");
len = strlen(frame);
for (j = 0; j < len; j++) {
    printf("%d", i + j);
    sprintf(temp, "%d", i + j);
    strcat(frame, temp);
}
printf("\n");
write(lfd, frame, sizeof(frame));
read(lfd, ack, 20);
sscanf(ack, "%d", &status);
if (status == -1) {
    printf("Transmission is successful….\n");
} else {
    printf("Received error in :%d\n\n", status);
    printf("Retransmitting Frame :");
    for (j = 0;;) {
        frame[j] = str[j + status];
        printf("%d", j + status);
        j++;
        if ((j + status) % SIZE == 0)
```

```
                    break;
                }
                printf("\n");
                frame[j] = '\0';
                len = strlen(frame);
                for (j = 0; j < len; j++) {
                    sprintf(temp, "%d", j + status);
                    strcat(frame, temp);
                }
                write(lfd, frame, sizeof(frame));
            }
            i += SIZE;
        }
        write(lfd, "exit", sizeof("exit"));
        printf("Exiting...\n");
        sleep(2);
        close(lfd);
        close(sfd);
        return 0;
    }
```

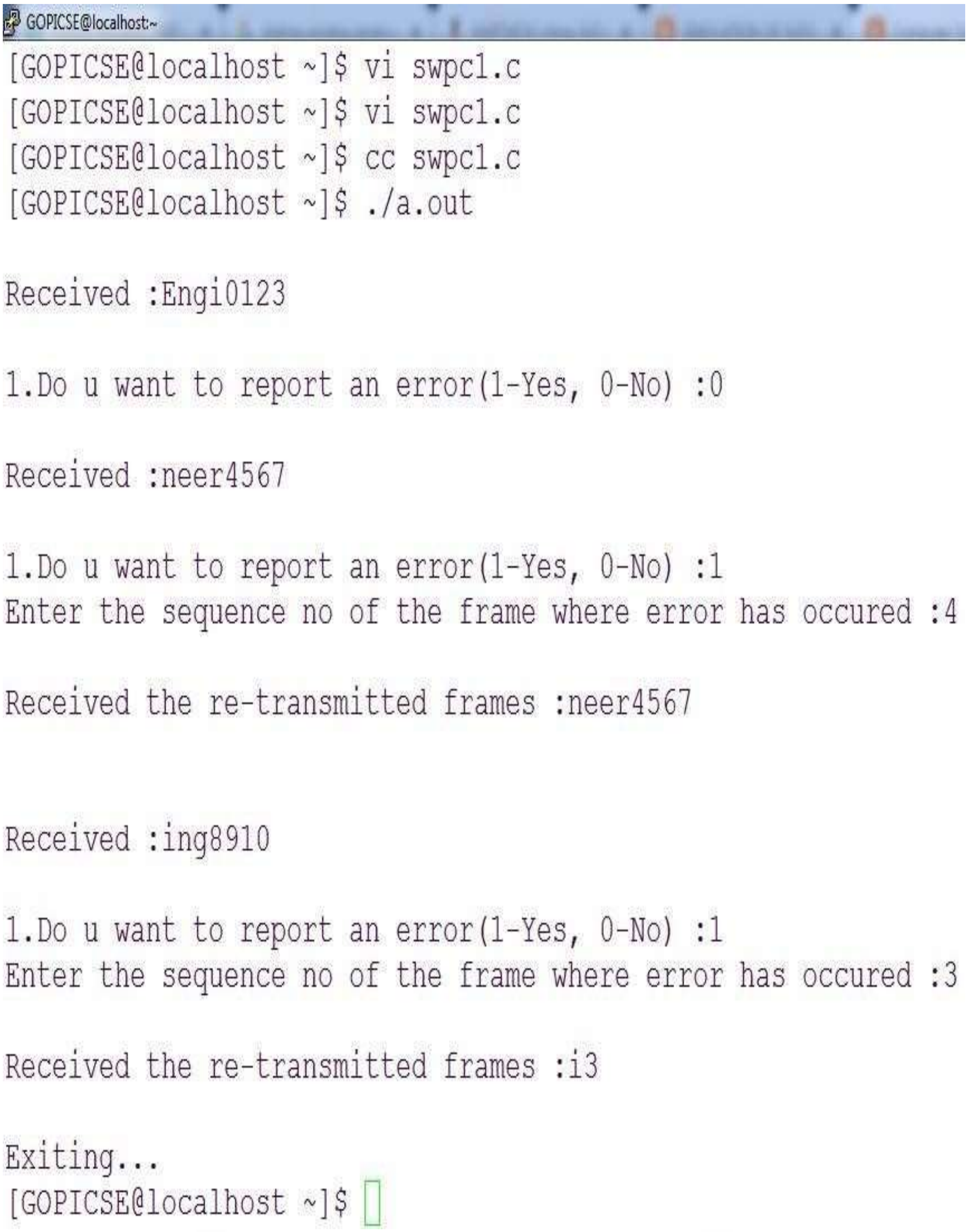
client.c

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

int main() {
    int sfd, choice;
    char str[20], err[20];
    struct sockaddr_in saddr;

    sfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sfd < 0) {
        perror("Socket Error");
        exit(1);
    }

    bzero(&saddr, sizeof(saddr));
    saddr.sin_family = AF_INET;
    saddr.sin_addr.s_addr = INADDR_ANY;
    saddr.sin_port = htons(5465);

if (connect(sfd, (struct sockaddr*)&saddr,
sizeof(saddr)) < 0) {
        perror("Connect Error");
        exit(1);
    }

    for (;;) {
        read(sfd, str, 20);
        if (!strcmp(str, "exit")) {
            printf("Exiting...\n");
            break;
        }
        printf("\nReceived: %s\n\n1.Do you want to
report an error (1-Yes, 0-No) :", str);
        scanf("%d", &choice);
        if (!choice) {
            write(sfd, "-1", sizeof("-1"));
        } else {
            printf("Enter the sequence no of the frame where error has occurred :");
            scanf("%s", err);
            write(sfd, err, sizeof(err));
            read(sfd, str, 20);
            printf("\nReceived the re-transmitted frames :%s\n\n", str);
        }
    }
    close(sfd);
    return 0;
}
```

Output:



```
GOPICSE@localhost:~
[GOPICSE@localhost ~]$ vi swps1.c
[GOPICSE@localhost ~]$ cc swps1.c
[GOPICSE@localhost ~]$ ./a.out
 Enter the text :Engineering
 Transmitting Frames :0123
 Transmission is successful....
 Transmitting Frames :4567
 Received error in :4


 Retransmitting Frame :4567
 Transmitting Frames :8910
 Received error in :3


 Retransmitting Frame :3
Exiting...
[GOPICSE@localhost ~]$ clear
```


GOPICSE@localhost:~

```
[GOPICSE@localhost ~]$ vi swpc1.c
[GOPICSE@localhost ~]$ vi swpc1.c
[GOPICSE@localhost ~]$ cc swpc1.c
[GOPICSE@localhost ~]$ ./a.out

Received :Engi0123

1.Do u want to report an error(1-Yes, 0-No) :0

Received :neer4567

1.Do u want to report an error(1-Yes, 0-No) :1
Enter the sequence no of the frame where error has occured :4

Received the re-transmitted frames :neer4567


Received :ing8910

1.Do u want to report an error(1-Yes, 0-No) :1
Enter the sequence no of the frame where error has occured :3

Received the re-transmitted frames :i3

Exiting...
[GOPICSE@localhost ~]$
```

EXPERIMENT – 8

Aim: Write a Program to implement Stop and Wait Protocol.

Source Code:

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

#define TIMEOUT 5
#define MAX_SEQ 1
#define TOT_PACKETS 8
#define inc(k) if(k<MAX_SEQ) k++; else k=0;

typedef struct {
    int data;
} packet;

typedef struct {
    int kind;
```

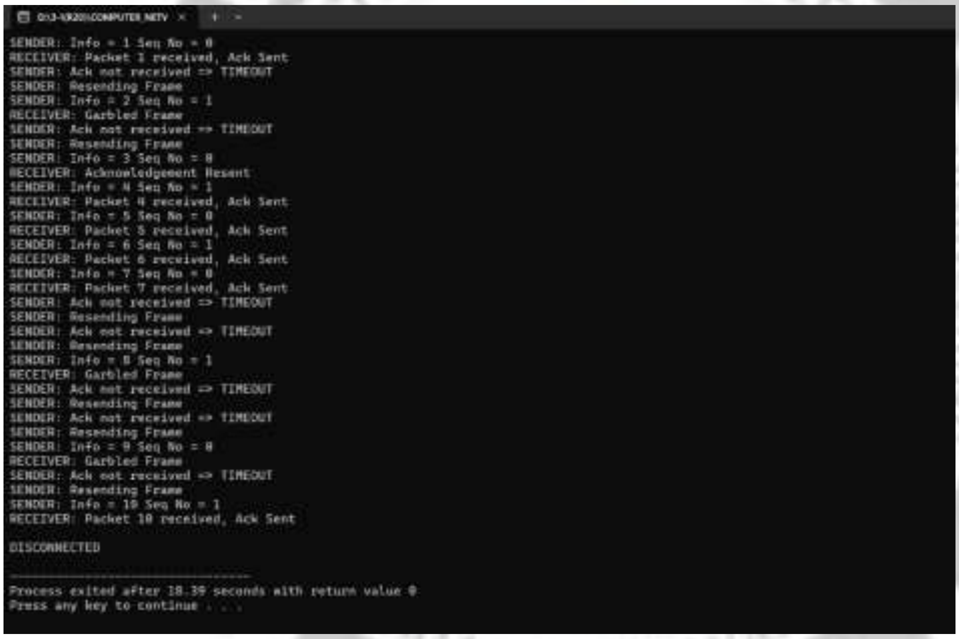
```
        int seq;
        int ack;
        packet info;
        int err;
    } frame;

    frame DATA;
    typedef enum { frame_arrival, err, timeout,
no_event } event_type;

    void from_network_layer(packet *);
    void to_network_layer(packet *);
    void to_physical_layer(frame *);
    void from_physical_layer(frame *);
    void wait_for_event_sender(event_type *);
    void wait_for_event_reciever(event_type *);
    void receiver();
    void sender();

    int i = 1;
    char turn;
```

```
int DISCONNECT = 0;

void sender() {
    static int frame_to_send = 0;
    static frame s;
    packet buffer;
    event_type event;
    static int flag = 0;

    if (flag == 0) {
        from_network_layer(&buffer);
        s.info = buffer;
        s.seq = frame_to_send;
        printf("SENDER: Info = %d Seq No = %d\n",
s.info.data, s.seq);
        turn = 'r';
        to_physical_layer(&s);
        flag = 1;
    }

    wait_for_event_sender(&event);
    if (turn == 's') {
        if (event == frame_arrival) {
            from_network_layer(&buffer);
            inc(frame_to_send);
            s.info = buffer;
            s.seq = frame_to_send;
            printf("SENDER: Info = %d Seq No =
%d\n", s.info.data, s.seq);
            turn = 'r';
            to_physical_layer(&s);
        }
        if (event == timeout) {
            printf("SENDER: Resending Frame\n");
            to_physical_layer(&s);
        }
    }
}

void receiver() {
    static int frame_expected = 0;
    frame r, s;
    event_type event;

    wait_for_event_reciever(&event);

    if (turn == 'r') {
        if (event == frame_arrival) {
            from_physical_layer(&r);
            if (r.seq == frame_expected) {
                to_network_layer(&r.info);
                inc(frame_expected);
            } else {
                printf("RECEIVER: Acknowledgement
Resent\n");
            }
            turn = 's';
            to_physical_layer(&s);
        }
        if (event == err) {
            printf("RECEIVER: Garbled Frame\n");
            turn = 's';
        }
    }
}

void from_network_layer(packet *buffer) {
    buffer->data = i;
    i++;
}

void to_physical_layer(frame *s) {
    s->err = rand() % 4;  // non-zero means no error
    DATA = *s;
}

void to_network_layer(packet *buffer) {
printf("RECEIVER: Packet %d received, Ack
Sent\n", buffer->data);
    if (i > TOT_PACKETS) {
        DISCONNECT = 1;
        printf("\nDISCONNECTED\n");
    }
```

```
}

void from_physical_layer(frame *buffer) {
    *buffer = DATA;
}

void wait_for_event_sender(event_type *e) {
    static int timer = 0;
    if (turn == 's') {
        timer++;
        if (timer == TIMEOUT) {
            *e = timeout;
            printf("SENDER: Ack not received =>
TIMEOUT\n");
            timer = 0;
            return;
        }
        if (DATA.err == 0) {
            *e = err;
        } else {
            timer = 0;
            *e = frame_arrival;
        }
    }
}

void wait_for_event_reciever(event_type *e) {
    if (turn == 'r') {
        if (DATA.err == 0) {
            *e = err;
        } else {
            *e = frame_arrival;
        }
    }
}

int main() {
    srand(time(NULL));  // Seed the random
number generator
    while (!DISCONNECT) {
        sender();
        usleep(400000);  // 400 ms delay
```

```
            receiver();
        }
        return 0;
    }
```

Output:



```
SENDER: Info = 1 Seq No = 0
RECEIVER: Packet 1 received, Ack Sent
SENDER: Ack not received => TIMEOUT
SENDER: Resending Frame
SENDER: Info = 2 Seq No = 1
RECEIVER: Garbled Frame
SENDER: Ack not received => TIMEOUT
SENDER: Resending Frame
SENDER: Info = 3 Seq No = 0
RECEIVER: Acknowledgement Resent
SENDER: Info = 4 Seq No = 1
RECEIVER: Packet 4 received, Ack Sent
SENDER: Info = 5 Seq No = 0
RECEIVER: Packet 5 received, Ack Sent
SENDER: Info = 6 Seq No = 1
RECEIVER: Packet 6 received, Ack Sent
SENDER: Info = 7 Seq No = 0
RECEIVER: Packet 7 received, Ack Sent
SENDER: Ack not received => TIMEOUT
SENDER: Resending Frame
SENDER: Ack not received => TIMEOUT
SENDER: Resending Frame
SENDER: Info = 8 Seq No = 1
RECEIVER: Garbled Frame
SENDER: Ack not received => TIMEOUT
SENDER: Resending Frame
SENDER: Ack not received => TIMEOUT
SENDER: Resending Frame
SENDER: Info = 9 Seq No = 0
RECEIVER: Garbled Frame
SENDER: Ack not received => TIMEOUT
SENDER: Resending Frame
SENDER: Info = 10 Seq No = 1
RECEIVER: Packet 10 received, Ack Sent

DISCONNECTED

--------------------------------
Process exited after 18.39 seconds with return value 0
Press any key to continue . . .
```

EXPERIMENT – 9

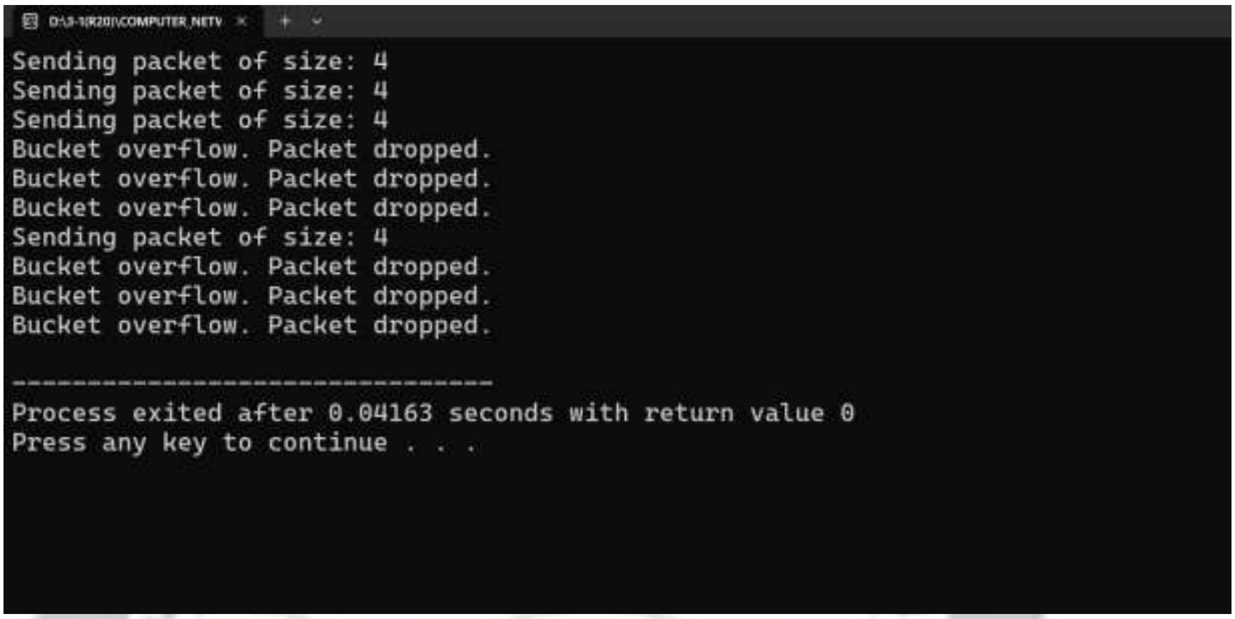
Aim: Write a program for congestion control using leaky bucket algorithm.

Source Code:

```
#include <stdio.h>
#include <stdbool.h>
#include <unistd.h>

#define BUCKET_SIZE 10
#define PACKET_SIZE 2
#define OUTPUT_RATE 1

typedef struct {
    int size;
} Packet;

void sendPacket(Packet packet) {
    printf("Sending packet of size: %d\n",
packet.size);
}
```

```
void leakyBucketAlgorithm(int arrivalRate, int
numPackets) {
    int bucketLevel = 0, i;
    int outgoingRate = OUTPUT_RATE;

    for (i = 0; i < numPackets; i++) {
        Packet packet;
        packet.size = arrivalRate;

        if (bucketLevel + packet.size <=
       BUCKET_SIZE) {
            bucketLevel += packet.size;
            sendPacket(packet);
        } else {
            printf("Bucket overflow. Packet
dropped.\n");
        }

        if (bucketLevel >= outgoingRate) {
            bucketLevel -= outgoingRate;
        } else {
            printf("Bucket underflow. Waiting...\n");
            usleep(1000000 / outgoingRate);
            bucketLevel = 0;
        }

        usleep(1000000); // Simulate time passing
for each iteration
    }
}

int main() {
    int arrivalRate = 4;
    int numPackets = 10; // Number of packets to
process
    leakyBucketAlgorithm(arrivalRate,
numPackets);
    return 0;
}
```

Output:



```
Sending packet of size: 4
Sending packet of size: 4
Sending packet of size: 4
Bucket overflow. Packet dropped.
Bucket overflow. Packet dropped.
Bucket overflow. Packet dropped.
Sending packet of size: 4
Bucket overflow. Packet dropped.
Bucket overflow. Packet dropped.
Bucket overflow. Packet dropped.

--------------------------------
Process exited after 0.04163 seconds with return value 0
Press any key to continue . . .
```

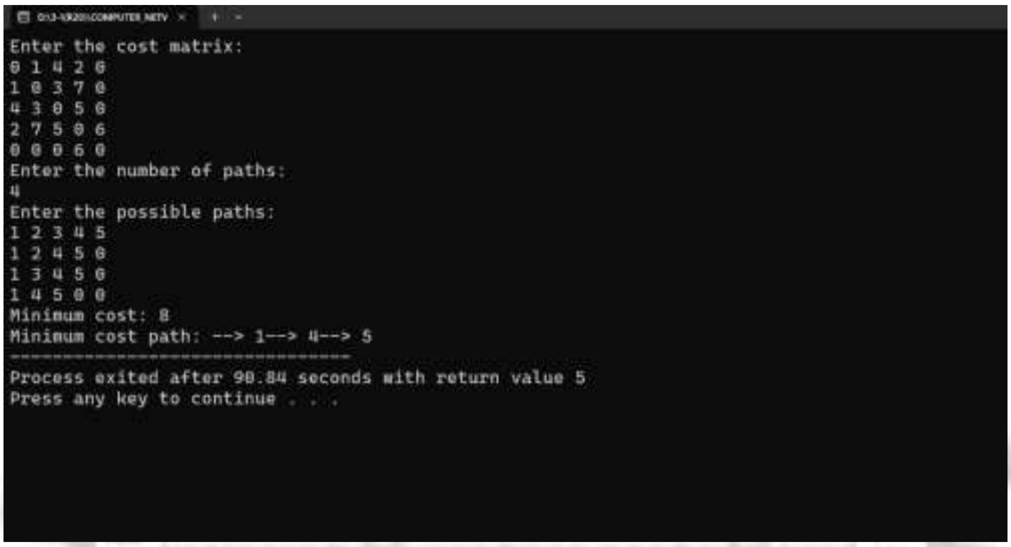
EXPERIMENT – 10

Aim: Write a Program to implement Dijkstra‘s algorithm to compute the Shortest path through a graph.

Source Code:

```
#include <stdio.h>

void main() {
    int path[5][5], i, j, min, a[5][5], p, st = 1, ed = 5,
stp, edp, t[5], index;

    printf("Enter the cost matrix:\n");
    for (i = 0; i < 5; i++) {
        for (j = 0; j < 5; j++) {
            scanf("%d", &a[i][j]);
        }
    }
    printf("Enter the number of paths:\n");
    scanf("%d", &p);

    printf("Enter the possible paths:\n");
    for (i = 0; i < p; i++) {
        for (j = 0; j < 5; j++) {
            scanf("%d", &path[i][j]);
        }
    }

    for (i = 0; i < p; i++) {
        t[i] = 0;
        stp = st - 1;  // Adjusting for 0-based
indexing
        for (j = 0; j < 4; j++) { // Only iterate
through 4 elements to avoid out-of-bounds
            edp = path[i][j + 1] - 1;  // Adjusting for 0-
based indexing
            t[i] = t[i] + a[stp][edp];
            if (edp == ed - 1) break;  // Adjusting for 0-
based indexing
            else stp = edp;
        }
    }

    min = t[0];
    index = 0;
    for (i = 1; i < p; i++) {
        if (min > t[i]) {
            min = t[i];
            index = i;
        }
    }

    printf("Minimum cost: %d\n", min);
    printf("Minimum cost path: ");
    for (i = 0; i < 5; i++) {
        printf("--> %d", path[index][i]);
        if (path[index][i] == ed) break;
    }

}
```

Output:



```
Enter the cost matrix:
0 1 4 2 0
1 0 3 7 0
4 3 0 5 0
2 7 5 0 6
0 0 0 6 0
Enter the number of paths:
4
Enter the possible paths:
1 2 3 4 5
1 2 4 5 0
1 3 4 5 0
1 4 5 0 0
Minimum cost: 8
Minimum cost path: --> 1--> 4--> 5
--------------------------------
Process exited after 90.84 seconds with return value 5
Press any key to continue . . .
```

EXPERIMENT – 11

Aim: Write a Program to implement Distance vector routing algorithm by obtaining routing table at each node (Take an example subnet graph with weights indicating delay between nodes).

Algorithm:

Step 1: Start.

Step 2: Each router prepares its routing table. By their local knowledge each router knows

- All the routers present in the network.
- Distance to its neighbouring routers.

Step 3: Each router exchanges its distance vector with its neighbouring routers.

Step 4: Each router prepares a new routing table using the distance vectors it has obtained from its neighbours.

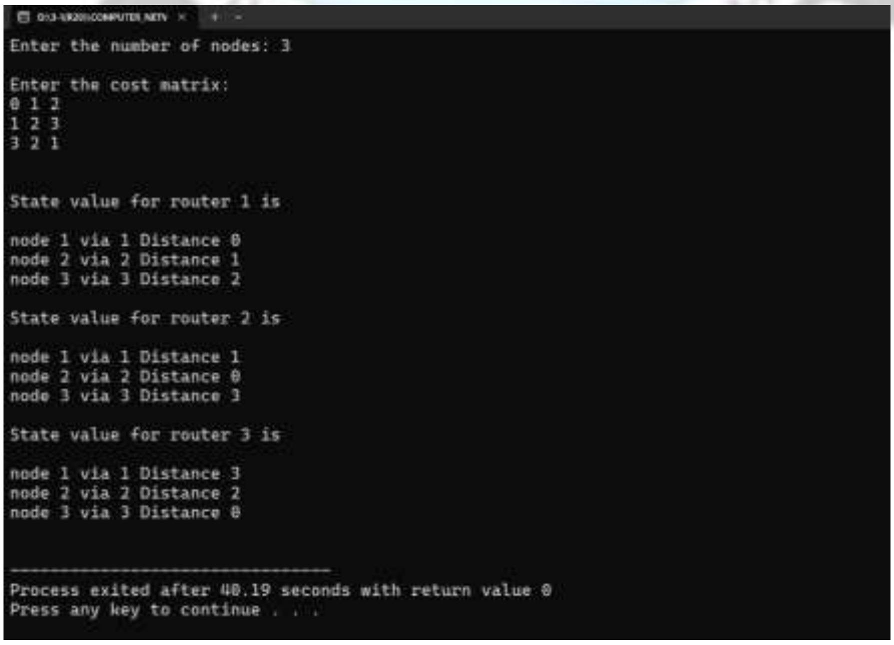
This step is repeated for (n-2) times if there are n routers in the network.

After this, routing tables converge / become stable.

Source Code:

```
#include <stdio.h>

struct node {
    unsigned dist[20];
    unsigned from[20];
} rt[10];

int main() {
    int dmat[20][20];
    int n, i, j, k, count = 0;

    printf("\nEnter the number of nodes: ");
    scanf("%d", &n);

    printf("\nEnter the cost matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &dmat[i][j]);
            dmat[i][i] = 0;
            rt[i].dist[j] = dmat[i][j];
            rt[i].from[j] = j;
        }
    }

    do {
        count = 0;
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                for (k = 0; k < n; k++) {
                    if (rt[i].dist[j] > dmat[i][k] +
rt[k].dist[j]) {
                        rt[i].dist[j] = dmat[i][k] + rt[k].dist[j];
                        rt[i].from[j] = k;
                        count++;
                    }
                }
            }
        }
    } while (count != 0);

    for (i = 0; i < n; i++) {
        printf("\n\nState value for router %d is \n", i + 1);
        for (j = 0; j < n; j++) {
                printf("\t\nnode %d via %d Distance %d", j + 1, rt[i].from[j] + 1, rt[i].dist[j]);
        }
    }

    printf("\n\n");
    return 0;
}
```

Output:



```
Enter the number of nodes: 3

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


State value for router 1 is

node 1 via 1 Distance 0
node 2 via 2 Distance 1
node 3 via 3 Distance 2

State value for router 2 is

node 1 via 1 Distance 1
node 2 via 2 Distance 0
node 3 via 3 Distance 3

State value for router 3 is

node 1 via 1 Distance 3
node 2 via 2 Distance 2
node 3 via 3 Distance 0


--------------------------------
Process exited after 40.19 seconds with return value 0
Press any key to continue . . .
```

EXPERIMENT – 12

Aim: Write a Program to implement Broadcast tree by taking subnet of hosts.

Algorithm:

Step 1: Start

Step 2: Select any edge of minimal value that is not a loop. That is the first edge of T.

Step 3: Select any remaining edge of G having minimal value that does not from a circuit with the edges already included in T.

Step 4: Continue step 3, until T contains n-1 edges where n is the number of vertices of G.

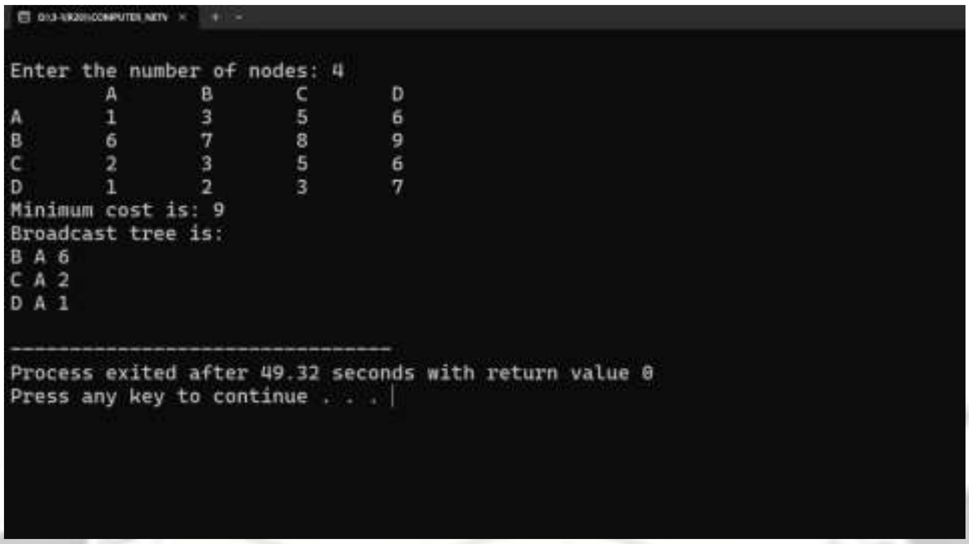
Step 5: Stop

Source Code:

```
#include <stdio.h>

int p, q, u, v, n;

int min = 99, mincost = 0;

int t[50][2], i, j;

int parent[50], edge[50][50];


void sunion(int l, int m);

int find(int l);
int main() {
    printf("\nEnter the number of nodes: ");
    scanf("%d", &n);

    for (i = 0; i < n; i++) {
        printf("\t%c", 65 + i);
        parent[i] = -1;
    }
    printf("\n");

    for (i = 0; i < n; i++) {
        printf("%c", 65 + i);
        for (j = 0; j < n; j++) {
            scanf("%d", &edge[i][j]);
        }
    }

    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (edge[i][j] != 99 && min > edge[i][j]) {
                min = edge[i][j];
                u = i;
                v = j;
            }
        }
        p = find(u);
        q = find(v);,m
        if (p != q) {
            t[i][0] = u;
            t[i][1] = v;
            mincost += edge[u][v];
            sunion(p, q);
        } else {
            t[i][0] = -1;
            t[i][1] = -1;
        }
        min = 99;
    }

printf("Minimum cost is: %d\nBroadcast tree is:\n",
mincost);
```

```
    for (i = 0; i < n; i++) {
        if (t[i][0] != -1 && t[i][1] != -1) {
            printf("%c %c %d\n", 65 + t[i][0], 65 +
t[i][1], edge[t[i][0]][t[i][1]]);
        }
    }
    return 0;
}

void sunion(int l, int m) {
    parent[l] = m;
}
int find(int l) {
    while (parent[l] > 0) {
        l = parent[l];
    }
    return l;
}
```

Output:



```
Enter the number of nodes: 4
        A       B       C       D
A       1       3       5       6
B       6       7       8       9
C       2       3       5       6
D       1       2       3       7
Minimum cost is: 9
Broadcast tree is:
B A 6
C A 2
D A 1

--------------------------------
Process exited after 49.32 seconds with return value 0
Press any key to continue . . .
```

EXPERIMENT – 13

Aim: Understanding about Wireshark in detail

Explanation:

Wireshark:

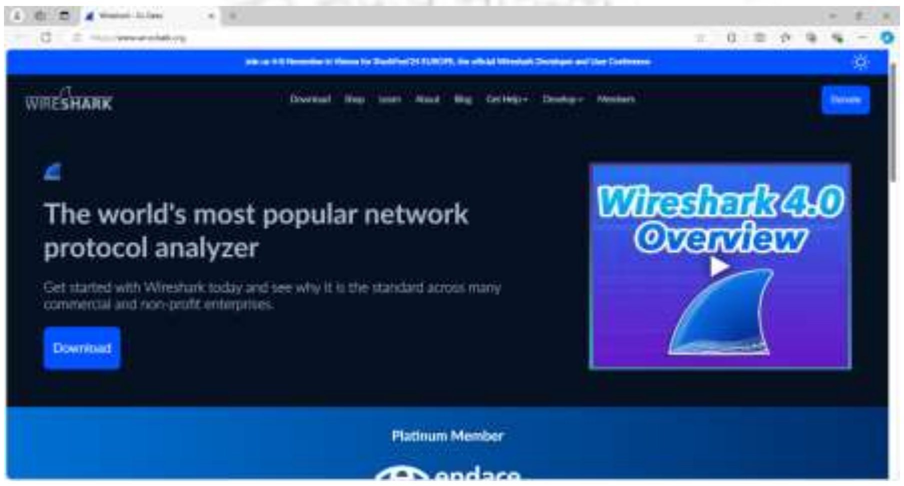
Wireshark is a powerful open-source tool used in computer networks for capturing and analyzing data packets as they travel across a network. It acts as a network protocol analyser , allowing users to observe the communication between devices on a network, whether it's a local area network (LAN) or the internet. By capturing packets, Wireshark provides insights into various protocols, such as TCP/IP, HTTP, and DNS, enabling users to diagnose issues, monitor network performance, and analyse security vulnerabilities.

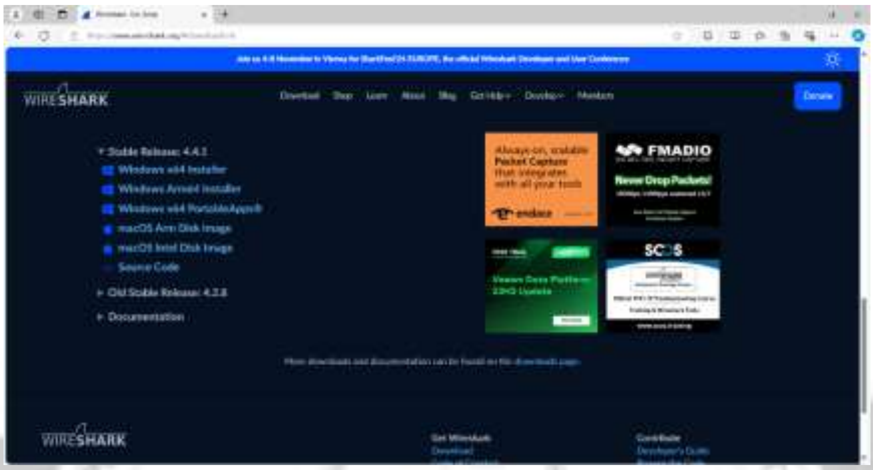
Wireshark is a powerful open-source network protocol analyser widely used in computer networks to capture and analyze data packets as they traverse the network. Being open source, it is freely available for anyone to use, modify, and distribute, which fosters community contributions and continuous improvements. Wireshark is developed collaboratively by a diverse group of volunteers and professionals, making it a robust tool that stays up to date with the latest network protocols and standards.

One of Wireshark's standout features is its cross-platform compatibility. It is available for multiple operating systems, including Windows, macOS, and various distributions of Linux. This flexibility allows users to deploy Wireshark in diverse environments and on different hardware configurations, making it accessible to a wide range of network professionals.

- **Downloading Wireshark**

1. **Visit the Official Website:** Go to the Wireshark website.
2. **Select the Download Option:** Click on the "Download" link, which will direct you to the download page.
3. **Choose Your Operating System:** Select the appropriate version for your operating system (Windows, macOS, or Linux).
4. **Download the Installer:** Click on the installer link to download the Wireshark installation file.


WIRESHARK
The world's most popular network protocol analyzer
Get started with Wireshark today and see why it is the standard across many commercial and non-profit enterprises.
Download
Wireshark 4.0 Overview
Platinum Member
endace



- **Installing Wireshark**

  1. **Run the Installer:** Once the download is complete, locate the installer file (e.g., .exe for Windows, .dmg for macOS) and double-click to run it.
  2. **Follow the Installation Wizard:** Follow the on-screen instructions. For Windows users, you may be prompted to install additional components, like WinPcap or Npcap, which are necessary for packet capturing.
  3. **Choose Components:** You can typically keep the default options selected, but you can customize your installation if needed.
  4. **Finish Installation:** Complete the installation process and launch Wireshark when prompted.

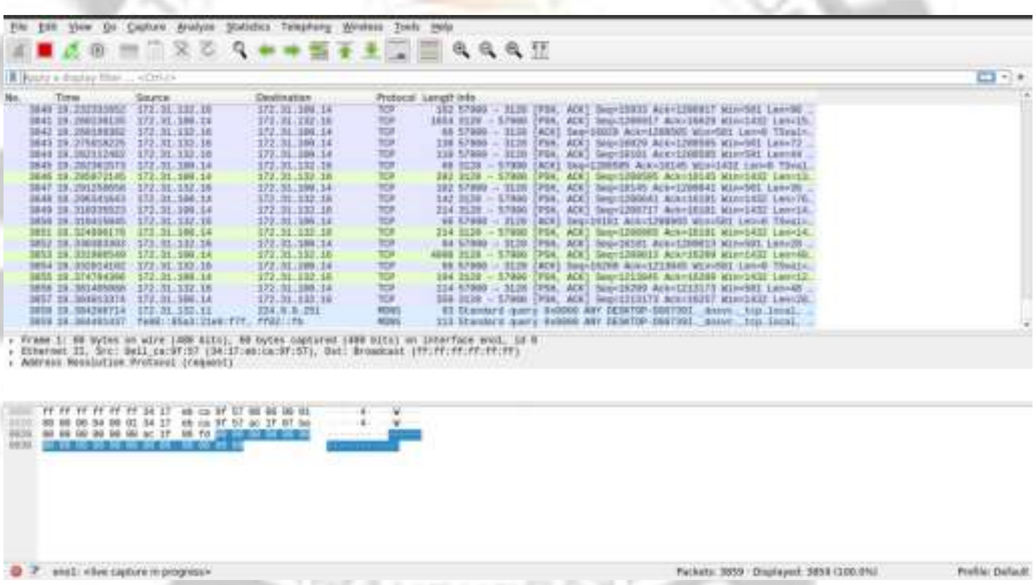
- **Major Components of Wireshark**

1. **Menu Bar**: Located at the top, it contains various menus like File, Edit, View, Capture, Analyse, and Help, providing access to functions such as opening files, starting or stopping captures, and configuring settings.
2. **Toolbar**: Just below the menu bar, it offers quick access to commonly used features like starting/stopping captures, filtering packets, and saving files.
3. **Packet List Pane**: This is the main area where captured packets are displayed in a list. Each row represents a single packet, showing details like the timestamp, source and destination IP addresses, protocol, and length.
4. **Packet Details Pane**: When you select a packet from the list, this pane displays detailed information about that packet. It breaks down the packet's protocol layers and fields, allowing for deep analysis.

5. **Packet Bytes Pane**: Located below the Packet Details Pane, this section shows the raw data of the selected packet in hexadecimal and ASCII format. It's useful for examining the actual byte structure of packets.
6. **Filter Bar**: This is where you can enter display filters to narrow down the packets shown in the Packet List Pane. Wireshark uses a powerful filtering syntax to allow for precise searches.
7. **Status Bar**: At the bottom of the window, it displays information about the current capture, such as the number of packets captured, the number of displayed packets after filtering, and any active capture filters.
8. **Capture Options**: Accessible through the Capture menu, this allows you to configure how packets are captured, including setting filters, choosing interfaces, and adjusting buffer sizes.

- **Running Wireshark**
  1. **Open Wireshark:** Find the Wireshark application in your applications menu or desktop and open it.
  2. **Select a Network Interface:** Upon launch, you'll see a list of available network interfaces. Choose the one from which you want to capture traffic (e.g., Ethernet, Wi-Fi).

3. **Start Capturing Packets:** Click on the selected interface to start capturing packets. You can also click the “Capture” menu and select “Start.”
4. **Monitor the Traffic:** As packets are captured, they will appear in real-time in the main window. You can view packet details in the panes provided.



```
No.   Time          Source          Destination     Protocol Length Info
 3840 19.232333952  172.31.132.16   172.31.100.14   TCP      152 57900 → 3128 [PSH, ACK] Seq=15933 Ack=1208817 Win=501 Len=90 …
 3841 19.260138135  172.31.100.14   172.31.132.16   TCP     1654 3128 → 57900 [PSH, ACK] Seq=1208817 Ack=16029 Win=1432 Len=15…
 3842 19.260188302  172.31.132.16   172.31.100.14   TCP       66 57900 → 3128 [ACK] Seq=16023 Ack=1208505 Win=501 Len=0 TSval=…
 3843 19.275658225  172.31.132.16   172.31.100.14   TCP      138 57900 → 3128 [PSH, ACK] Seq=16029 Ack=1208505 Win=501 Len=72 …
 3844 19.282312482  172.31.132.16   172.31.100.14   TCP      110 57900 → 3128 [PSH, ACK] Seq=16101 Ack=1208585 Win=501 Len=44 …
 3845 19.282382573  172.31.100.14   172.31.132.16   TCP       66 3128 → 57900 [ACK] Seq=1208505 Ack=16145 Win=1432 Len=0 TSval…
 3846 19.285872145  172.31.100.14   172.31.132.16   TCP      182 3128 → 57900 [PSH, ACK] Seq=1208505 Ack=16145 Win=1432 Len=11…
 3847 19.291258056  172.31.132.16   172.31.100.14   TCP      102 57900 → 3128 [PSH, ACK] Seq=16145 Ack=1208641 Win=501 Len=36 …
 3848 19.296541643  172.31.100.14   172.31.132.16   TCP      142 3128 → 57900 [PSH, ACK] Seq=1208641 Ack=16181 Win=1432 Len=76…
 3849 19.310335523  172.31.100.14   172.31.132.16   TCP      214 3128 → 57900 [PSH, ACK] Seq=1208717 Ack=16181 Win=1432 Len=14…
 3850 19.310459845  172.31.132.16   172.31.100.14   TCP       66 57900 → 3128 [ACK] Seq=16181 Ack=1208865 Win=501 Len=0 TSval=…
 3851 19.324896176  172.31.100.14   172.31.132.16   TCP      214 3128 → 57900 [PSH, ACK] Seq=1208865 Ack=16181 Win=1432 Len=14…
 3852 19.330083302  172.31.132.16   172.31.100.14   TCP       84 57900 → 3128 [PSH, ACK] Seq=16181 Ack=1209013 Win=501 Len=28 …
 3853 19.331986549  172.31.100.14   172.31.132.16   TCP     4800 3128 → 57900 [PSH, ACK] Seq=1209013 Ack=16209 Win=1432 Len=46…
 3854 19.332014102  172.31.132.16   172.31.100.14   TCP       66 57900 → 3128 [ACK] Seq=16209 Ack=1213845 Win=501 Len=0 TSval=…
 3855 19.374784386  172.31.100.14   172.31.132.16   TCP      194 3128 → 57900 [PSH, ACK] Seq=1213845 Ack=16209 Win=1432 Len=12…
 3856 19.381485069  172.31.132.16   172.31.100.14   TCP      114 57900 → 3128 [PSH, ACK] Seq=16209 Ack=1213173 Win=501 Len=48 …
 3857 19.384853374  172.31.100.14   172.31.132.16   TCP      350 3128 → 57900 [PSH, ACK] Seq=1213173 Ack=16257 Win=1432 Len=28…
 3858 19.384288714  172.31.132.11   224.0.0.251     MDNS      83 Standard query 0x0000 ANY DESKTOP-S087301._asvc._tcp.local…
 3859 19.384485437  fe80::85a3:21e0:f7f… ff02::fb   MDNS     113 Standard query 0x0000 ANY DESKTOP-S087301._asvc._tcp.local…

▸ Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface eno1, id 0
▸ Ethernet II, Src: Dell_ca:9f:57 (34:17:eb:ca:9f:57), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ Address Resolution Protocol (request)

0000  ff ff ff ff ff ff 34 17  eb ca 9f 57 08 06 00 01   ......4.
0010  08 00 06 04 00 01 34 17  eb ca 9f 57 ac 1f 07 be   ......4.
0020  00 00 00 00 00 00 ac 1f  08 fd 00 00 00 00 00 00
0030  00 00 00 00 00 00 00 00  00 00 00 00

eno1: <live capture in progress>        Packets: 3859 · Displayed: 3859 (100.0%)        Profile: Default
```

- **Viewing Captured Traffic**

1. **Packet List Pane**: As packets are captured, they will appear in the Packet List Pane. Each row represents a packet with columns showing time, source and destination addresses, protocol, length, and info.
2. **Filter Traffic**: You can enter display filters in the Filter Bar at the top to narrow down the visible packets (e.g., http, tcp.port == 80).

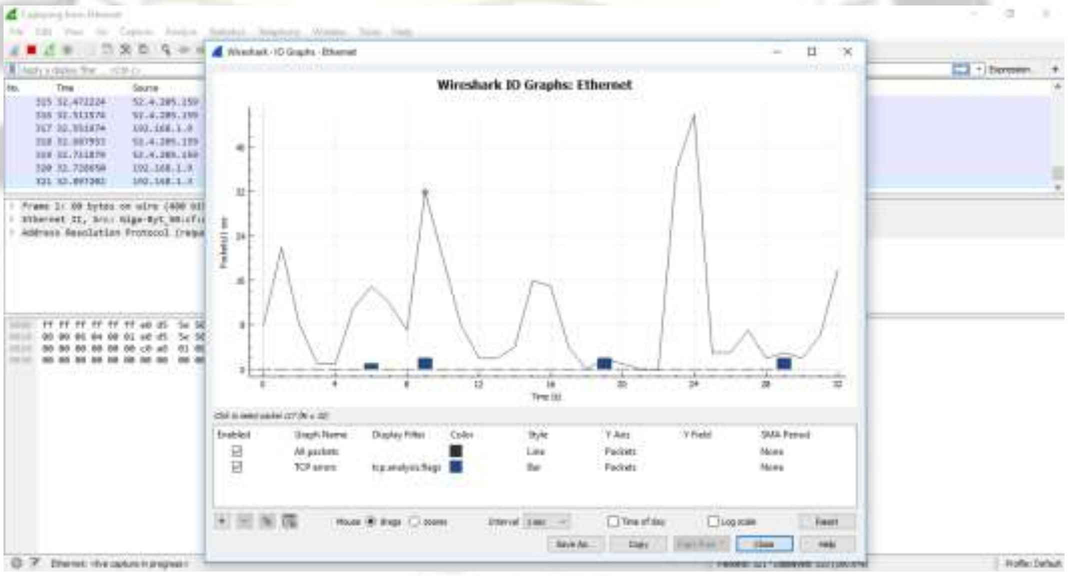
3. **Packet Details and Bytes**: Click on a packet to view detailed information in the Packet Details Pane, which breaks down the protocols and fields. The Packet Bytes Pane shows the raw data.

- **Analysis and Statistics & Filters**

1. **Basic Analysis**: Use the Packet Details Pane to analyze specific fields of interest. This can help you understand the packet structure and contents.
2. **Statistics**: Wireshark offers various statistical tools under the Statistics menu:
   - **Summary**: Gives an overview of the capture.
   - **Protocol Hierarchy**: Shows the distribution of protocols.
   - **Conversations**: Lists all the conversations (pairs of source/destination addresses).
   - **Endpoints**: Displays all endpoints with packet counts and bytes.
3. **Filters**:
   - **Display Filters**: Use these to refine the view of captured packets. Syntax can be specific (e.g., ip.src == 192.168.1.1) or general (e.g., tcp).
   - **Capture Filters**: Set before capturing to limit the packets that are captured based on criteria (e.g., port 80).

4. **Saving Captures**: You can save your capture files for later analysis using File > Save As. Wireshark supports various file formats, including its native .pcapng format.
5. **Exporting Data**: To export specific packet data, you can use options like File > Export Specified Packets or File > Export Packet Dissections.

   By following these steps, you can effectively capture and analyze network traffic using Wireshark.



EXPERIMENT – 14

Aim: How to run Nmap scan.

Explanation:

- **What is an Nmap?**

Nmap (Network Mapper) is a powerful open-source tool used for network discovery and security auditing. It can be used to identify hosts and services on a network, perform security assessments, and detect vulnerabilities.

- **What is an Nmap Scan?**

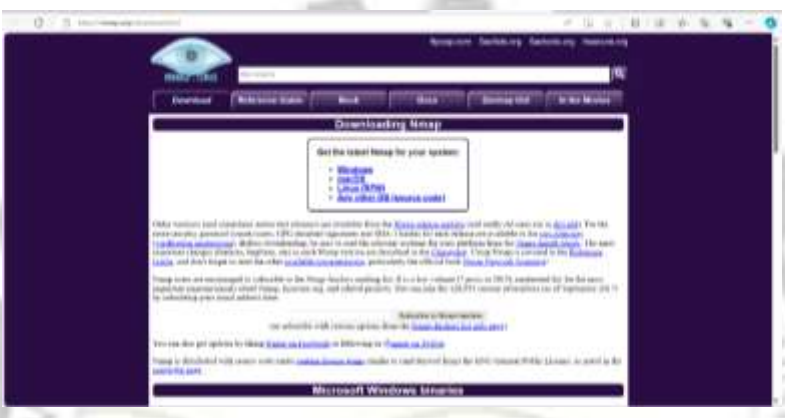
An Nmap scan involves sending packets to target hosts and analysing the responses to gather information about them, such as:

- Live hosts
- Open ports
- Running services
- Operating system details

Nmap is cross-platform. It runs on various operating systems, including:

- **Windows**
- **macOS**
- **Linux** (various distributions)

Step 1: Download the installer from the [Nmap download page]().



Downloading Nmap

Get the latest Nmap for your system:

- Windows
- macOS
- Linux (RPM)
- Any other OS (source code)

[illegible]

Microsoft Windows binaries

Step 2: Launch Zenmap

- Open Zenmap from your applications or program list.

Step 3: Enter Target Information

1. **Target Field**: In the main Zenmap window, you'll see a field labelled "Target."
2. **Enter Target**: Input the IP address, hostname, or range of addresses you want to scan (e.g., 192.168.1.1 or 192.168.1.0/24).

Step 4: Choose a Profile

1. **Profile Drop-down**: Below the target field, there's a drop-down menu labelled "Profile."
2. **Select a Scan Type**: Choose from predefined scan types, such as:

- **Intense scan**: Comprehensive scan with OS detection, service version detection, etc.
- **Quick scan**: Faster scan that checks for open ports.
- **Ping scan**: Checks which hosts are up without scanning ports.

Step 5: Start the Scan

1. **Click "Scan"**: After entering the target and selecting a profile, click the "Scan" button.
2. **View Progress**: Zenmap will start the scan, and you'll see progress updates in the window.

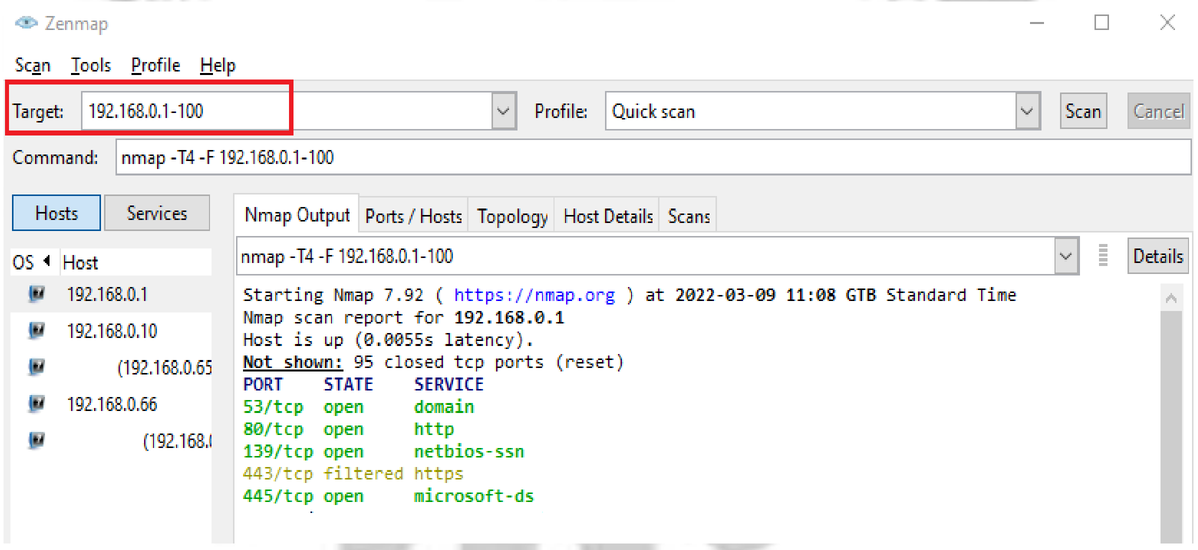
Step 6: Analyse Results

1. **Scan Results**: Once the scan is complete, results will be displayed in several tabs:
   - **Nmap Output**: Detailed output similar to what you would see in the command-line interface.
   - **Topology**: Visual representation of the network and its devices.
   - **Ports/Hosts**: A list of open ports and services for the scanned hosts.
   - **Host Details**: Information about each host, including OS detection, service versions, and more.

2. **Inspect Specific Hosts/Ports**: Click on individual hosts or ports to see more detailed information.

Step 7: Save and Compare Scans (Optional)

- **Save Results**: You can save your scan results for future reference by going to File > Save Scan and choosing a location.
- **Compare Scans**: If you run multiple scans, you can compare results to track changes over time by using File > Compare.



```
Starting Nmap 7.92 ( https://nmap.org ) at 2022-03-09 11:08 GTB Standard Time
Nmap scan report for 192.168.0.1
Host is up (0.0055s latency).
Not shown: 95 closed tcp ports (reset)
PORT     STATE    SERVICE
53/tcp   open     domain
80/tcp   open     http
139/tcp  open     netbios-ssn
443/tcp  filtered https
445/tcp  open     microsoft-ds
```

EXPERIMENT – 15

Aim: Operating System Detection using Nmap

Explanation:

Operating System (OS) detection using Nmap is a powerful feature that allows you to identify the operating system and version running on a target host. This can be useful for network management, security assessments, and vulnerability scanning. Here's how it works and how to perform OS detection with Nmap.

How OS Detection Works

Nmap uses several techniques to determine the operating system of a target:

1. **TCP/IP Stack Fingerprinting**: Nmap sends a series of TCP and UDP packets to the target and analyzes the responses. Different operating systems implement the TCP/IP stack differently, which allows Nmap to identify the OS based on these variations.
2. **Service Version Detection**: By identifying the services running on open ports and their versions, Nmap can make educated guesses about the underlying OS.
3. **Network Behavior**: Nmap may also analyze the way a host responds to different types of network

traffic, which can provide further clues about the OS.

Performing OS Detection with Nmap

To perform OS detection using Nmap, follow these steps:

Step 1: Open a Terminal or Command Prompt

Launch your command line interface (Command Prompt on Windows, Terminal on macOS/Linux).

Step 2: Run an Nmap Command with OS Detection Option

Use the -O flag to enable OS detection. Here's the basic command format:

```
nmap -O [target]
```

- **[target]**: Replace this with the IP address, hostname, or range of addresses you want to scan.

Example Command

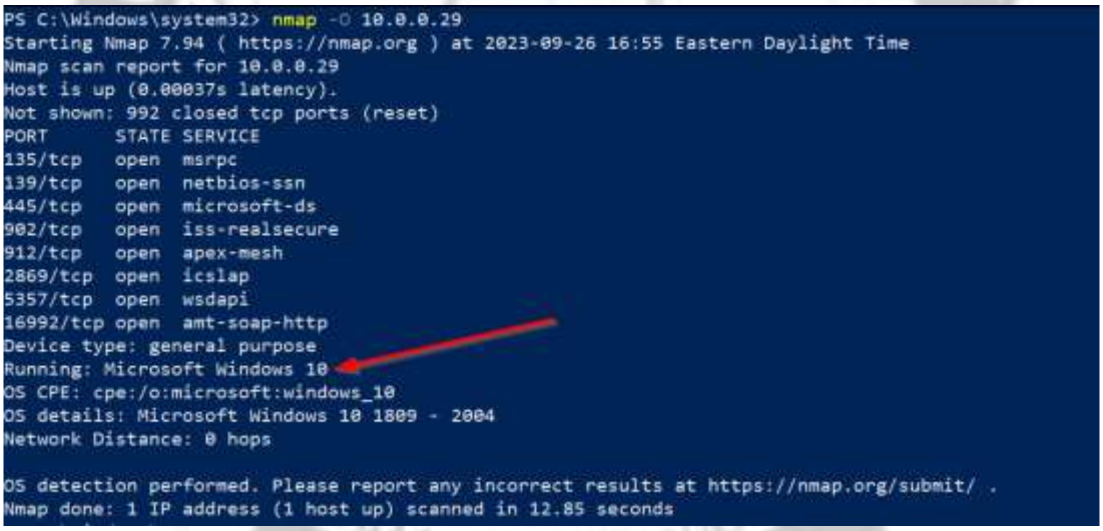
To scan a single host for its operating system, you might use:

```
nmap -O 192.168.1.1
```

Step 3: Review the Output

Once the scan is complete, Nmap will provide output that includes:

- **OS Details**: It will attempt to identify the operating system and version based on the data collected.
- **Accuracy Level**: Nmap often gives an accuracy percentage to indicate how confident it is in the detection.



```
PS C:\Windows\system32> nmap -O 10.0.0.29
Starting Nmap 7.94 ( https://nmap.org ) at 2023-09-26 16:55 Eastern Daylight Time
Nmap scan report for 10.0.0.29
Host is up (0.00037s latency).
Not shown: 992 closed tcp ports (reset)
PORT      STATE SERVICE
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
902/tcp   open  iss-realsecure
912/tcp   open  apex-mesh
2869/tcp  open  icslap
5357/tcp  open  wsdapi
16992/tcp open  amt-soap-http
Device type: general purpose
Running: Microsoft Windows 10
OS CPE: cpe:/o:microsoft:windows_10
OS details: Microsoft Windows 10 1809 - 2004
Network Distance: 0 hops

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 12.85 seconds
```

EXPERIMENT – 16

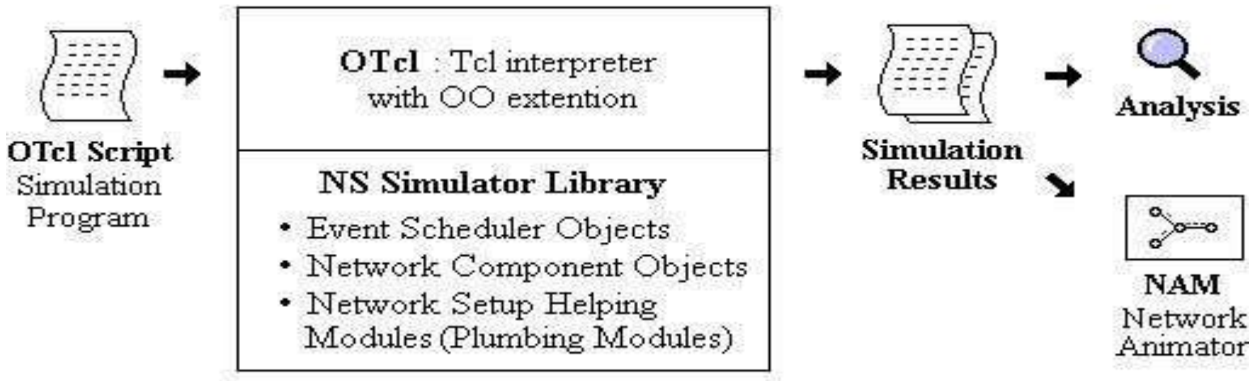
Aim: Understanding about NS2 Simulator in detail

Explanation:

i. Introduction:

NS2 is an open-source simulation tool that runs on Linux. It is a discreet event simulator targeted at networking research and provides substantial support for simulation of routing, multicast protocols and IP protocols, such as UDP, TCP, RTP and SRM over wired and wireless (local and satellite) networks. Widely known as NS2, is simply an event driven simulation tool. 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 behaviours.

Basic Architecture of NS2


OTcl Script
Simulation Program
OTcl : Tcl interpreter with OO extention
NS Simulator Library
• Event Scheduler Objects
• Network Component Objects
• Network Setup Helping Modules (Plumbing Modules)
Simulation Results
Analysis
NAM
Network Animator

TCL (Tool Command Language)

TCL (Tool Command Language) is a scripting language that is widely used in various applications, including network simulation with NS2.

Key Components of TCL

1. **Syntax and Structure**:
   - **Commands**: TCL commands are simple strings that can be executed, often consisting of a command name followed by its arguments.
   - **Variables**: Variables are defined using the set command. For example, set varName value.
   - **Lists**: TCL supports lists, which are sequences of elements that can be manipulated using various commands.
2. **Control Structures**:

   - **Conditionals**: Use if, else, and elseif for branching logic.
   - **Loops**: Supports for, while, and foreach for iteration.
   - **Procedures**: Define reusable code blocks with the proc command, e.g., proc procedureName {args} { ... }.

3. **Namespaces**: TCL supports namespaces for organizing commands and variables, helping avoid naming conflicts.
4. **Error Handling**: Provides mechanisms to catch and handle errors using the catch command.
5. **Event Handling**: Supports event-driven programming, allowing for the handling of events in simulations.

In the context of NS2, TCL is used to:

- **Define Network Topologies**: Users create simulations by specifying nodes, links, and protocols in a script.
- **Control Simulation Parameters**: Users can adjust parameters such as simulation time, traffic types, and routing protocols.
- **Analyze Results**: TCL scripts can be used to parse trace files and generate reports on simulation outcomes.

ii & iii. Simulate a three-node point-to-point network with a duplex link between them. Set the queue size and vary the bandwidth and find the number of packets dropped.

Steps:

Step 1: Set Up Your Environment

- Make sure NS2 is installed on your system. Open your terminal or command prompt.

Step 2: Define the Network Topology

- Create three nodes: Node 0, Node 1, and Node 2.

Step 3: Configure Link Parameters

- Connect Node 0 to Node 1 and Node 1 to Node 2 using duplex links.
- Set a queue size (e.g., 10 packets) for the links.
- Define different bandwidth settings for the links (e.g., 1Mb, 5Mb, 10Mb).

Step 4: Create Traffic Sources

- Use a TCP agent attached to Node 0 to send packets.
- Attach a null agent (sink) to Node 2 to receive packets.

Step 5: Send Packets

- Program the TCP agent to send a specified number of packets (e.g., 100 packets) to Node 2.

Step 6: Run the Simulation

- Execute the simulation for a predetermined duration (e.g., 10 seconds).

Step 7: Analyze Output

- Check the generated trace file (sim.tr) for information on packet drops.

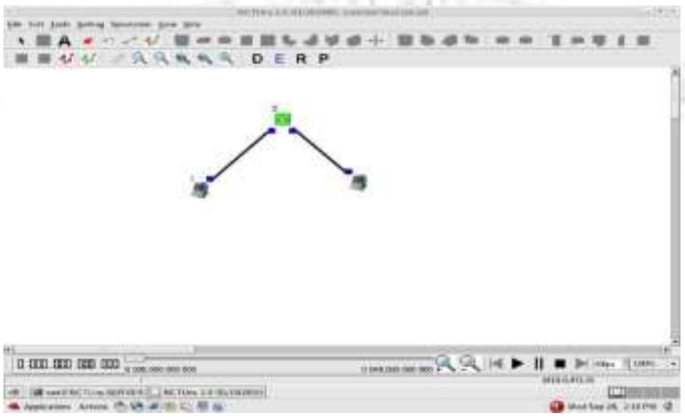
Step 8: Repeat for Different Bandwidths

- Modify the bandwidth settings in your configuration and repeat Steps 5-7 for each bandwidth (1Mb, 5Mb, 10Mb).

Step 9: Collect Results

- Record the number of packets dropped for each bandwidth setting to analyze how varying bandwidth affects packet loss.

The topology will look like below,

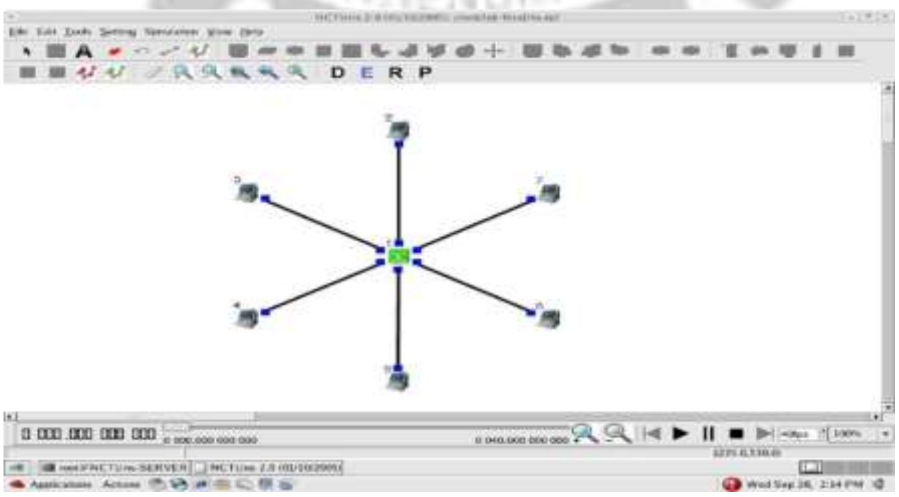


iv. Simulate the transmission of ping messages over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.

Steps:

Step 1: Click on the subnet icon on the toolbar and then click on the screen of the working window.
Step 2: Select the required number of hosts and a suitable radius between the host and the switch.
Step 3: In the edit mode, get the IP address of one of the hosts say, host 1 and then for the other host say, host2 set the drop packet and no: of collisions statistics as described in the earlier experiments
Step 4: Now run the simulation.

The topology will look like below,



v. Simulate to Compare Data Rate& Throughput.

Steps:

Step 1: Set Up Your Environment

- Ensure NS2 is installed on your system and accessible via the terminal.

Step 2: Define the Network Topology

- Create a simple topology, such as two nodes (Node 0 and Node 1) connected by a duplex link.

Step 3: Configure Link Parameters

- Set the link parameters, including bandwidth and delay. You might want to test different bandwidths (e.g., 1Mb, 5Mb, 10Mb).
- Example parameters:
  - Bandwidth: 1Mb
  - Delay: 10ms

Step 4: Create Traffic Sources

- Use a TCP agent at Node 0 to send data packets to Node 1.
- Attach a null agent to Node 1 to receive packets.

Step 5: Set Packet Size and Transmission

- Decide on the packet size (e.g., 1000 bytes).
- Use a loop to send a fixed number of packets (e.g., 100 packets) from Node 0 to Node 1.

Step 6: Run the Simulation

- Run the simulation for a specified duration (e.g., 20 seconds).

Step 7: Analyze Output for Throughput

- After the simulation, check the generated trace file (sim.tr) to find the throughput.
- Formula for calculating throughput,

  Throughput = Total Data Sent / Total Time

- Use tools like awk to extract relevant data from the trace file.

Step 8: Repeat for Different Data Rates

- Modify the link bandwidth in your configuration and repeat Steps 5-7 for each bandwidth setting.
- For example, change the bandwidth to 5Mb and then to 10Mb, observing how throughput changes.

Step 9: Collect and Compare Results

- Record the throughput results for each bandwidth setting.
- Create a summary table to compare the data rates (bandwidth) and the observed throughput.

✦✧✦ THE END ✦✧✦