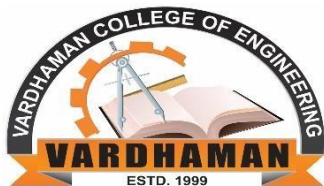


# VARDHAMAN COLLEGE OF ENGINEERING (Autonomous)

NAAC-A Grade, NBA Accredited: Affiliated to  
JNTUH, Hyderabad ISO 9001:2008 Certified.

Shamshabad - 501 218, Hyderabad

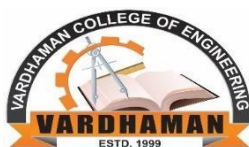


DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**(II B.Tech- II SEMESTER)**

**(VCE-R19)**

Name :  
Roll No :  
Branch : Computer Science & Engineering  
Section : CSE - C  
Course : COMPUTER NETWORKS  
Academic Year : 2020 - 2021



# VARDHAMAN COLLEGE OF ENGINEERING

(Autonomous)

NAAC-A Grade, NBA Accredited:  
Affiliated to JNTUH, Hyderabad ISO  
9001:2008 Certified.

Shamshabad - 501 218, Hyderabad

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

## CERTIFICATE

*Certified that this is the bonafied record of practical work done by*

*Mr./Ms. ....Roll Number .....*

*of B.Tech./~~M-Tech~~/MBA*

*in the ..... laboratory during the year.....*

*No. of Experiments done:*

*Total No. of Experiments:*

***Date:***

***HOD***

***Staff Member Incharge***

---

*Roll No .....*

*Submitted for the practical exam held on .....*

Internal Examiner

External Examiner

# VARDHAMAN COLLEGE OF ENGINEERING

(Autonomous)

NAAC-A Grade, NBA Accredited: Affiliated to JNTUH, Hyderabad ISO 9001:2008 Certified.

Shamshabad - 501 218, Hyderabad

## INDEX

S.No.	Date	Title of the Experiment	Page No.	Marks	Sign. Of the Staff
1		Implementation of bit stuffing.	4		
2		Implementation of hamming code	6		
3		Implementation of data encryption and decryption.	11		
4		Create scenario, simulate, and study the evolution of the Stop and Wait Protocol.	13		
5		Create scenario, simulate, and study the evolution of Go Back N protocol.	17		
6		Create scenario, simulate, and study the evolution of Selective Repeat Protocol.	21		
7		Implementation of 16-bit CRC Error Detection Technique.	25		
8		Implementation of Dijkstra's Algorithm for computing the shortest path in a graph.	29		
9		Implementation of Distance vector routing algorithm.	32		
10		Implement, and verify through a simulator, a program to create sub-network and assign addresses based on the number of hosts connected to the network.	36		
11		Create a simulator to transfer of files from PC to PC using packet tracer software. (Additional Practice)	41		
12		Implementation of Iterative and Concurrent Echo Server using Connection Oriented Protocol (TCP) and Connection Less Protocol (UDP). (Additional Practice)	43		

## WEEK-1: Implementation of bit stuffing.

Whenever the sender's data link layer encounters five consecutive 1s in the data, it automatically stuffs a zero bit into the outgoing bit stream. This technique is called bit stuffing. When the receiver sees five consecutive 1s in the incoming data stream, followed by a zero bit, it automatically destuffs the 0 bit. The boundary between two frames can be determined by locating the flag pattern.

(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

(b) 0 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 0 0 1 0

Stuffed bits

### ALGORITHM:

1. Read a bit string
2. Trace 5 bits in the string
3. Check if 5 bits are 11111 then shift a '0' bit after it
4. Repeat 2 & 3 until end of string
5. Display stuffed bit string

### CODE:

```
#include<stdio.h>

#include<conio.h>

void main()
{
    char a[20],fs[50]="",t[6],r[5];
    inti,j,p=0,q=0;
    clrscr();
    printf("enter bit string : ");
    scanf("%s",a);
    strcat(fs,"01111110");
    if(strlen(a)<5)
    {
        strcat(fs,a);
    }
    else
```

```

{
    for(i=0;i<strlen(a)-4;i++)
    {
        for(j=i;j<i+5;j++)
        {
            t[p++]=a[j];
        }
        t[p]='\0';
        if(strcmp(t,"11111")==0)
        {
            strcat(fs,"111110"); i=j-1;
        }
        else
        {
            r[0]=a[i];
            r[1]='\0';
            strcat(fs,r);
        }
        p=0;
    }
}

```

## OUTPUT

```

enter the bits to be stuffed1011111100010101111110
the data after stuffed is 101111101000101011111010
-----
Process exited after 10.12 seconds with return value 53
Press any key to continue . . . █

```

## WEEK-2 IMPLEMENTATION OF HAMMING CODE.

Hamming code is a set of error-correction codes that can be used to **detect and correct the errors** that can occur when the data is moved or stored from the sender to the receiver. It is **technique developed by R.W. Hamming for error correction**.

The theoretical lower limit can, in fact, be achieved using a method due to Hamming (1950). The bits of the codeword are numbered consecutively, starting with bit 1 at the left end, bit 2 to its immediate right, and so on. The bits that are powers of 2 (1, 2, 4, 8, 16, etc.) are check bits. The rest (3, 5, 6, 7, 9, etc.) are filled up with the m data bits. Each check bit forces the parity of some collection of bits, including itself, to be even (or odd). A bit may be included in several parity computations. To see which check bits the data bit in position k contributes to, rewrite k as a sum of powers of 2. For example,  $11 = 1 + 2 + 8$  and  $29 = 1 + 4 + 8 + 16$ . A bit is checked by just those check bits occurring in its expansion (e.g., bit 11 is checked by bits 1, 2, and 8).

### SOURCE CODE :

```
#include <stdio.h>

#include <string.h>

#include <math.h>

#include <malloc.h>

int i, j, k=0, v;

void find_r_encode(int m, int * r) {
    int fact=4;
    while(m+(*r)+1 > fact) {
        fact *= 2;
        (*r)++;
    }
    return;
}

void find_r_decode(int q, int * r) {
    int fact=4;
    while(q+1 > fact) {
        fact *= 2;
        (*r)++;
    }
}
```

```

        return;
    }
    void x_or(char * temp, char * val) {
        for(i=0; i<strlen(temp); i++)
            temp[i] = (temp[i]=='0' && val[i]=='0' || temp[i]=='1' && val[i]=='1') ? '0' :
'1';
        return;
    }
    void final_frame(char * f, char * temp, char * ans, int r, int len) {
        j = r-1;
        k=0;
        for(i=0; i<len; i++) {
            float p = log(len-i)/log(2);
            ans[i] = (p != (int)p) ? f[k++] : temp[j--];
        }
        ans[i] = '\0';
        return;
    }
    void correction(char * temp, int r) {
        int fa = 1, fla = 0;
        for(i=0; i<r; i++) {
            fla += fa * (temp[i]-48);
            fa *= 2;
        }
        if(fla!=0)
            printf("\n\n%dth bit is Wrong in the data Transmission.", fla);
        else
            printf("\n\nData Transferred Successfully!!!");
        return;
    }
    void encode(char * f, char * ans) {

```

```

    int r=2, len;
    find_r_encode(strlen(f), &r);
    len = strlen(f) + r;
    char * temp = (char*)malloc(100);
    char * val = (char*)malloc(100);
    for(i=0; i<r; i++)
        temp[i] = '0';
    temp[i] = '\0';
    for(i=0; i<len; i++) {
        v = len-i;
        float p = log(len-i)/log(2);
        if(p != (int)p && f[k++] == '1') {
            for(j=0; j<r; j++) {
                val[j] = v%2 + 48;
                v /= 2;
            }
            val[j] = '\0';
            x_or(temp, val);
        }
    }
    final_frame(f, temp, ans, r, len);
}

void decode(char *f) {
    int r=2, len = strlen(f);
    find_r_decode(len, &r);
    char * temp = (char*)malloc(100);
    char * val = (char*)malloc(100);
    for(i=0; i<r; i++)
        temp[i] = '0';
    temp[i] = '\0';

```



```

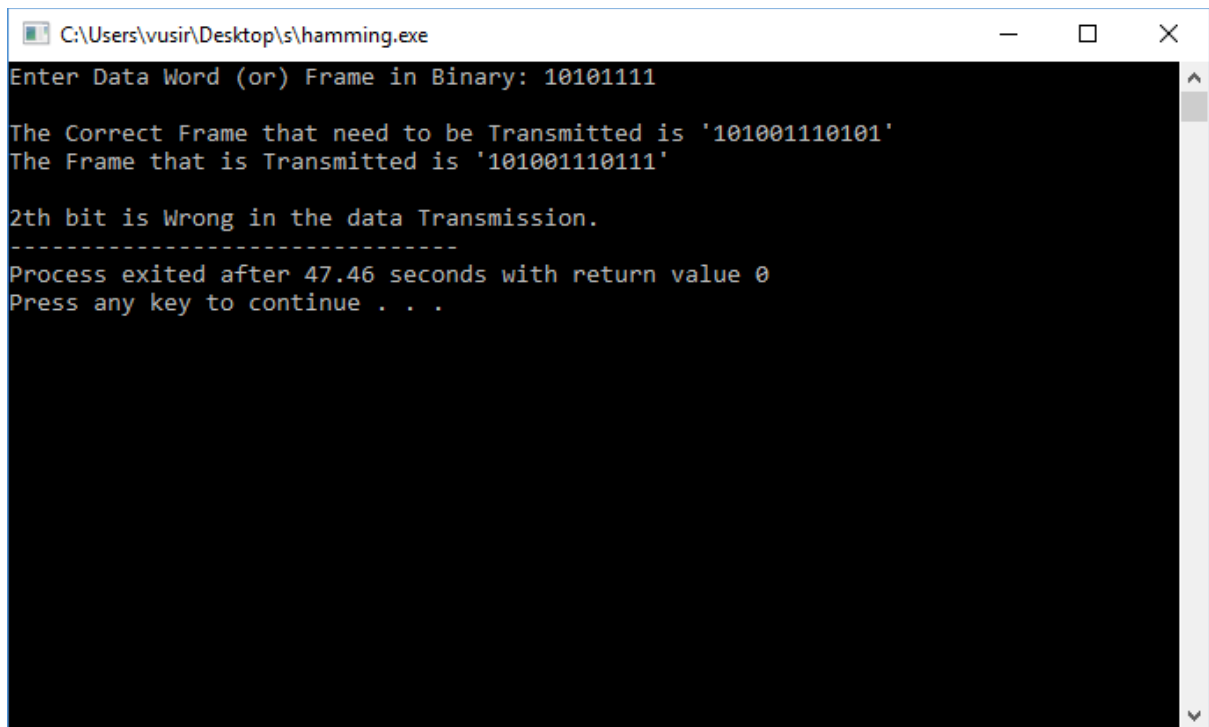
        for(i=0; i<len; i++) {
            v = len-i;
            if(f[i] == '1') {
                for(j=0; j<r; j++) {
                    val[j] = v%2 + 48;
                    v /= 2;
                }
                val[j] = '\0';
                x_or(temp, val);
            }
        }
        correction(temp, r);
    }
}

void only_decode(int dr) {
    char * ans = (char*)malloc(100);
    scanf("%s", ans);
    decode(ans);
    return;
}

int main() {
    char * f = (char*)malloc(100);
    printf("Enter Data Word (or) Frame in Binary: ");
    scanf("%s", f);
    char * ans = (char*)malloc(100);
    /**encode**/
    encode(f, ans);
    printf("\nThe Correct Frame that need to be Transmitted is '%s'", ans);
    /** some data has changed let it be 2nd position**/
    ans[strlen(ans)-2] = '1';
    printf("\nThe Frame that is Transmitted is '%s'", ans);
}

```

```
    /**decode**/  
    decode(ans);  
    // only_decode(4);  
    return 0;  
}
```



```
C:\Users\vusir\Desktop\s\hamming.exe  
Enter Data Word (or) Frame in Binary: 10101111  
The Correct Frame that need to be Transmitted is '101001110101'  
The Frame that is Transmitted is '101001110111'  
  
2th bit is Wrong in the data Transmission.  
-----  
Process exited after 47.46 seconds with return value 0  
Press any key to continue . . .
```

## WEEK-3 IMPLEMENTATION OF DATA ENCRYPTION AND DECRYPTION.

RSA Algorithm is used to encrypt and decrypt data in modern computer systems and other electronic devices. RSA algorithm is an asymmetric cryptographic algorithm as it creates 2 different keys for the purpose of encryption and decryption. It is public key cryptography as one of the keys involved is made public. RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman who first publicly described it in 1978.

RSA makes use of prime numbers (arbitrary large numbers) to function. The public key is made available publicly (means to everyone) and only the person having the private key with them can decrypt the original message.

### Working of RSA Algorithm

RSA involves use of public and private key for its operation. The keys are generated using the following steps: -

1. Two prime numbers are selected as  $p$  and  $q$
2.  $n = pq$  which is the modulus of both the keys.
3. Calculate totient =  $(p-1)(q-1)$
4. Choose  $e$  such that  $e > 1$  and coprime to totient which means  $\gcd(e, \text{totient})$  must be equal to 1,  $e$  is the public key
5. Choose  $d$  such that it satisfies the equation  $de = 1 + k(\text{totient})$ ,  $d$  is the private key not known to everyone.
6. Cipher text is calculated using the equation  $c = m^e \bmod n$  where  $m$  is the message.
7. With the help of  $c$  and  $d$  we decrypt message using equation  $m = c^d \bmod n$  where  $d$  is the private key.

**Note:** If we take the two prime numbers very large it enhances security but requires implementation of Exponentiation by squaring algorithm and square and multiply algorithm for effective encryption and decryption. For simplicity the program is designed with relatively small prime numbers. Program can be written as per the understanding of the student.

### CODE

```
#include <stdio.h>

int main()
{
    int i, x;
    char str[100];
    printf("\nPlease enter a string:\t");
    gets(str);
    printf("\nPlease choose following options:\n");
    printf("1 = Encrypt the string.\n");
    printf("2 = Decrypt the string.\n");
    scanf("%d", &x);
    switch(x)
    {
        case 1:
```

```

    for(i = 0; (i < 100 && str[i] != '\0'); i++)
        str[i] = str[i] + 3;
    printf("\nEncrypted string: %s\n", str);
    break;
case 2:
    for(i = 0; (i < 100 && str[i] != '\0'); i++)
        str[i] = str[i] - 3;
    printf("\nDecrypted string: %s\n", str);
    break;
default:
    printf("\nError\n");
}
return 0;
}

```

## OUTPUT

```

Please enter a string:  VARDHAMAN COLLEGE OF ENGINEERING

Please choose following options:
1 = Encrypt the string.
2 = Decrypt the string.
1

Encrypted string: YDUGKDPDQ#FROOHJH#RI#HQJLQHHULQJ

-----
Process exited after 66.96 seconds with return value 0
Press any key to continue . . . █

```

```

Please enter a string:  YDUGKDPDQ#FROOHJH#RI#HQJLQHHULQJ

Please choose following options:
1 = Encrypt the string.
2 = Decrypt the string.
2

Decrypted string: VARDHAMAN COLLEGE OF ENGINEERING

-----
Process exited after 5.145 seconds with return value 0
Press any key to continue . . .

```

## WEEK-4 Simulation of Stop and Wait protocol

### Characteristics

- Used in Connection-oriented communication.
- It offers error and flow control
- It is used in Data Link and Transport Layers
- Stop and Wait ARQ mainly implements Sliding Window Protocol concept with Window Size 1

### Useful Terms:

- Propagation Delay: Amount of time taken by a packet to make a physical journey from one router to another router.
- Propagation Delay = (Distance between routers) / (Velocity of propagation)
- RoundTripTime (RTT) = 2\* Propagation Delay
- TimeOut (TO) = 2\* RTT
- Time To Live (TTL) = 2\* TimeOut. (Maximum TTL is 180 seconds)

### Simple Stop and Wait:

#### Sender:

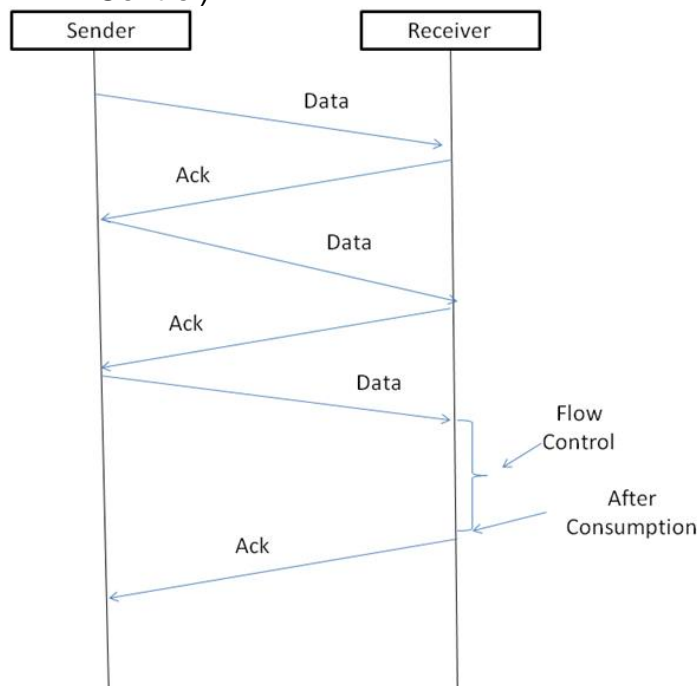
Rule 1) Send one data packet at a time.

Rule 2) Send next packet only after receiving acknowledgement for previous.

#### Receiver:

Rule 1) Send acknowledgement after receiving and consuming of data packet.

Rule 2) After consuming packet acknowledgement need to be sent (Flow Control)



### Characteristics of Stop and Wait ARQ:

It uses link between sender and receiver as half duplex link

Throughput = 1 Data packet/frame per RTT

If Bandwidth\*Delay product is very high, then stop and wait for the protocol is not so useful. The sender has to keep waiting for acknowledgments before sending the processed next packet.

It is an example of "Closed Loop OR connection-oriented " protocols

It is a special category of SWP where its window size is 1

Irrespective of number of packets sender is having stop and wait for protocol requires only 2 sequence numbers 0 and 1

The Stop and Wait ARQ solves main three problems but may cause big performance issues as sender always waits for acknowledgment even if it has next packet ready to send. Consider a situation where you have a high bandwidth connection and propagation delay is also high (you are connected to some server in some other country through a high-speed connection). To solve this problem, we can send more than one packet at a time with a larger sequence numbers. We will be discussing these protocols in next articles.

So Stop and Wait ARQ may work fine where propagation delay is very less, for example, LAN connections but performs badly for distant connections like satellite connection.

### SOURCE CODE:

#### Sender:

```
#include <stdio.h>
int i;
int main() {
    int flag = 0;
    char frame[8];
    FILE *f1, *f2;
    while(1) {
        f1 = fopen("file1.txt", "r");
        char ch = fgetc(f1);
        fclose(f1);
        if(ch == '1') {
            printf("\nDidn't received ACK in time.");
        }
        if(ch == '0') {
            printf((flag!=0)? "\nACK received\n": "");
            flag = 1;
            printf("\nEnter frame data: ");
            scanf("%s", frame);

            f2 = fopen("file2.txt", "w");
            fprintf(f2, "%s", frame);
            fclose(f2);

            f1 = fopen("file1.txt", "w");
            putc('1', f1);
        }
    }
}
```

```

        fclose(f1);
        Sleep(1000);
        printf("\nData sent Successfully!!!");
        Sleep(1000);
    }
}
return 0;
}

```

## Receiver:

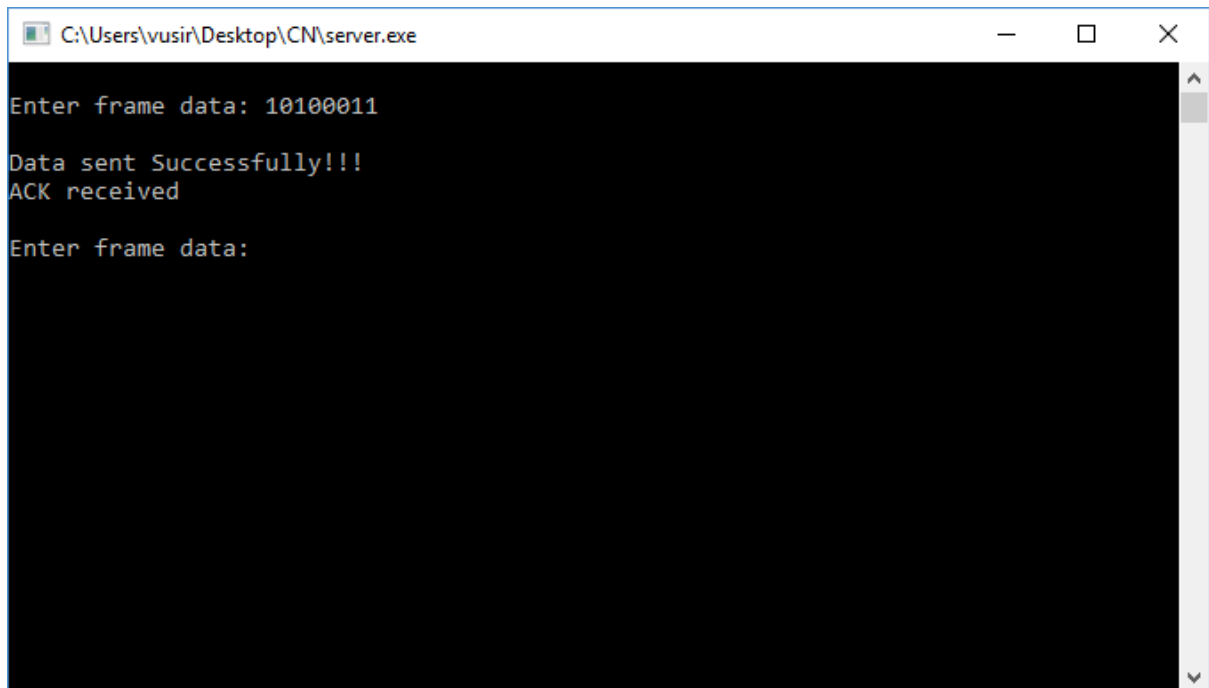
```

#include <stdio.h>
#include <malloc.h>
int i;
int main() {
    char * frame = (char*)malloc(9);
    FILE *f1, *f2;
    while(1) {
        f1 = fopen("file1.txt", "r");
        char ch = fgetc(f1);
        if(ch == '1') {
            f2 = fopen("file2.txt", "r");
            fscanf(f2, "%s", frame);
            fclose(f2);

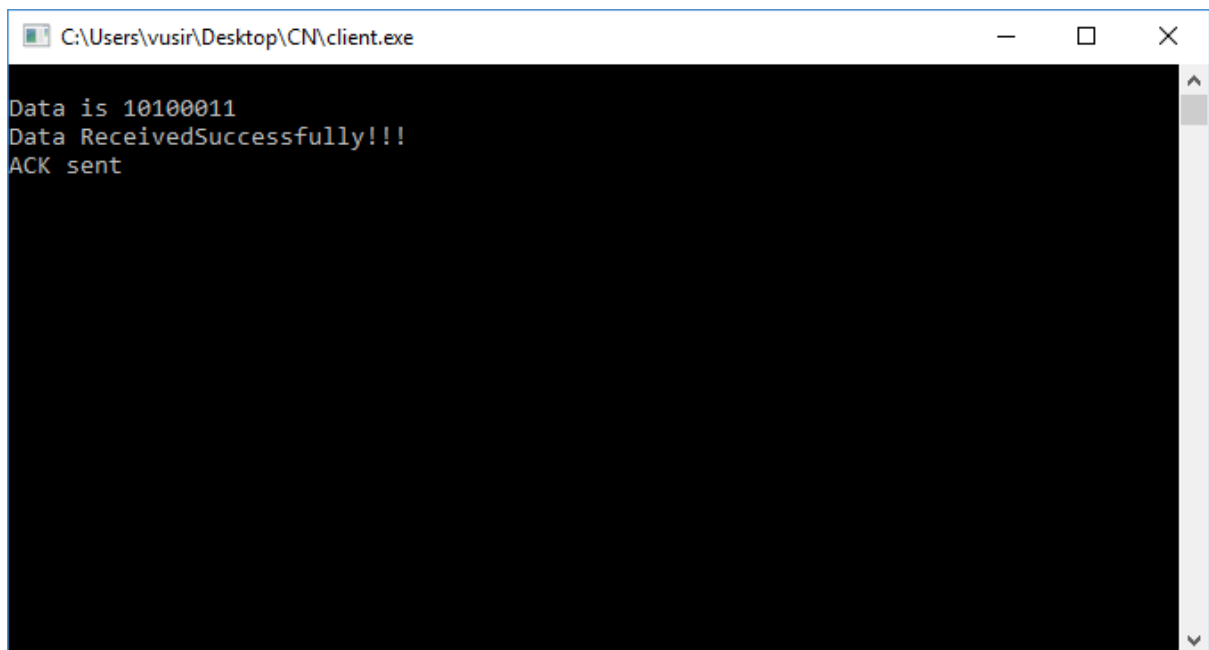
            printf("\nData is ");
            puts(frame);
            f1 = fopen("file1.txt", "w");
            putc('0', f1);
            printf("Data Received Successfully!!!");
            Sleep(1000);
            printf("\nACK sent\n");
        }
        fclose(f1);
    }
    return 0;
}

```

## OUTPUT:



```
C:\Users\vusir\Desktop\CN\server.exe
Enter frame data: 10100011
Data sent Successfully!!!
ACK received
Enter frame data:
```



```
C:\Users\vusir\Desktop\CN\client.exe
Data is 10100011
Data ReceivedSuccessfully!!!
ACK sent
```



## WEEK-5 Simulation of Go back N protocol

Go Back n is a connection-oriented protocol in which the transmitter has a window of sequence numbers that may be transmitted without acknowledgment. The receiver will only accept the next sequence number it is expecting - other sequence numbers are silently ignored.

The protocol simulation shows a time-sequence diagram with users A and B, protocol entities A and B that support them, and a communications medium that carries messages. Users request data transmissions with `DatReq(DATAN)`, and receive data transmissions as `DatInd(DATAN)`. Data messages are simply numbered `DATA0`, `DATA1`, etc. without explicit content. The transmitting protocol sends the protocol message `DT(n)` that gives only the sequence number, not the data. Once sequence numbers reach a maximum number (like 7), they wrap back round to 0. An acknowledgement `AK(n)` means that the `DT` message numbered `n` is the next one expected (i.e. all messages up to but not including this number have been received). Since sequence numbers wrap round, an acknowledgment with sequence number 1 refers to messages 0, 1, 7, 6, etc. Note that if a `DT` message is received again due to re-transmission, it is acknowledged but discarded.

The protocol has a maximum number of messages that can be sent without acknowledgment. If this window becomes full, the protocol is blocked until an acknowledgment is received for the earliest outstanding message. At this point, the transmitter is clear to send more messages.

The receiver delivers the protocol messages `DT(n)` to the user in order. Any received out of order are ignored.

### SOURCE CODE:

#### Sender:

```
#include<stdio.h>

#include<stdlib.h>

int main(){
    FILE *fp = NULL;
    int n;
    printf("Enter window size: ");
    scanf("%d", &n);
    char frame[n][8];
    fp = fopen("file1.txt", "w");
    putc('0', fp);
    fclose(fp);
    while (1){
        fp = fopen("file1.txt", "r");
```

```

char ch = fgetc(fp);
fclose(fp);
if (ch == '0'){
    fp = fopen("file1.txt", "w");
    fprintf(fp, "5\n");
    printf("Enter %d frames of 4 bit: \n", n);
    int i;
    for (i = 0; i < n; i++){
        scanf("%s", frame[i]);
        fflush(stdin);
        fprintf(fp, frame[i]);
        fprintf(fp, "\n");
    }
    fclose(fp);
}
else if (ch == '-'){
    fp = fopen("file1.txt", "r");
    int pos;
    ch = fgetc(fp);
    ch = fgetc(fp);
    pos = ch - '0';
    fclose(fp);

    fp = fopen("file1.txt", "w");
    fprintf(fp, "1");
    fprintf(fp, "\n");
    printf("Retransmission of following frames\n");
    int i;
    for (i = pos; i < n; i++){
        fprintf(fp, frame[i]);
        printf("%s\n", frame[i]);
        fprintf(fp, "\n");
    }
}

```

```

        }
        fclose(fp);
    }
}
return 0;
}

```

### Receiver:

```

#include<stdio.h>
#include<stdlib.h>
int main(){
    FILE *fp = NULL;
    int n;
    printf("Enter window size: ");
    scanf("%d", &n);
    while (1){
        fp = fopen("file1.txt", "r");
        char ch = fgetc(fp);
        fclose(fp);
        if (ch >= '1'){
            fp = fopen("file1.txt", "r");
            char ch = fgetc(fp);
            while ((ch = fgetc(fp)) != EOF)
                printf("%c", ch);
            int error_pos;
            printf("\nEnter the number of frame that is error, -1 for no error\n");
            scanf("%d", &error_pos);
            if (error_pos > -1 && error_pos < n){
                fp = fopen("file1.txt", "w");
                fprintf(fp, "-");
                char p[2];
                itoa(error_pos, p, 10);
            }
        }
    }
}

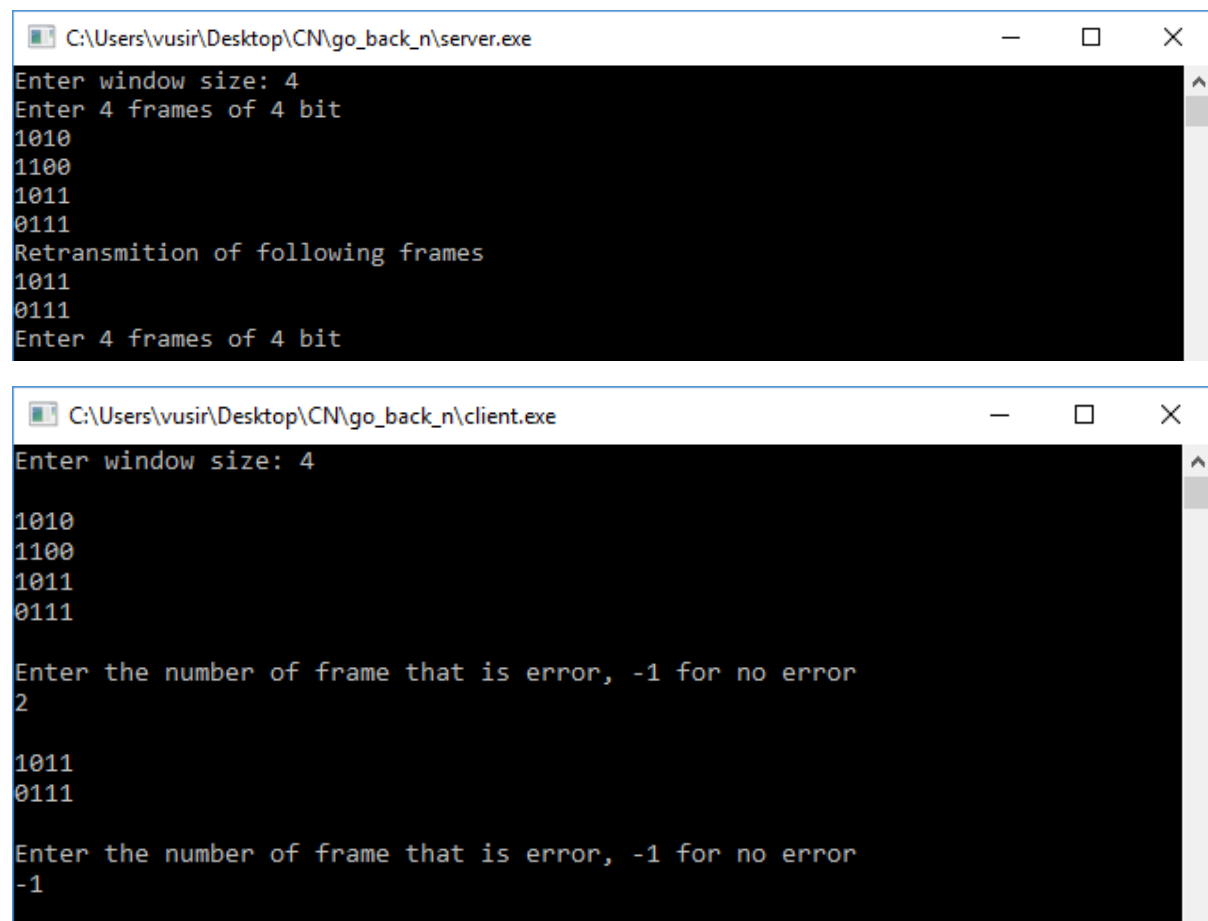
```

```

        fprintf(fp, p);
        fclose(fp);
    }
    if (error_pos == -1){
        fp = fopen("file1.txt", "w");
        fprintf(fp, "0");
        fclose(fp);
    }
}
}
return 0;
}

```

## OUTPUT:



The image shows two terminal windows side-by-side, illustrating the execution of a Go-Back-N protocol simulation.

**Top Window: C:\Users\vusir\Desktop\CN\go\_back\_n\server.exe**

```

Enter window size: 4
Enter 4 frames of 4 bit
1010
1100
1011
0111
Retransmission of following frames
1011
0111
Enter 4 frames of 4 bit

```

**Bottom Window: C:\Users\vusir\Desktop\CN\go\_back\_n\client.exe**

```

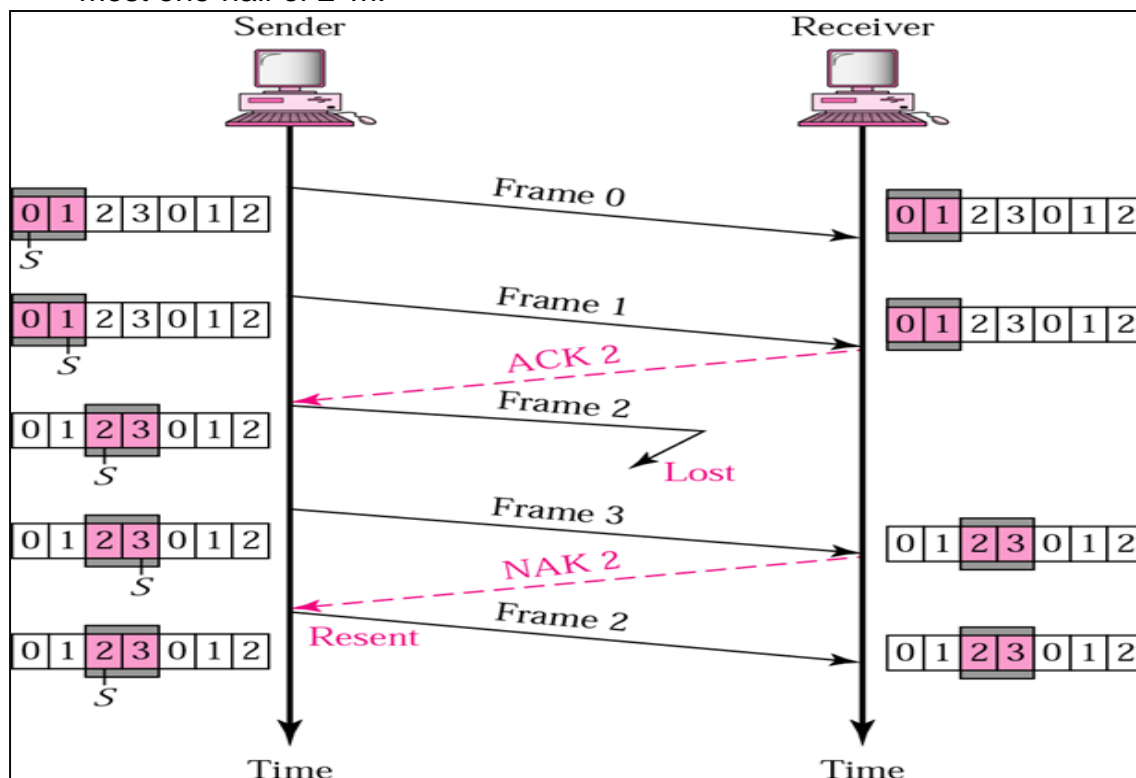
Enter window size: 4
1010
1100
1011
0111
Enter the number of frame that is error, -1 for no error
2
1011
0111
Enter the number of frame that is error, -1 for no error
-1

```

## WEEK-6 Simulation of Selective Repeat Protocol

This protocol (SRP) is mostly identical to GBN protocol, except that buffers are used and the receiver, and the sender, each maintains a window of size. SRP works better when the link is very unreliable. Because in this case, retransmission tends to happen more frequently, selectively retransmitting frames is more efficient than retransmitting all of them. SRP also requires a full duplex link. backward acknowledgments are also in progress.

- Sender's Windows (  $W_s$  ) = Receiver's Windows (  $W_r$  ).
- Window size should be less than or equal to half the sequence number in SR protocol. This is to avoid packets being recognized incorrectly. If the windows size is greater than half the sequence number space, then if an ACK is lost, the sender may send new packets that the receiver believes are retransmissions.
- The sender can transmit new packets as long as their number is with  $W$  of all unACKed packets.
- Sender retransmits un-ACKed packets after a timeout – Or upon a NAK if NAK is employed.
- Receiver ACKs all correct packets.
- Receiver stores correct packets until they can be delivered in order to the higher layer.
- In Selective Repeat ARQ, the size of the sender and receiver window must be at most one-half of  $2^m$ .



## SOURCE CODE:

### Sender:

```
#include<stdio.h>
#include<stdlib.h>
int main(){
    FILE *fp = NULL;
    int n;
    printf("Enter window size: ");
    scanf("%d", &n);
    char frame[n][8];
    fp = fopen("file1.txt", "w");
    putc('0', fp);
    fclose(fp);
    while (1){
        fp = fopen("file1.txt", "r");
        char ch = fgetc(fp);
        fclose(fp);
        if (ch == '0'){
            fp = fopen("file1.txt", "w");
            fprintf(fp, "%d\n", n);
            printf("Enter %d frames of 4 bit: \n", n);
            int i;
            for (i = 0; i < n; i++){
                scanf("%s", frame[i]);
                fflush(stdin);
                fprintf(fp, frame[i]);
                fprintf(fp, "\n");
            }
            fclose(fp);
        }
        else if (ch == '-') {
            fp = fopen("file1.txt", "r");
            int pos;
            ch = fgetc(fp);
            ch = fgetc(fp);
            pos = ch - '0';
            fclose(fp);

            fp = fopen("file1.txt", "w");
            fprintf(fp, "1\n");
            printf("Retransmission of following frames\n");
            fprintf(fp, frame[pos]);
            printf("%s\n", frame[pos]);
            fprintf(fp, "\n");
            fclose(fp);
        }
    }
    return 0;
}
```

### Receiver Code:

```
#include<stdio.h>
#include<stdlib.h>
int main(){
    FILE *fp = NULL;
    int n;
    printf("Enter window size: ");
    scanf("%d", &n);
    while (1){
        fp = fopen("file1.txt", "r");
        char ch = fgetc(fp);
        fclose(fp);
        if (ch >= '1'){
            fp = fopen("file1.txt", "r");
            char ch = fgetc(fp);
            while ((ch = fgetc(fp))!= EOF)
                printf("%c", ch);
            int error_pos;
            printf("\nEnter the number of frame that is error, -1 for no
error\n");
            scanf("%d", &error_pos);
            if (error_pos >-1 && error_pos<n){
                fp = fopen("file1.txt", "w");
                fprintf(fp, "-%d", error_pos);
                fclose(fp);
            }
            if (error_pos == -1){
                fp = fopen("file1.txt", "w");
                fprintf(fp, "0");
                fclose(fp);
            }
        }
    }
    return 0;
}
```

## OUTPUT:

```
C:\Users\vusir\Desktop\CN\selective_repeat\server.exe
Enter window size: 4
Enter 4 frames of 4 bit:
1010
1100
0011
1011
Retransmission of following frames
0011
Retransmission of following frames
1011
Enter 4 frames of 4 bit:
```

```
C:\Users\vusir\Desktop\CN\selective_repeat\client.exe
Enter window size: 4

1010
1100
0011
1011

Enter the number of frame that is error, -1 for no error
2

0011

Enter the number of frame that is error, -1 for no error
3

1011

Enter the number of frame that is error, -1 for no error
-1
```



## WEEK-7 Implement Crc Technique For Error Handling

The polynomial code, also known as a CRC (Cyclic Redundancy Check). Polynomial codes are based upon treating bit strings as representations of polynomials with coefficients of 0 and 1 only. A k-bit frame is regarded as the coefficient list for a polynomial with k terms, ranging from  $x^{k-1}$  to  $x^0$ . Such a polynomial is said to be of degree k - 1. The high-order (leftmost) bit is the coefficient of  $x^{k-1}$ ; the next bit is the coefficient of  $x^{k-2}$ , and so on. For example, 110001 has 6 bits and thus represents a six-term polynomial with coefficients 1, 1, 0, 0, 0, and 1:  $x^5 + x^4 + x^0$ .

### SOURCE CODE :

```
#include <stdio.h>

#include <malloc.h>

#include <string.h>

char x_or(char a, char b) {
    return (a=='0' && b=='0' || a=='1' && b=='1')?'0':'1';
}

void fun(char * temp, char * g, int g_l) {
    for(int i=0; i<g_l-1; i++)
        temp[i] = x_or(temp[i+1], g[i+1]);
    return;
}

void crc_code(char * f, char * g, int pos) {
    int g_l = strlen(g), f_l = strlen(f), i, j;
    if(pos == 0) {
        for(i=f_l; i<f_l+g_l-1; i++)
            f[i] = '0';
        f[i] = '\0';
    }
    char * temp = (char*)malloc(100);
    for(i=0; i<g_l; i++)
        temp[i] = f[i];
    temp[i] = '\0';
    j=g_l;
```

```

while(1) {
    // printf("\n%s", temp);
    if(temp[0] == '1') {
        fun(temp, g, g_l);
        if(j == f_l+g_l-1 && pos == 0) {
            j = f_l;
            for(i=0; i<g_l-1; i++)
                f[j++] = temp[i];
            f[j] = '\0';
            printf("\nThe crc is: ");
            for(int k=0; k<g_l-1; k++)
                printf("%c", temp[k]);
            return;
        }
        if(j == f_l && pos == 1) {
            for(i=0; i<g_l-1; i++)
                if(temp[i] != '0') {
                    printf("\nError\nThe remainder is ");
                    for(int k=0; k<g_l-1; k++)
                        printf("%c", temp[k]);
                    printf(" instead of 0's");
                    return;
                }
            printf("\nData Transferred Successfully!!!");
            return;
        }
        temp[g_l-1] = f[j];
        temp[g_l] = '\0';
    }
    else {

```

```

        for(i=0; i<g_l-1; i++)
            temp[i] = temp[i+1];
        if(j == f_l+g_l-1 && pos == 0) {
            j = f_l;
            for(i=0; i<g_l-1; i++)
                f[j++] = temp[i];
            f[j] = '\0';
            printf("\nThe crc is: ");
            for(int k=0; k<g_l-1; k++)
                printf("%c", temp[k]);
            return;
        }
        if(j == f_l && pos == 1) {
            for(i=0; i<g_l-1; i++)
                if(temp[i] != '0') {
                    printf("\nError\nThe remainder is ");
                    for(int k=0; k<g_l-1; k++)
                        printf("%c", temp[k]);
                    printf(" instead of 0's");
                    return;
                }
            printf("\nData Transferred Successfully!!!");
            return;
        }
        temp[g_l-1] = f[j];
        temp[g_l] = '\0';
    }
    j++;
}
return;

```

```

}

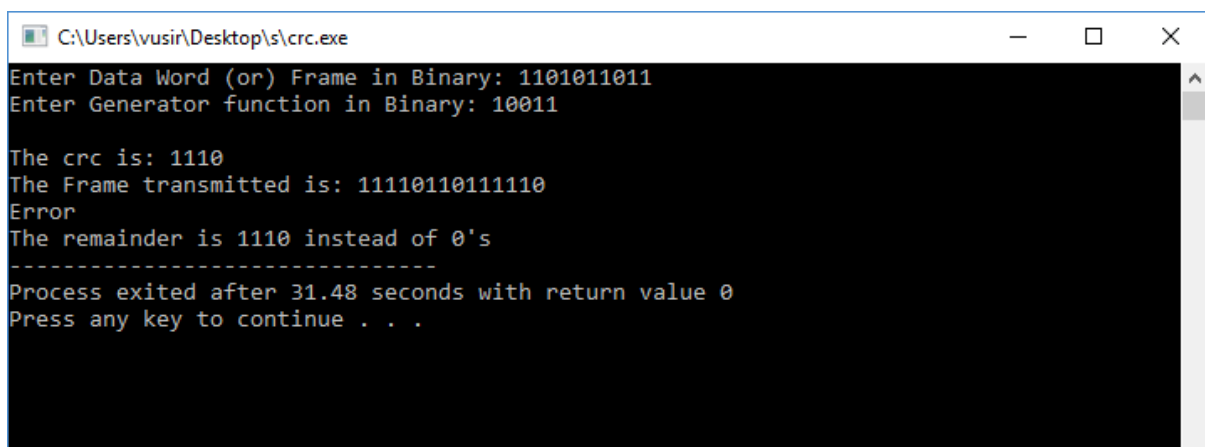
void crc_encode(char * f, char * g) {
    crc_code(f, g, 0);
}

void crc_decode(char * f, char * g) {
    crc_code(f, g, 1);
}

int main() {
    char * f = (char*)malloc(100);
    char * g = (char*)malloc(100);
    printf("Enter Data Word (or) Frame in Binary: ");
    scanf("%s", f);
    printf("Enter Generator function in Binary: ");
    scanf("%s", g);
    crc_encode(f, g);
    f[2] = '1';
    printf("\nThe Frame transmitted is: %s", f);
    crc_decode(f, g);
    return 0;
}

```

## OUTPUT:



```

C:\Users\vusir\Desktop\s\crc.exe
Enter Data Word (or) Frame in Binary: 1101011011
Enter Generator function in Binary: 10011

The crc is: 1110
The Frame transmitted is: 11110110111110
Error
The remainder is 1110 instead of 0's
-----
Process exited after 31.48 seconds with return value 0
Press any key to continue . . .

```

## WEEK-8 Implementation of Dijkstra's Algorithm

Dijkstra's algorithm is a greedy algorithm that solves the shortest path problem for a directed graph  $G$ . Dijkstra's algorithm solves the single-source shortest-path problem when all edges have non-negative weights.

### Dijkstra's Algorithm

1. DIJKSTRA( $G, s$ )
  2. INITIALIZE-SINGLE-SOURCE( $G, S$ )
  3.  $S \leftarrow \emptyset$
  4.  $Q \leftarrow V[G]$
  5. while  $Q \neq \emptyset$ 
    6. do  $u \leftarrow \text{EXTRACT-MIN}(Q)$
    7.  $S \leftarrow S \cup \{u\}$
    8. for each vertex  $v \in \text{Adj}[u]$
    9. do if  $\text{dist}[v] > \text{dist}[u] + w(u, v)$
    10. then  $d[v] \leftarrow d[u] + w(u, v)$
- 
1. INITIALIZE-SINGLE-SOURCE( Graph  $g$ , Node  $s$  )
  2.  $\text{dist}[s] = 0$ ;
  3. for each vertex  $v$  in Vertices  $V[G] - s$
  4.  $\text{dist}[v] \leftarrow \infty$

### SOURCE CODE

```
#include<stdio.h>

#include<conio.h>

#define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()
{
    int G[MAX][MAX],i,j,n,u;
    printf("Enter no. of vertices:");
    scanf("%d",&n);
    printf("\nEnter the adjacency matrix:\n");
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%d",&G[i][j]);
    printf("\nEnter the starting node:");
```

```

scanf("%d",&u);
dijkstra(G,n,u);
return 0;
}

void dijkstra(int G[MAX][MAX],int n,int startnode)
{

    int cost[MAX][MAX],distance[MAX],pred[MAX];
    int visited[MAX],count,mindistance,nextnode,i,j;
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            if(G[i][j]==0)
                cost[i][j]=INFINITY;
            else
                cost[i][j]=G[i][j];

    for(i=0;i<n;i++)
    {
        distance[i]=cost[startnode][i];
        pred[i]=startnode;
        visited[i]=0;
    }
    distance[startnode]=0;
    visited[startnode]=1;
    count=1;
    while(count<n-1)
    {
        mindistance=INFINITY;
        for(i=0;i<n;i++)
            if(distance[i]<mindistance&&!visited[i])
            {
                mindistance=distance[i];
                nextnode=i;
            }
    }
}

```

```

    }
    visited[nextnode]=1;
    for(i=0;i<n;i++)
        if(!visited[i])
            if(mindistance+cost[nextnode][i]<distance[i])
            {
                distance[i]=mindistance+cost[nextnode][i];
                pred[i]=nextnode;
            }

    count++;
}
for(i=0;i<n;i++)
    if(i!=startnode)
    {
        printf("\nDistance of node%d=%d",i,distance[i]);
        printf("\nPath=%d",i);
        j=i;
        do
        {
            j=pred[j];
            printf("<-%d",j);
        }while(j!=startnode);
    }
}

```

```

Enter the no. of vertices:: 4
Enter the adjacency matrix::
0 1 1 1
1 0 1 0
1 1 0 1
1 0 1 0
Enter the starting node:: 1
Distance of 0 = 1
Path = 0<-1
Distance of 2 = 1
Path = 2<-1
Distance of 3 = 2
Path = 3<-0<-1

```

## WEEK-9 Configure A Network Using Distance Vector Routing Algorithm.

In computer communication theory relating to packet-switched networks, a distance-vector routing protocol is one of the two major classes of intradomain routing protocols, the other major class being the link-state protocol. Distance-vector routing protocols use the Bellman-Ford algorithm, Ford–Fulkerson algorithm, or DUAL FSM (in the case of Cisco System's protocols) to calculate paths.

A distance-vector routing protocol requires that a router inform its neighbors of topology changes periodically. Compared to link-state protocols, which require a router to inform all the nodes in a network of topology changes, distance-vector routing protocols have less computational complexity and message overhead. [citation needed]

The term *distance vector* refers to the fact that the protocol manipulates *vectors* (arrays) of distances to other nodes in the network. The distance vector algorithm was the original ARPANET routing algorithm and was also used on the Internet under the name of RIP (Routing Information Protocol).

Examples of distance-vector routing protocols include RIPv1 and RIPv2, IGRP, EIGRP, and Babel.

Distance vector routing protocols, such as RIP, are susceptible to a convergence problem known as the count-to-infinity problem. This problem is a consequence of the fact that distance vector routing protocols exchange routing information only with their neighbors. Here, it may happen that, after the failure of a link, information about routes that use the failed link are propagated a long time after the failure has occurred. This results in a slow convergence of the routing tables. Each time the router exchange RIP packets, the cost of a path that uses the failed link increases, but it takes a long time until all routers realize that routes through the failed link is unavailable.

The goal of this part of the lab is to observe the count-to-infinity problem. RIP has a number of protocol features that try to avoid the count-to-infinity problem. These features will be disabled. Still, since the count-to-infinity problem requires that routing updates occur in a certain order, the count-to-infinity problem is not always observable



**SOURCE CODE:**

```
#include<stdio.h>

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

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

    do
    {
        count=0;

        for(i=0;i<nodes;i++)//We choose arbitrary vertex k and we calculate the direct
        distance from the node i to k using the cost matrix

        for(j=0;j<nodes;j++)

        for(k=0;k<nodes;k++)
```

```

        if(rt[i].dist[j]>costmat[i][k]+rt[k].dist[j])
        {
            //We calculate the minimum distance
            rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
            rt[i].from[j]=k;
            count++;
        }break;
    }while(count!=0);
    for(i=0;i<nodes;i++)
    {
        printf("\n\n For router %d\n",i+1);
        for(j=0;j<nodes;j++)
        {
            printf("\tnode %d via %d Distance %d ",j+1, rt[i].from[j]+1, rt[i].dist[j]);
        }
    }

    printf("the finalmatrix is :");
    printf("\n");
    for(i=0;i<nodes;i++)
    {
        for(j=0;j<nodes;j++)
        {
            printf("\t %d",rt[i].dist[j]);
        }
        printf("\n");
    }

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

```

## OUTPUT:

```
Enter the number of nodes : 2

Enter the cost matrix :
0 6
3 0

For router 1
node 1 via 1 Distance 0
node 2 via 2 Distance 6

For router 2
node 1 via 1 Distance 3
node 2 via 2 Distance 0 the finalmatrix is :
    0    6
    3    0
```

```
Enter the number of nodes : 3

Enter the cost matrix :
0 2 6
1 0 3
5 4 0

For router 1
node 1 via 1 Distance 0
node 2 via 2 Distance 2
node 3 via 2 Distance 5

For router 2
node 1 via 1 Distance 1
node 2 via 2 Distance 0
node 3 via 3 Distance 3

For router 3
node 1 via 1 Distance 5
node 2 via 2 Distance 4
node 3 via 3 Distance 0 the finalmatrix is :
    0    2    5
    1    0    3
    5    4    0
```

## **WEEK-10 Create Sub-Network And Assign Addresses Based On The Number Of Hosts Connected To The Network.**

- **Address** - The unique number ID assigned to one host or interface in a network.
- **Subnet** - A portion of a network that shares a particular subnet address.
- **Subnet mask** - A 32-bit combination used to describe which portion of an address refers to the subnet and which part refers to the host.
- **Interface** - A network connection.

If you have already received your legitimate address(es) from the Internet Network Information Center (InterNIC), you are ready to begin. If you do not plan to connect to the Internet, Cisco strongly suggests that you use reserved addresses from RFC 1918 .

### **Understand IP Addresses**

An IP address is an address used in order to uniquely identify a device on an IP network. The address is made up of 32 binary bits, which can be divisible into a network portion and host portion with the help of a subnet mask. The 32 binary bits are broken into four octets (1 octet = 8 bits). Each octet is converted to decimal and separated by a period (dot). For this reason, an IP address is said to be expressed in dotted decimal format (for example, 172.16.81.100). The value in each octet ranges from 0 to 255 decimal, or 00000000 - 11111111 binary.

Here is how binary octets convert to decimal: The right most bit, or least significant bit, of an octet holds a value of 1. The bit just to the left of that holds a value of 2. This continues until the left-most bit, or most significant bit, which holds a value of 128. So if all binary bits are a one, the decimal equivalent would be 255 as shown here:

1 1 1 1 1 1 1 1

128 64 32 16 8 4 2 1 (128+64+32+16+8+4+2+1=255)

Here is a sample octet conversion when not all of the bits are set to 1.

0 1 0 0 0 0 0 1

0 64 0 0 0 0 0 1 (0+64+0+0+0+0+0+1=65)

And this sample shows an IP address represented in both binary and decimal.

10. 1. 23. 19 (decimal)

00001010.00000001.00010111.00010011 (binary)

These octets are broken down to provide an addressing scheme that can accommodate large and small networks. There are five different classes of networks, A to E. This document focuses on classes A to C, since classes D and E are reserved and discussion of them is beyond the scope of this document.

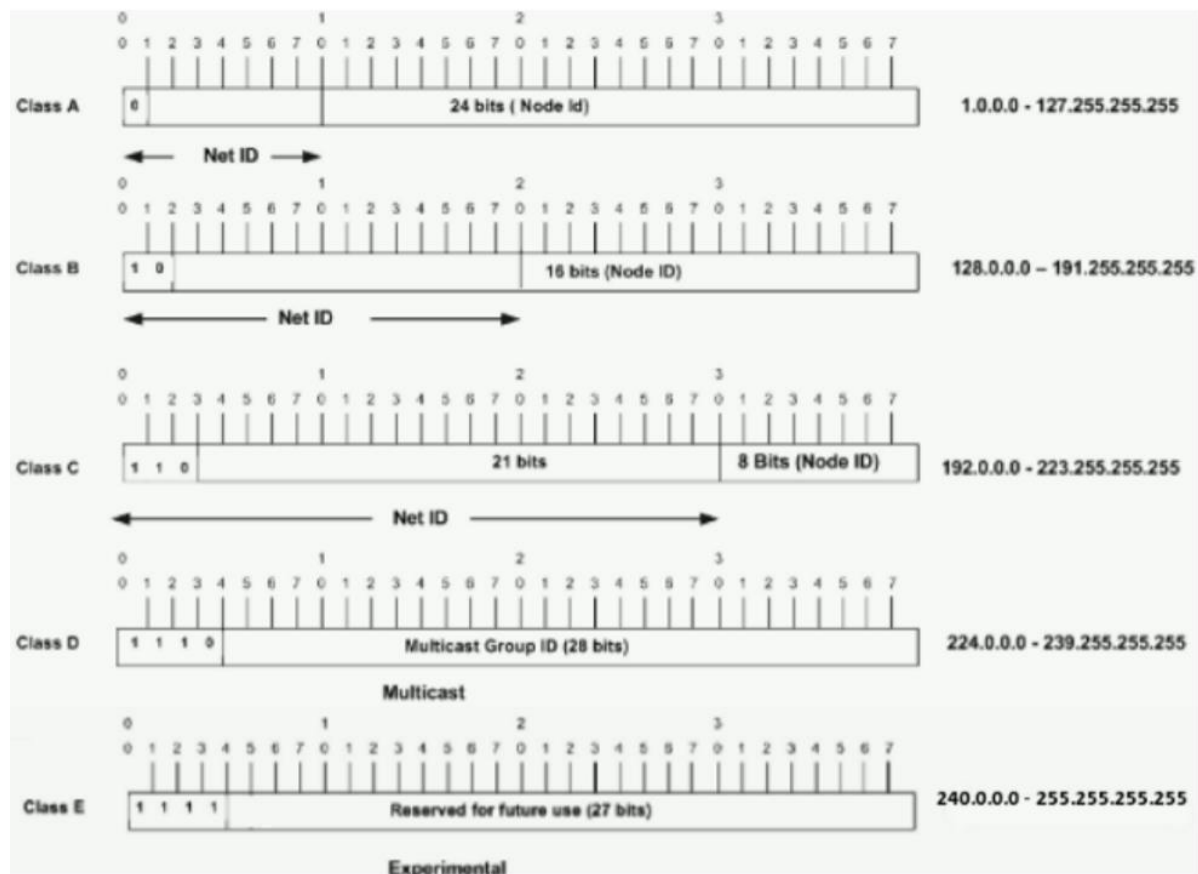
**Note:** Also note that the terms "Class A, Class B" and so on are used in this document in order to help facilitate the understanding of IP addressing and subnetting. These terms are rarely used in the industry anymore because of the introduction of classless interdomain routing (CIDR).

Given an IP address, its class can be determined from the three high-order bits (the three left-most bits in the first octet). Figure 1 shows the significance in the three high order bits

and the range of addresses that fall into each class. For informational purposes, Class D and Class E addresses are also shown.

In a Class A address, the first octet is the network portion, so the Class A example in Figure 1 has a major network address of 1.0.0.0 - 127.255.255.255. Octets 2, 3, and 4 (the next 24 bits) are for the network manager to divide into subnets and hosts as he/she sees fit. Class A addresses are used for networks that have more than 65,536 hosts (actually, up to 16777214 hosts!). In a Class B address, the first two octets are the network portion, so the Class B example in Figure 1 has a major network address of 128.0.0.0 - 191.255.255.255. Octets 3 and 4 (16 bits) are for local subnets and hosts. Class B addresses are used for networks that have between 256 and 65534 hosts. In a Class C address, the first three octets are the network portion. The Class C example in Figure 1 has a major network address of 192.0.0.0 - 223.255.255.255. Octet 4 (8 bits) is for local subnets and hosts - perfect for networks with less than 254 hosts.

**Figure 1**



### Network Masks

A network mask helps you know which portion of the address identifies the network and which portion of the address identifies the node. Class A, B, and C networks have default masks, also known as natural masks, as shown here:

Class A: 255.0.0.0

Class B: 255.255.0.0

Class C: 255.255.255.0

An IP address on a Class A network that has not been subnetted would have an address/mask pair similar to: 8.20.15.1 255.0.0.0. In order to see how the mask helps you identify the network and node parts of the address, convert the address and mask to binary numbers.

8.20.15.1 = 00001000.00010100.00001111.00000001

255.0.0.0 = 11111111.00000000.00000000.00000000

Once you have the address and the mask represented in binary, then identification of the network and host ID is easier. Any address bits which have corresponding mask bits set to 1 represent the network ID. Any address bits that have corresponding mask bits set to 0 represent the node ID.

8.20.15.1 = 00001000.00010100.00001111.00000001

255.0.0.0 = 11111111.00000000.00000000.00000000

-----

net id | host id

netid = 00001000 = 8

hostid = 00010100.00001111.00000001 = 20.15.1

### Understand Subnetting

Subnetting allows you to create multiple logical networks that exist within a single Class A, B, or C network. If you do not subnet, you are only able to use one network from your Class A, B, or C network, which is unrealistic.

Each data link on a network must have a unique network ID, with every node on that link being a member of the same network. If you break a major network (Class A, B, or C) into smaller subnetworks, it allows you to create a network of interconnecting subnetworks. Each data link on this network would then have a unique network/subnetwork ID. Any device, or gateway, that connects n networks/subnetworks has n distinct IP addresses, one for each network / subnetwork that it interconnects. In order to subnet a network, extend the natural mask with some of the bits from the host ID portion of the address in order to create a subnetwork ID. For example, given a Class C network of 204.17.5.0 which has a natural mask of 255.255.255.0, you can create subnets in this manner:

204.17.5.0 - 11001100.00010001.00000101.00000000

255.255.255.224 - 11111111.11111111.11111111.11100000

-----|sub|----

By extending the mask to be 255.255.255.224, you have taken three bits (indicated by "sub") from the original host portion of the address and used them to make subnets. With these three bits, it is possible to create eight subnets. With the

remaining five host ID bits, each subnet can have up to 32 host addresses, 30 of which can actually be assigned to a device since host IDs of all zeros or all ones are not allowed (it is very important to remember this). So, with this in mind, these subnets have been created.

204.17.5.0 255.255.255.224 host address range 1 to 30

204.17.5.32 255.255.255.224 host address range 33 to 62

204.17.5.64 255.255.255.224 host address range 65 to 94

204.17.5.96 255.255.255.224 host address range 97 to 126

204.17.5.128 255.255.255.224 host address range 129 to 158

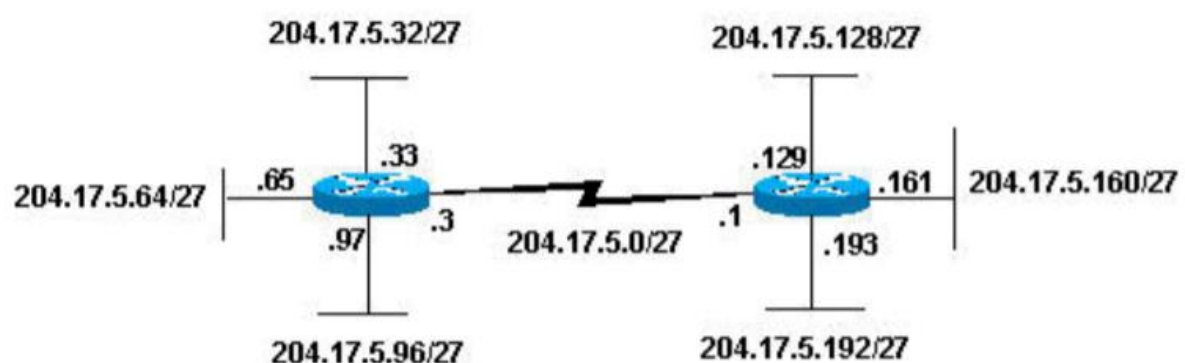
204.17.5.160 255.255.255.224 host address range 161 to 190

204.17.5.192 255.255.255.224 host address range 193 to 222

204.17.5.224 255.255.255.224 host address range 225 to 254

Note: There are two ways to denote these masks. First, since you use three bits more than the "natural" Class C mask, you can denote these addresses as having a 3-bit subnet mask. Or, secondly, the mask of 255.255.255.224 can also be denoted as /27 as there are 27 bits that are set in the mask. This second method is used with CIDR. With this method, one of these networks can be described with the notation prefix/length. For example, 204.17.5.32/27 denotes the network 204.17.5.32 255.255.255.224. When appropriate, the prefix/length notation is used to denote the mask throughout the rest of this document.

The network subnetting scheme in this section allows for eight subnets, and the network might appear as:



Notice that each of the routers in Figure 2 is attached to four subnetworks, one subnetwork is common to

both routers. Also, each router has an IP address for each subnetwork to which it is attached. Each

subnetwork could potentially support up to 30 host addresses.

This brings up an interesting point. The more host bits you use for a subnet mask, the more subnets you

have available. However, the more subnets available, the less host addresses available per subnet. For

example, a Class C network of 204.17.5.0 and a mask of 255.255.255.224 (/27) allows you to have eight

subnets, each with 32 host addresses (30 of which could be assigned to devices). If you use a mask of

255.255.255.240 (/28), the break down is:

204.17.5.0 - 11001100.00010001.00000101.00000000

255.255.255.240 - 11111111.11111111.11111111.11110000

-----|sub |---

Since you now have four bits to make subnets with, you only have four bits left for host addresses. So in this case you can have up to 16 subnets, each of which can have up to 16 host addresses (14 of which can be assigned to devices). Take a look at how a Class B network might be subnetted. If you have network 172.16.0.0, then you know that its natural mask is 255.255.0.0 or 172.16.0.0/16. Extending the mask to anything beyond 255.255.0.0 means you are subnetting. You can quickly see that you have the ability to create a lot more subnets than with the Class C network. If you use a mask of 255.255.248.0 (/21), how many subnets and hosts per subnet does this allow for?

172.16.0.0 - 10101100.00010000.00000000.00000000

255.255.248.0 - 11111111.11111111.11111000.00000000

-----| sub |-----

You use five bits from the original host bits for subnets. This allows you to have 32 subnets (25). After using the five bits for subnetting, you are left with 11 bits for host addresses. This allows each subnet so have 2048 host addresses (211), 2046 of which could be assigned to devices. Note: In the past, there were limitations to the use of a subnet 0 (all subnet bits are set to zero) and all ones subnet (all subnet bits set to one). Some devices would not allow the use of these subnets. Cisco Systems devices allow the use of these subnets when the ip subnet zero command is configured



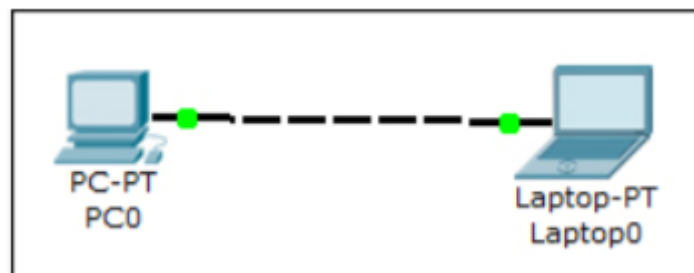
## WEEK-11 Transfer Files From Pc To Pc Using Packet Tracer Software

Creating a simple topology

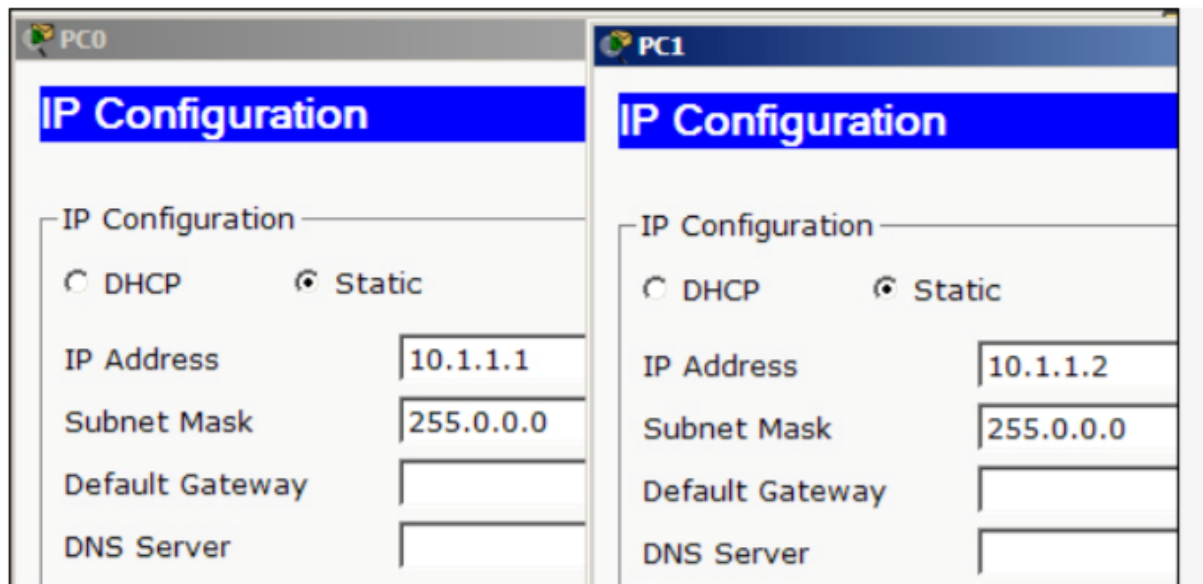
After creating, you can check how packets are moving from one PC to other PC.

Now that you're familiar with the GUI of Packet Tracer, you can create your first network topology by carrying out the following steps:

1. From the network component box, click on End Devices and drag-and-drop a Generic PC icon and a Generic laptop icon into the Workspace.
2. Click on Connections, then click on Copper Cross-Over, then on PC0, and select Fast Ethernet. After this, click on Laptop0 and select FastEthernet. The link status LED should show up in green, indicating that the link is up.



3. Click on the PC, go to the Desktop tab, click on IP Configuration, and enter an IP address and subnet mask. In this topology, the default gateway and DNS server information is not needed as there are only two end devices in the network.
4. Close the window, open the laptop, and assign an IP address to it in the same way. Make sure that both of the IP addresses are in the same subnet.



5. Close the IP Configuration box, open the command prompt, and ping the IP address of the device at the end to check connectivity

```

PC>ping 10.1.1.1

Pinging 10.1.1.1 with 32 bytes of data:

Reply from 10.1.1.1: bytes=32 time=62ms TTL=128
Reply from 10.1.1.1: bytes=32 time=31ms TTL=128
Reply from 10.1.1.1: bytes=32 time=32ms TTL=128
Reply from 10.1.1.1: bytes=32 time=31ms TTL=128

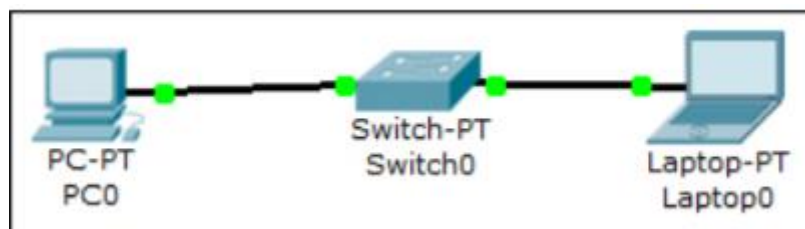
Ping statistics for 10.1.1.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 31ms, Maximum = 62ms, Average = 39ms

```

### Pinging Laptop0 from PC0

Add an Ethernet switch to this topology so that more than two end devices can be connected, by performing the following steps:

1. Click on Switches from the device-type selection box and insert any switch (except Switch-PT Empty) into the workspace.
2. Remove the link between the PC and the laptop using the delete tool from the common tools bar.
3. Choose the Copper Straight-Through cable and connect the PC and laptop with the switch. At this point, the link indicators on the switch are orange in color because the switchports are undergoing the listening and learning states of the Spanning Tree Protocol (STP).



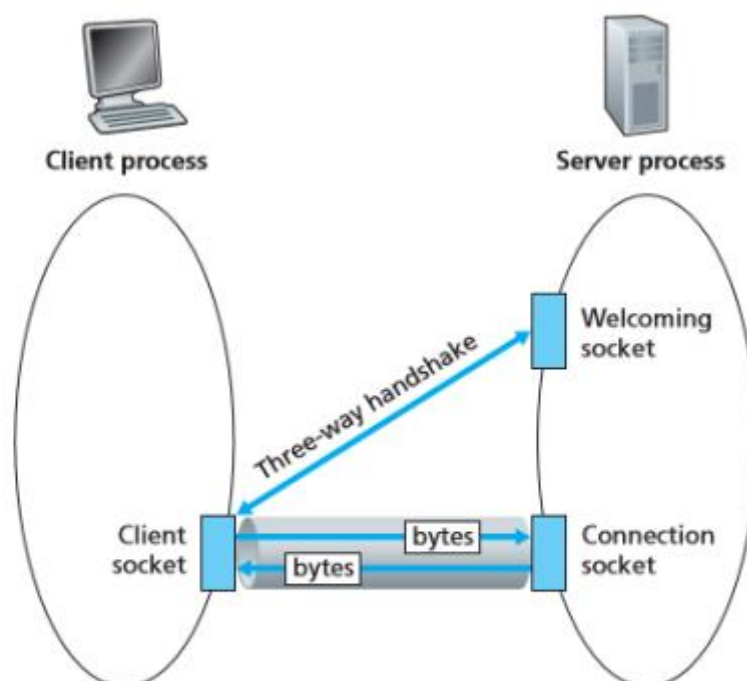
4. Once the link turns green, as shown in the previous screenshot, ping again to check the connectivity. The next chapter, Chapter 2, Network Devices, will deal with the configuration of network devices.
5. To save this topology, navigate to File | Save As and choose a location. The topology will be saved with a .pkt extension, with the devices in the same state.

## WEEK-12 Implement Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) using NS-2.

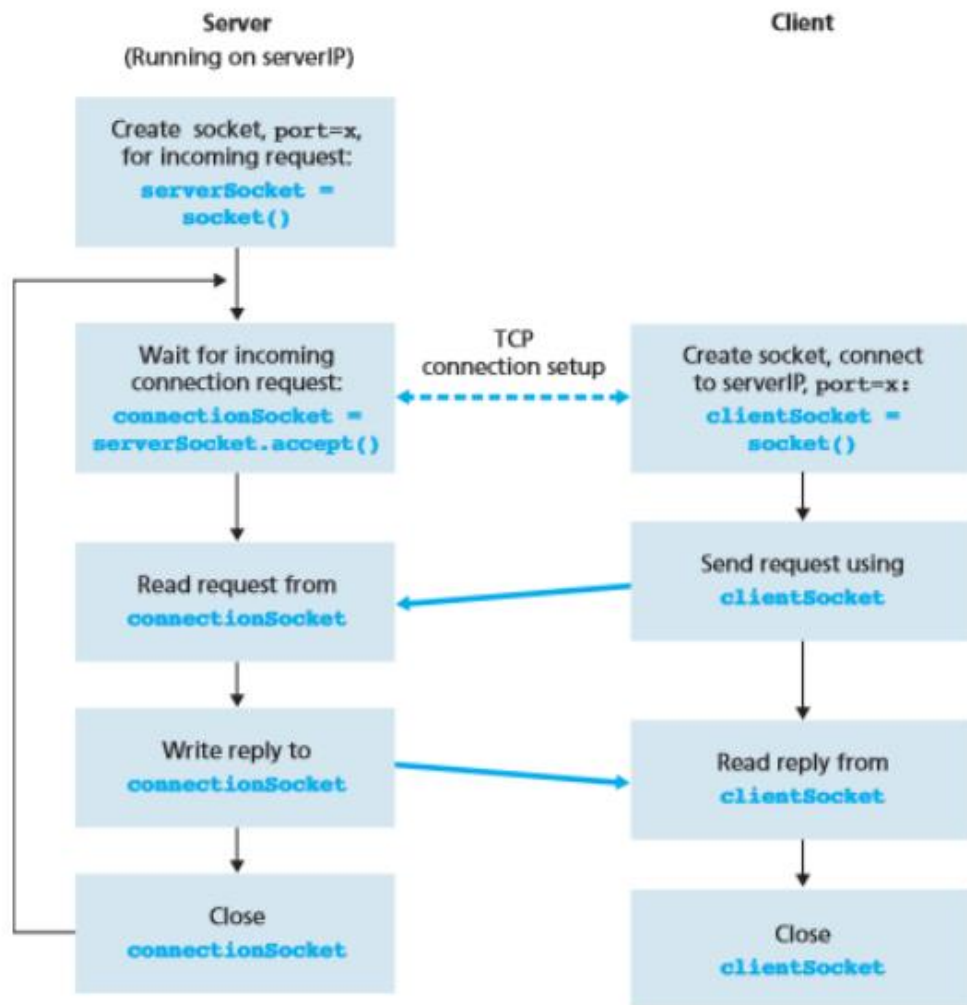
### TCP

TCP is a connection-oriented protocol. This means that before the client and server can start to send data to each other, they first need to handshake and establish a TCP connection. One end of the TCP connection is attached to the client socket and the other end is attached to a server socket. When creating the TCP connection, we associate with it the client socket address (IP address and port number) and the server socket address (IP address and port number). With the TCP connection established, when one side wants to send data to the other side, it just drops the data into the TCP connection via its socket.

With the server process running, the client process can initiate a TCP connection to the server. This is done in the client program by creating a TCP socket. When the client creates its TCP socket, it specifies the address of the welcoming socket in the server, namely, the IP address of the server host and the port number of the socket. After creating its socket, the client initiates a three-way handshake and establishes a TCP connection with the server.



**Figure 2.29** ♦ The TCPServer process has two sockets



**Figure 2.30** ♦ The client-server application using TCP

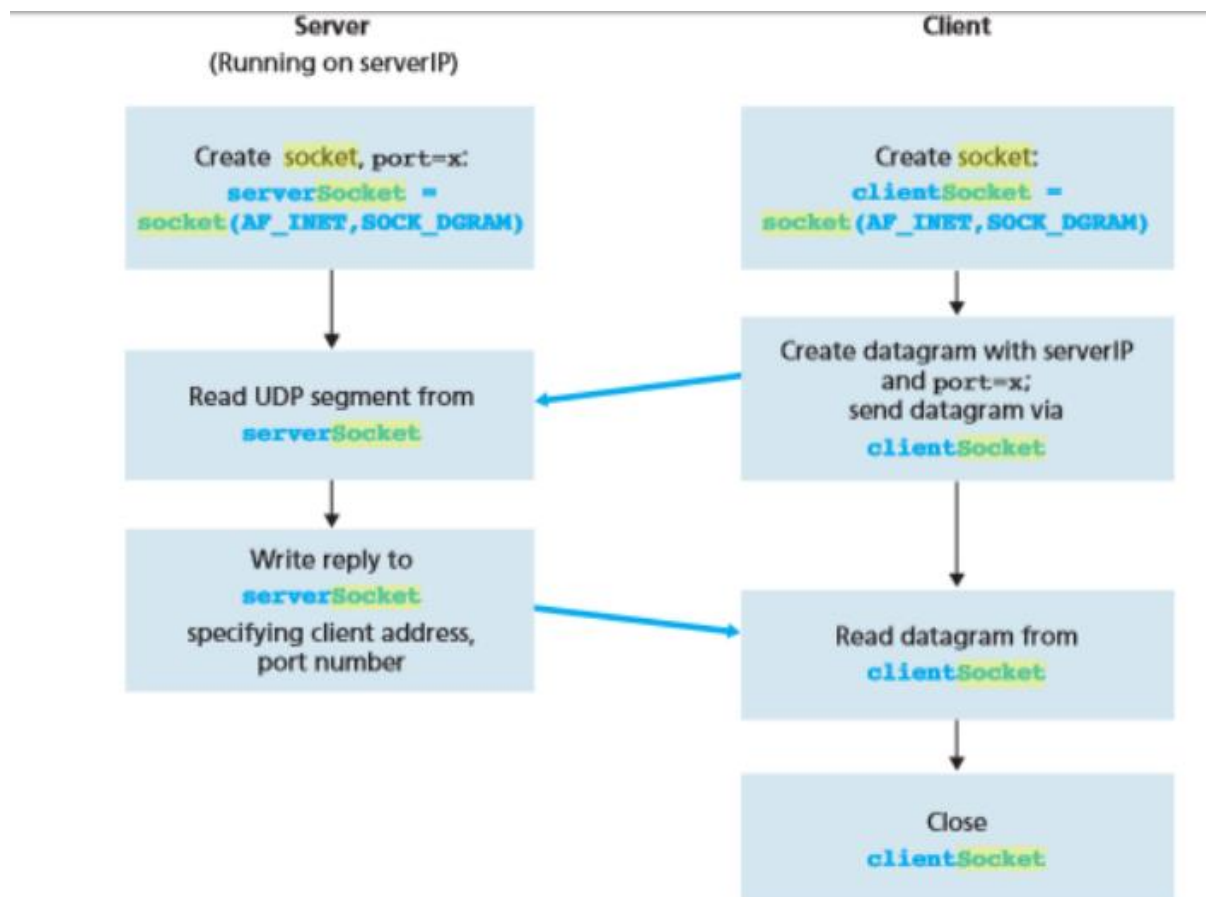
## UDP:-

Write a program on datagram socket for client/server to display the messages on client side, typed at the server side.

## DESCRIPTION:

Using User Datagram Protocol, Applications can send data/message to the other hosts without prior communications or channel or path. This means even if the destination host is not available, application can send data. There is no guarantee that the data is received in the other side. Hence it's not a reliable service.

UDP is appropriate in places where delivery of data doesn't matters during data transition.



**Figure 2.28** ♦ The client-server application using UDP

## SOURCE CODE:

### TCP for various networks.

#### 1 Simple client-server model using TCP

```

set ns [new Simulator]
set nf [open cs.nam w]
$ns namtrace-all $nf
set nd [open cs.tr w]
$ns trace-all $nd
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
  
```

```

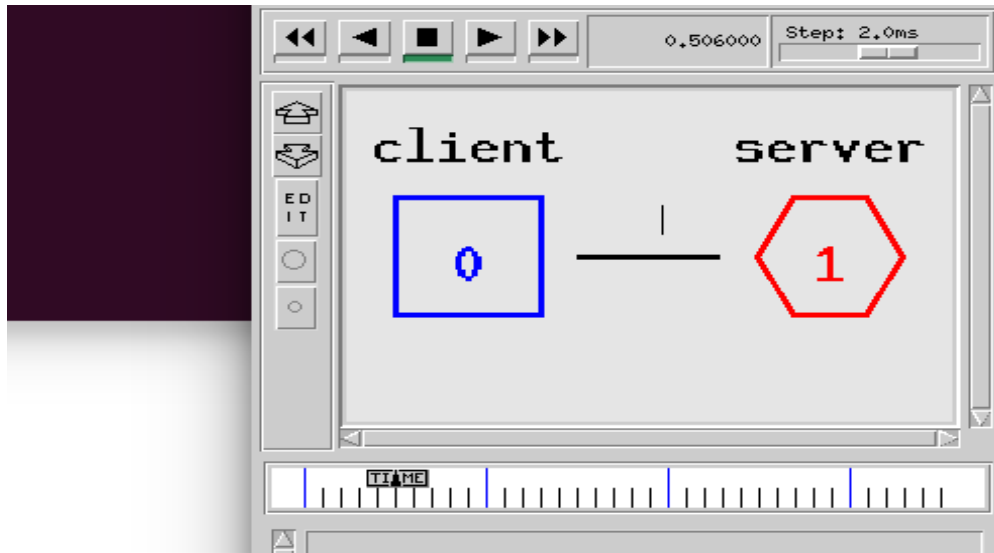
$ns duplex-link $n0 $n1 10Mbps 10ms DropTail
$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link $n1 $n2 10Mbps 10ms DropTail
$ns duplex-link-op $n1 $n2 orient right
$ns duplex-link $n2 $n3 10Mbps 10ms DropTail
$ns duplex-link-op $n2 $n3 orient right
$n0 shape box
$n1 shape hexagon
$ns at 0.1 "$n0 label client"
$ns at 0.2 "$n1 label server"
$ns at 0.1 "$n0 color blue"
$ns at 0.2 "$n1 color red"
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set Sink [new Agent/Null]
$ns attach-agent $n3 $Sink
$ns connect $tcp0 $Sink
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize _ 500
$cbr0 set interval _ 0.005
$cbr0 attach-agent $tcp0
$ns at 0.5 "$cbr0 start"
$ns at 2.5 "$cbr0 stop"
proc finish { } {
global ns nf nd
$ns flush-trace
close $nf
exec nam cs.nam &
exit 0
}

```

\$ns at 5.0 "finish"

\$ns run

## OUTPUT:



## 2 Data transfer among set of nodes:

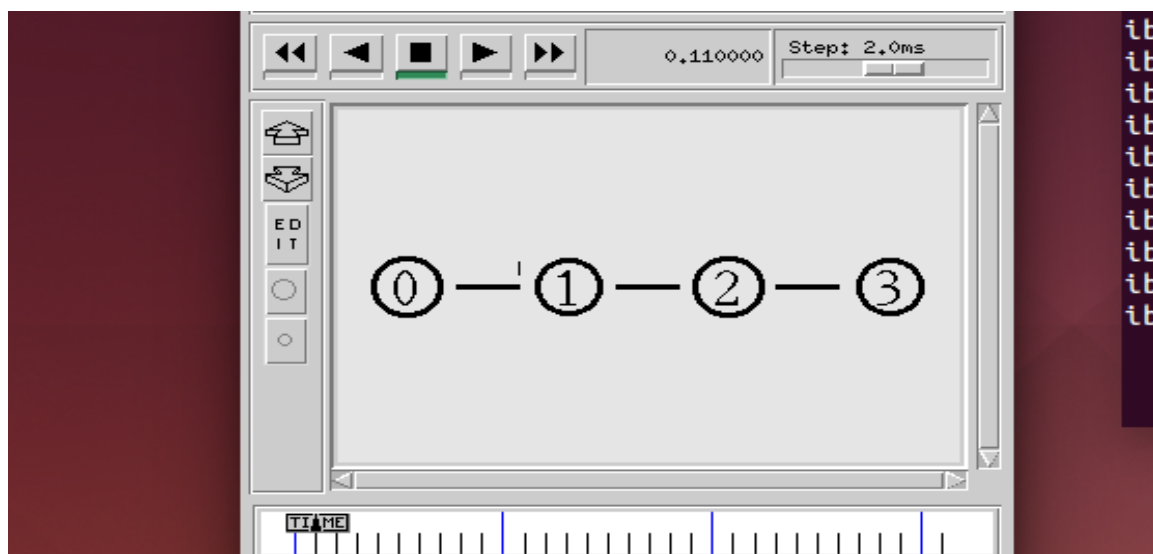
```
set ns [new Simulator]
set nf [open cse.nam w]
$ns namtrace-all $nf
set nd [open cse.tr w]
$ns trace-all $nd
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$ns duplex-link $n0 $n1 10Mbps 10ms DropTail
$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link $n1 $n2 10Mbps 10ms DropTail
$ns duplex-link-op $n1 $n2 orient right
$ns duplex-link $n2 $n3 10Mbps 10ms DropTail
$ns duplex-link-op $n2 $n3 orient right
```

```

set udp0 [new Agent/TCP]
$ns attach-agent $n0 $udp0
set Sink [new Agent/Null]
$ns attach-agent $n3 $Sink
$ns connect $udp0 $Sink
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize _ 1500
$cbr0 set interval _ 0.005
$cbr0 attach-agent $udp0
$ns at 0.1 "$cbr0 start"
$ns at 0.4 "$cbr0 stop"
proc finish { } {
    global ns nf nd
    $ns flush-trace
    close $nf
    exec nam cse.nam &
    exit 0
}
$ns at 5.0 "finish"
$ns run

```

### OUTPUT:





## UDP Programs

```
set ns [new Simulator]

set nf [open cs.nam w]

$ns namtrace-all $nf

set nd [open cs.tr w]

$ns trace-all $nd

set n0 [$ns node]

set n1 [$ns node]

$ns duplex-link $n0 $n1 10Mbps 10ms DropTail

$ns duplex-link-op $n0 $n1 orient right

$n0 shape box

$n1 shape hexagon

$ns at 0.1 "$n0 label client"

$ns at 0.2 "$n1 label server"

$ns at 0.1 "$n0 color blue"

$ns at 0.2 "$n1 color red"

set udp0 [new Agent/UDP]

$ns attach-agent $n0 $udp0

set udp1 [new Agent/Null]

$ns attach-agent $n1 $udp1

$ns connect $udp0 $udp1

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize _ 500

$cbr0 set interval _ 0.005

$cbr0 attach-agent $udp0
```

```
$ns at 0.5 "$cbr0 start"
```

```
$ns at 2.5 "$cbr0 stop"
```

```
proc finish { } {
```

```
global ns nf nd
```

```
$ns flush-trace
```

```
close $nf
```

```
exec nam cs.nam &
```

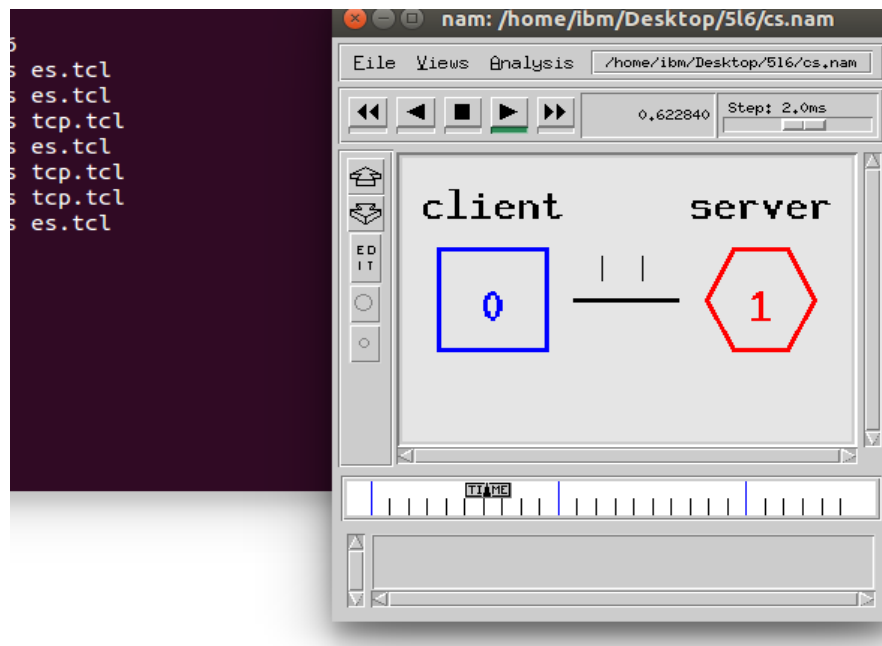
```
exit 0
```

```
}
```

```
$ns at 5.0 "finish"
```

```
$ns run
```

## OUTPUT:



```
set ns [new Simulator]
```

```
set nf [open cse.nam w]
```

```
$ns namtrace-all $nf
```

```
set nd [open cse.tr w]
```

```
$ns trace-all $nd
```

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

$ns duplex-link $n0 $n1 10Mbps 10ms DropTail
$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link $n1 $n2 10Mbps 10ms DropTail
$ns duplex-link-op $n1 $n2 orient right
$ns duplex-link $n2 $n3 10Mbps 10ms DropTail
$ns duplex-link-op $n2 $n3 orient right

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

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

$ns connect $udp0 $Sink

set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize _ 500
$cbr0 set interval _ 0.005
$cbr0 attach-agent $udp0

$ns at 0.1 "$cbr0 start"
$ns at 0.4 "$cbr0 stop"

proc finish { } {
global ns nf nd
$ns flush-trace
close $nf
exec nam cse.nam &
```

```
exit 0
```

```
}
```

```
$ns at 5.0 "finish"
```

```
$ns run
```

### OUTPUT:

