**K. L. E. SOCIETY’S**

**B. V. B. COLLEGE OF ENGINEERING &TECHNOLOGY, HUBLI – 580031**

**(An Autonomous Institution)**



**DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING**

**COMPUTER NETWORKING LAB REPORT**

Submitted by

**Akash S Bilgi**

**USN: 2BV14IS007**

**Roll num: 06**

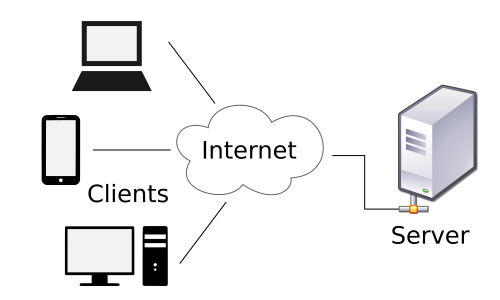
1. a) Analysis of client-server code.

b) Execute client-server code on local host.

c) Implement a basic chat application on peer machine.

**Theoretical description:-**

A client-server model is a distributed application to communicate over a computer network.



Server: A Server is an entity which passively waits for and responds for clients. It is a passive socket.

Client: A client is an entity which initiates the communication. It must know the address and the port of the server. It is an active socket.

**Algorithm**

Server side

1. Create a socket
2. Bind it to the operating system.
3. Listen over it.
4. Accept connections.
5. Receive data from client and send it back to client.
6. Close the socket.

Client side

1. Create a socket.
2. Connect to the server using connect ().
3. Send data to server and receive data from the server.
4. Close the socket.

**Code Snippet:**

**Client side**

while(1)

{

printf ("Me: ");

gets (msg);

send(sck , msg , strlen(msg) , 0);

if((recv\_size = recv(sck , server\_reply , 2000 , 0)) == SOCKET\_ERROR)

puts("recv failed");

else

{

server\_reply[recv\_size] = '\0';

puts(server\_reply);

}

}

**Sever side**

while(1)

{

if((recv\_size = recv(new\_socket , client\_reply , 2000 , 0)) == SOCKET\_ERROR)

puts("recv failed");

else

{

client\_reply[recv\_size] = '\0';

puts(client\_reply);

printf ("Me: ");

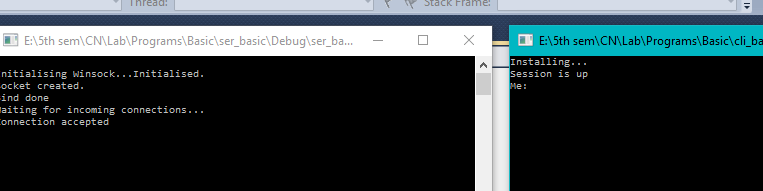
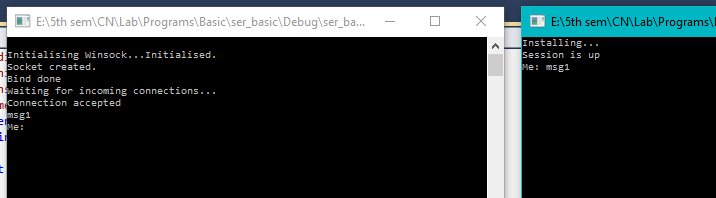
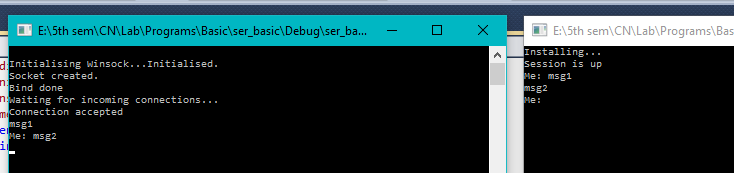
gets (msg);

send(new\_socket , msg , strlen(msg) , 0);

}

}

**Snapshot:**

1. Client-server connected. Connection initiated by client.
2. Client first sending a msg1 and that is received by the server.
3. Server then sends msg2 and that will b received by the client.
4. Implement a basic chat application on peer machine

**Algorithm**

Server side

1. Create a socket  
2. Bind it to the operating system.  
3. Listen over it.  
4. Accept connections.  
5. Receive data from client and send it back to client.  
6. Close the socket.

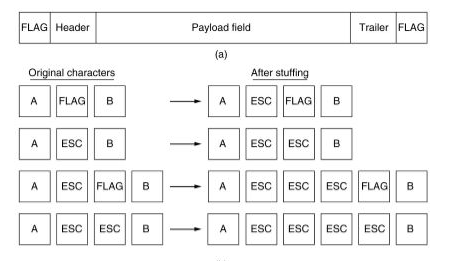
Client side

1. Create a socket.  
2. Connect to the server using connect () and enter the IP address of the peer machine.  
3. Send data to server and receive data from the server.  
4. Close the socket.

1. Write a client server program for a given input by the user to perform byte stuffing. Calculate the overhead cost.

**Theoretical description:-**

* In byte stuffing data is transmitted in terms of bytes.
* It is an algorithm for encoding data bytes that result in efficient, reliable, unambiguous packet framing regardless of packet content, thus making it easy for receiving applications to recover from malformed packets.



**Algorithm**

**Step1:** User enters an input.

**Step 2:** Adds “STX” at the beginning of the input to identify the start of the frame. Adds “ETX” at the end of the input to identify the end of the frame.

**Step 3:** Checks if the data itself contain “ETX”. If so,”DLE”/”ESC” is added to the input at the client side and is sent to server. If the data contains “DLE”/”ESC” then again “DLE”/”ESC” is prefixed to it.

**Step 4:** After receiving the input

**Code Snippet**:

Client side

while(1){

printf ("Enter your message: ");

gets (msg);

strcpy(msg1,msg);

int k=100;

for(int i=0; i<strlen(msg); i++){

if(msg[i] == 'S' && msg[i+1] == 'T' && msg[i+2] == 'X' ){

k=i;

for(int j=i; j<strlen(msg); j++){

msg[j+3] = msg1[j];

}

break;

}

}

if(k!=100){

msg[k] = 'D';

msg[k+1] = 'L';

msg[k+2] = 'E';

}

strcpy(flag1,flag);

strcat(flag1,msg);

strcat(flag1,flag2);

printf("Message after stuffing: ");

puts(flag1);

send(sck , flag1 , strlen(flag1) , 0);

}

Server side

while(1){

if((recv\_size = recv(new\_socket , client\_reply , 2000 , 0)) == SOCKET\_ERROR)

puts("recv failed");

else{

client\_reply[recv\_size] = '\0';

printf("Recieved message from the client: ");

puts(client\_reply);

int c =0;

while(c < (strlen(client\_reply) - 6)){

cli\_msg[c] = client\_reply[c+3];

c++;}

cli\_msg[c]='\0';

strcpy(cli\_msg1,cli\_msg);

for(int i=0; i< strlen(cli\_msg); i++){

if(cli\_msg[i] == 'D' && cli\_msg[i+1] == 'L' && cli\_msg[i+2] == 'E'){

for(int j=i; j<strlen(cli\_msg); j++) {

cli\_msg[j] = cli\_msg1[j+3]; }

break; }}

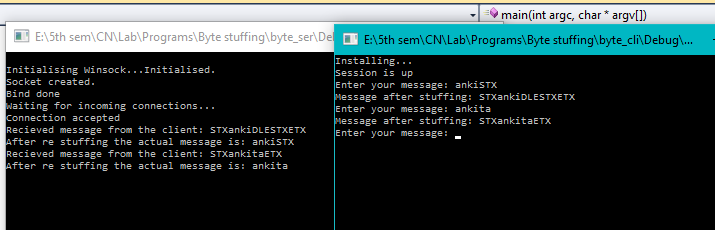
printf("After re stuffing the actual message is: ");

puts(cli\_msg);

}

}

**Snapshot**

****

1. Write a client-server program for the given standard polynomial used for the internet and detect how many numbers of bits is corrupted.

**Theoretical description:-**

A cyclic redundancy check (CRC) is an error-detecting code. Blocks of data entering these systems get a short *check value* attached, based on the remainder of a polynomial division of their contents. On retrieval, the calculation is repeated and, in the event the check values do not match, corrective action can be taken against data corruption.

Standard polynomials are

* + CRC-8 = x8+x2+x+1
  + CRC-10 = x10+x9+x5+x4+x+1
  + CRC-12 = x12+x11+x3+x2+x+1
  + CRC-16 = x16+x15+x2+1
  + CRC-CCITT = x16+x12+x5+1
  + CRC-32 = x32+x26+x23+x22+x16+x12+x11+x10+x8+x7+x5+x4+x2+x+1

**Algorithm:-**

**Step 1:** Choose one standard polynomial from the above mentioned polynomials.

**Step 2:** Enter the data / message(dividend)

**Step 3:** Divide the message by the divisor (standard polynomial)

**Step 4:** Quotient is ignored and the remainder is added to the message and sent.

**Step 5:** In server side, the concatenated message is received and is divided. If the remainder is zero, there is no error during transmission else there was an error during transmission.

**Code Snippet:**

**Client side**

while(1){

{

printf("Enter your choice\n");

printf("1-->CRC-32\n");

printf("2-->CRC-16\n");

printf("3-->CRC-12\n");

printf("4-->CRC-10\n");

printf("5-->CRC-8\n");

int ch;

scanf("%d",&ch);

switch(ch)

{

case 1: strcpy(g,"100000100110000010001110110111");

break;

case 2: strcpy(g,"11000000000000101");

break;

case 3: strcpy(g,"1100000001111");

break;

case 4: strcpy(g,"11000110011");

break;

case 5: strcpy(g,"100000111");

break;

}

//scanf("%s",g);

printf("\n%s\n",g);

printf("\nEnter the message\n");

scanf("%s",user);

strcpy(t,user);

}

crc();

a1[0]=strlen(t);

for(e=a1[0];e<a1[0]+a1[0]-N;e++)

t[e]=cs[e-a1[0]];

printf("\nFinal codeword is(after performing xor operation i.e appending with remainder) :\n%s",t);

printf("\n");

strcpy(msg,t);

send(sck , user , strlen(user) , 0);

}

**Server side**

while(1)

{

if((recv\_size = recv(new\_socket , client\_reply , 2000 , 0)) == SOCKET\_ERROR)

puts("recv failed");

else

{

client\_reply[recv\_size] = '\0';

printf ("message is : ");

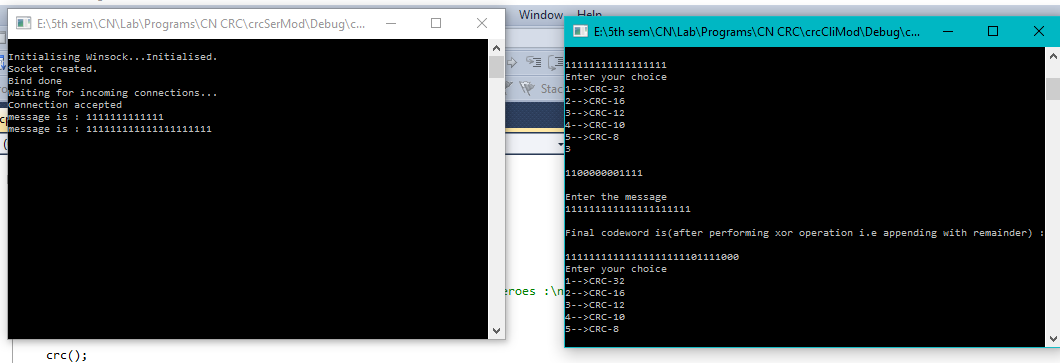
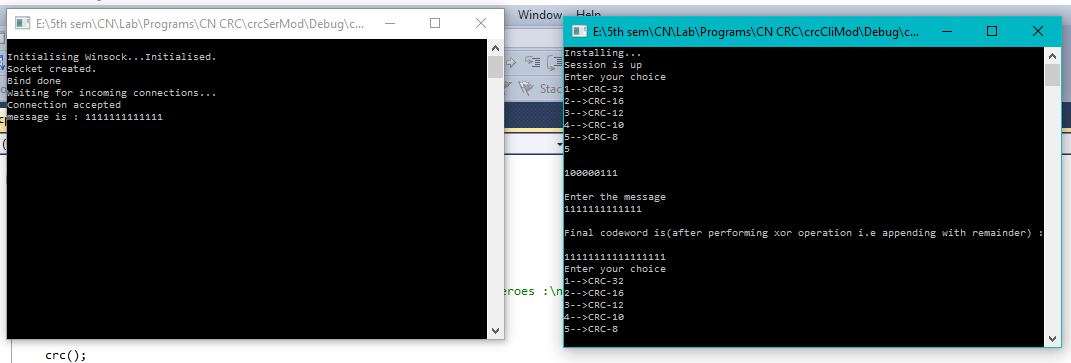
cout<<client\_reply<<endl;

send(new\_socket , msg , strlen(msg) , 0);

}

}

**Snapshot**



1. Create a client-server environment for realizing flow control at data link layer, where receiver does not have negative acknowledgement feature. Declare the entire variable at both sender and receiver w.r.t sliding window in line with textbook referred during the course. Program should allow user to study abnormal situations.

**Theoretical description:**

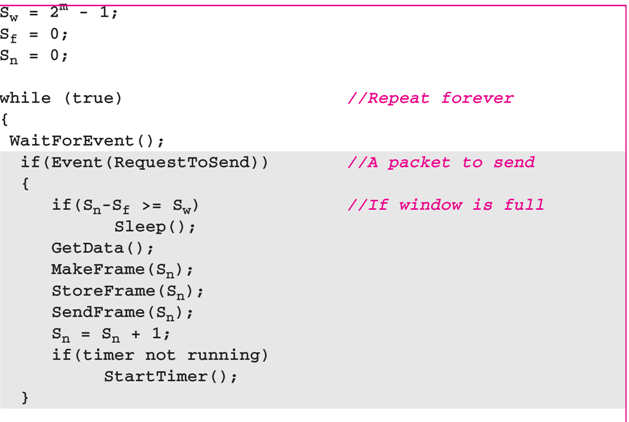
Go back N protocol:

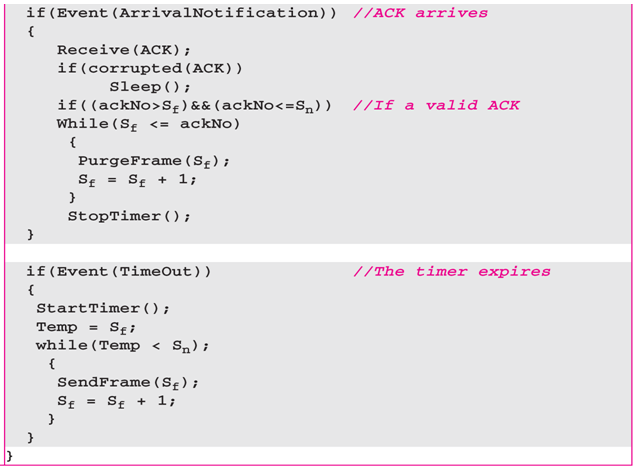
In the Go-Back-N Protocol, the sequence numbers are modulo 2*m*, where m is the size of the sequence number field in bits. The sender window is an abstract concept defining an imaginary box of size 2*m* – 1. The sender window can slide one or more slots when a valid acknowledgment arrives. The receiver window is an abstract concept defining an imaginary box of size 1 with one single variable Rn. The window slides, when a correct frame has arrived; sliding occurs one slot at a time. In Go-Back-N ARQ, the size of the sender window must be less than 2*m*; the size of the receiver window is always 1.

**Algorithm:**

**Go-Back-N receiver algorithm**



** Go-Back-N sender algorithm**



**Code Snippet:**

**Client side**

while(1)

{

printf ("Enter data: ");

gets (msg);

printf("Enter window size:");

scanf("%d",&size);

while((c= getchar()) != '\n' && c != EOF);

if(strlen(msg)<size)

{

for(int i=0; i<strlen(msg); i=i+size)

{

strncpy(msg1,msg + i,i+size);

msg1[size] = '\0';

begin = clock();

send(sck , msg1 , strlen(msg1) , 0);

if((recv\_size = recv(sck , server\_reply , 2000 , 0)) == SOCKET\_ERROR)

puts("recv failed");

else

{

end=clock();

int c = end - begin;

if(c>100)

{

printf("Acknowledgement not recieved in time\n");

printf("Resending frame");

send(sck , msg1 , strlen(msg1) , 0);

if((recv\_size = recv(sck , server\_reply , 2000 , 0)))

puts("recv failed");

else

{

server\_reply[recv\_size] = '\0';

puts(server\_reply);

}

}

else

{

server\_reply[recv\_size] = '\0';

puts(server\_reply);

}

}

}

}

}

**Server side**

while(1)

{

char msg1 [10] = {'A','C','K','\0'};

if((recv\_size = recv(new\_socket , client\_reply , 2000 , 0)) == SOCKET\_ERROR)

puts("recv failed");

else

{

client\_reply[recv\_size] = '\0';

puts(client\_reply);

strcat(msg1,client\_reply);

strcpy(msg,msg1);

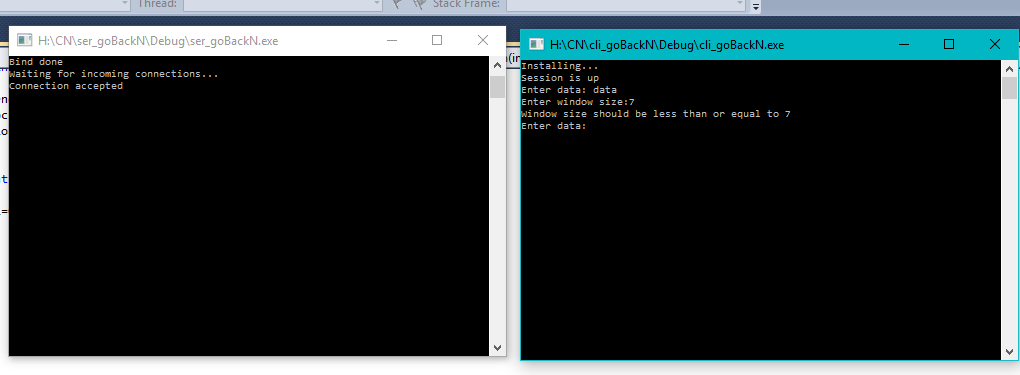
send(new\_socket , msg1 , strlen(msg1) , 0);

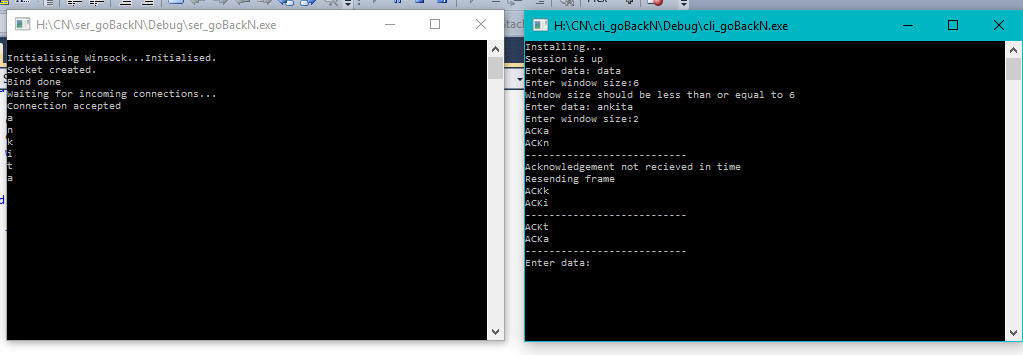
}

}

**Snapshots**

If window size is greater than the data the message wont be sent.





The 3rd frame is sent after a delay. So that frame is resent again.

1. Create a client-server environment to realize access control when CSMA/CD with p persistent method is used. Generate the report of waiting time, travelling time, number of success transmission and failed transmission by assuming appropriate number of frames to be transmitted from sender.

**Theoretical Description**

Carrier sense multiple access with collision detection (CSMA/CD) is a media access control method used most notably in early Ethernet technology for local area networking. It uses a carrier sensing scheme in which a transmitting station detects collisions by sensing transmissions from other stations while transmitting a frame. When this collision condition is detected, the station stops transmitting that frame, transmits a jam signal, and then waits for a random time interval before trying to resend the frame.

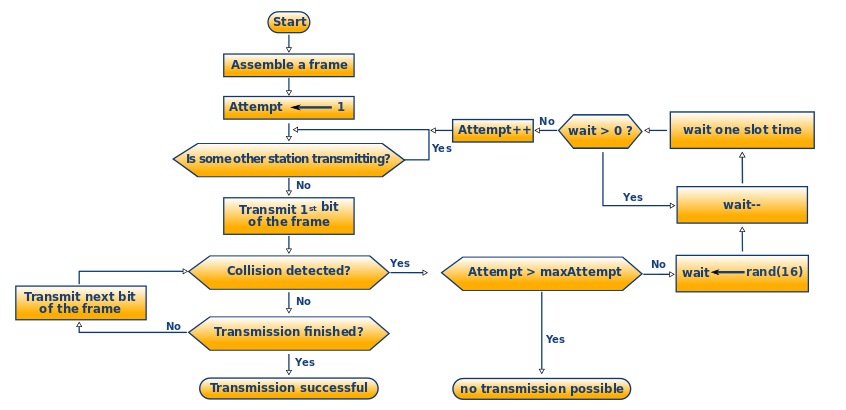
**Algorithm**

**Procedure to initiate a transmission**

1. Is my frame ready for transmission? If yes, it goes on to the next point.
2. Is medium idle? If not, wait until it becomes ready.
3. Start transmitting and monitor for collision during transmission.
4. Did a collision occur? If so, go to collision detected procedure.
5. Reset retransmission counters and end frame transmission.

**Collision detected procedure**

1. Continue transmission (with a jam signal instead of frame header/data/CRC) until minimum packet time is reached to ensure that all receivers detect the collision.
2. Increment retransmission counter.
3. Was the maximum number of transmission attempts reached? If so, abort transmission.
4. Calculate and wait the random back off period based on number of collisions.
5. Re-enter main procedure at stage 1.

****

**Code Snippet:**

**Client Side**

while(1)

{

printf ("Me: ");

gets (msg);

clock\_t begin = clock();

send(sck , msg , strlen(msg) , 0);

clock\_t end = clock();

printf("%d",(begin-end));

}

**Server side**

while(s<5)

{

if((recv\_size = recv(new\_socket , client\_reply , 2000 , 0)) == SOCKET\_ERROR)

puts("recv failed");

else

{

client\_reply[recv\_size] = '\0';

puts(client\_reply);

}

timer (1);

s++;

}

duration = ( clock() - start ) / (double) CLOCKS\_PER\_SEC;

printf("Waiting Time=%lf \n\n",duration);

iResult = shutdown(new\_socket, SD\_RECEIVE);

if (iResult == SOCKET\_ERROR) {

printf("shutdown failed: %d\n", WSAGetLastError());

closesocket(new\_socket);

WSACleanup();

return 1;

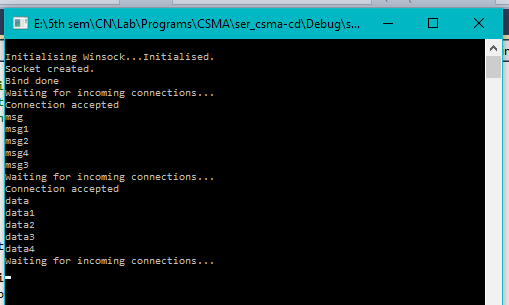
}

**Snapshot**

**Server waiting for clients.**

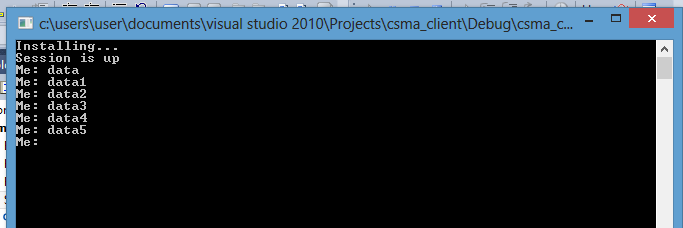
Clients can send a message for certain period. After that period that client will be disconnected and other clients can send message.

Server

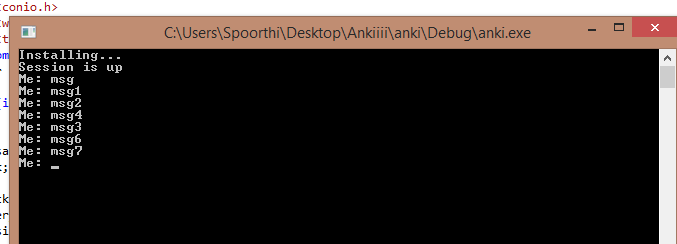


Client 1

Only five messages are received. After that even the client sends the message it wont be received by the server.



Client 2



1. Realize using client-server environment using following scenario. Consider the mesh topology of 5 nodes and the cost of it of your choice. Use the Bellman ford algorithm to find the set of all shortest path from all the nodes to node 5.

**Theoretical Description**

**Routing** is the process of selecting a path for traffic in a network or between or across multiple networks. Routing is performed for many types of networks, including circuit-switched networks, computer networks as well as in networks used in public and private transportation.

Shortest path is chosen for moving information across an inter-network from a source to a destination. Here Bellman ford algorithm is used to find the shortest distance.

**Algorithm**

We can use Bellman Ford for directed as well as un-directed graphs.

1. It maintains a list of unvisited vertices.
2. It chooses a vertex (the source) and assigns a maximum possible cost (i.e. infinity) to every other vertex.
3. The cost of the source remains zero as it actually takes nothing to reach from the source vertex to itself.
4. In every subsequent iteration of the algorithm it tries to relax each edge in the graph (by minimizing the cost of the vertex on which the edge is incident).
5. It repeats step 4 for |V|-1 times. By the last iteration we would have gotten some shortest path from Source to each vertex.

**Code Snippet**

**Client side**

while(1)

{

if((recv\_size = recv(sck , server\_reply , 2000 , 0)) == SOCKET\_ERROR)

puts("recv failed");

else

{

server\_reply[recv\_size] = '\0';

for ( int i = 0; i < 5; i++ )

{

printf( "Vertex %d -> cost = %d\n" ,i+1, \*(server\_reply+i));

}

}

}

**Server side**

while(1){

int count,src\_router,i,j,k,w,v,min;

int cost\_matrix[100][100],dist[100],last[100];

int flag[100];

count = 5;

printf("\n Enter the cost of the following nodes\n");

for(i=0;i<count;i++){

for(j=0;j<count;j++){

printf("\n%d->%d:",i+1,j+1);

scanf("%d",&cost\_matrix[i][j]);

if(cost\_matrix[i][j]<0)cost\_matrix[i][j]=1000;

}}

src\_router = 5;

src\_router = src\_router - 1;

for(v=0;v<count;v++){

flag[v]=0;

last[v]=src\_router;

dist[v]=cost\_matrix[src\_router][v];

}

flag[src\_router]=1;

for(i=0;i<count;i++){

min=1000;

for(w=0;w<count;w++){

if(!flag[w])

if(dist[w]<min){

v=w;

min=dist[w];

}

}

flag[v]=1;

for(w=0;w<count;w++){

if(!flag[w])

if(min+cost\_matrix[v][w]<dist[w]){

dist[w]=min+cost\_matrix[v][w];

last[w]=v;

}

}

}

for(i=0;i<count;i++){

printf("\n%d==>%d\nPath taken: %d",src\_router+1,i+1,i+1);

w=i;

while(w!=src\_router){

printf("<--%d",last[w]+1);

w=last[w];

}

printf("\n Shortest path cost: %d\n",dist[i]);

}

printf("\n");

for (i = 0 ; i < count ; i++){

msg[i] = dist[i];

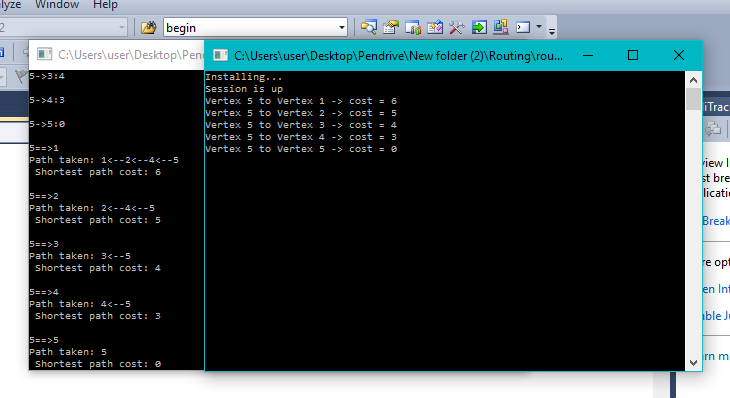
}

send(new\_socket , msg , strlen(msg) , 0);

}

**Snapshots**

The shortest path and distance is calculated by the client and sent to server.



1. Create a client-server environment with socket programming. Consider any jpeg image of your choice and perform encryption and decryption of an image using DES algorithm.

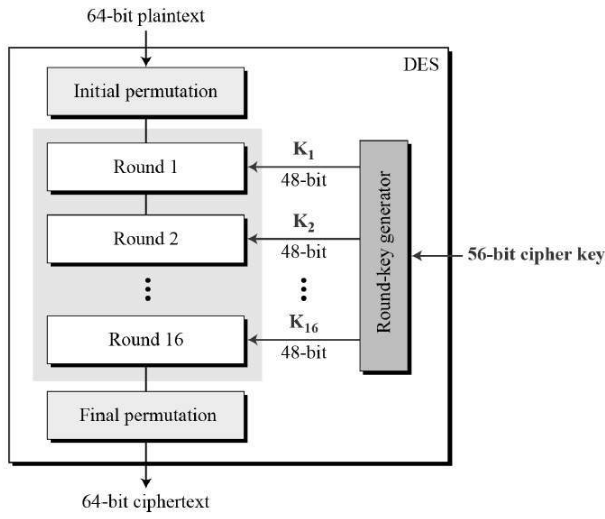
**Theoretical Description**

**Network security** consists of the policies and practices adopted to prevent and monitor unauthorized access, misuse, modification, or denial of a **computer network** and **network**-accessible resources.

There are many algorithms for encryption and decryption of data.

**Algorithm**

**DES** is a symmetric, **64 bit** block cipher as it uses the same key for both encryption and decryption and **only** operates on **64 bit** blocks of data at a time5 (be they plaintext or ciphertext). The key size used is 56 **bits**, however a **64 bit** (or eight-byte) key is actually input.

****

**Code Snippet**

**Server side**

while(1)

{

if((recv\_size = recv(new\_socket , client\_reply , 2000 , 0)) == SOCKET\_ERROR)

puts("recv failed");

else

{

Des d1;

char \*str1=new char[1000];

printf("\nRecieved data\n ");

client\_reply[recv\_size] = '\0';

puts(client\_reply);

str1=d1.Decrypt(client\_reply);

cout<<"\nData after decryption"<<endl<<str1<<endl;

}

}

**Client side**

Des d1,d2;

char \*str=new char[1000];

char \*str1=new char[1000];

FILE \*fptr;

FILE \*txt;

int c;

fptr=fopen("image1.jpeg","r");

txt=fopen("test1.txt","w");

if(fptr==NULL)

{

printf("IMAGE FILE IS EMPTY");

fclose(fptr);

}

else

{

printf("IMAGE READ SUCCESSFULLY\n");

do

{

c=fgetc(fptr);

for(int i=0;i<=7;i++)

{

if(c&(1<<(7-i)))

{

fputc('1',txt);

}

else

{

fputc('0',txt);

}

}

}while(c!=EOF);

}

fclose(fptr);

fclose(txt);

txt=fopen("test1.txt","r");

fgets(msg, 64, txt);

printf("Image data\n");

str=msg;

cout << str;

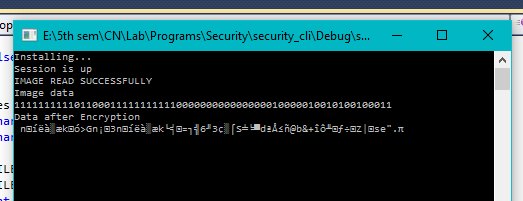
str1=d1.Encrypt(str);

cout<<"\nData after Encryption\n "<<str1<<endl;

send(sck , str1 , strlen(str1) , 0);

**Snapshots**

Client-side: Image data is encrypted and is sent to the server.



Server-side: Encrypted image is received from the client and is decrypted by the server.

