**COMPUTER NETWORKING LAB REPORT**



**Submitted by :**

Name : Akash S Bilgi

Division : A

Roll.no : 06

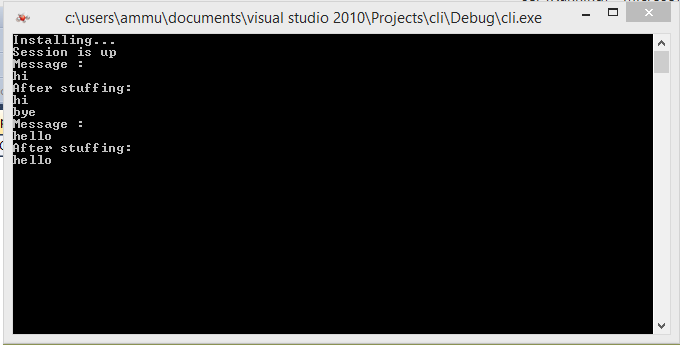
USN : 2bv14is007

**CONTENTS**

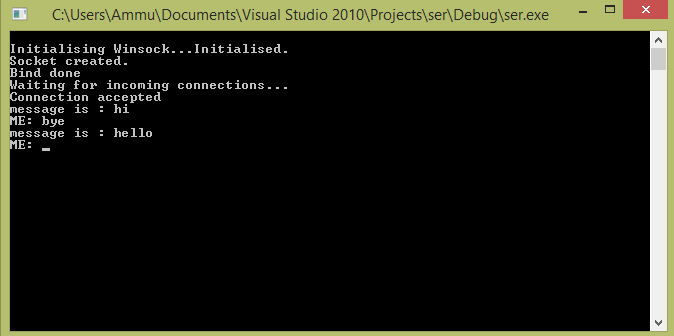
1. Socket programming
2. Byte stuffing
3. CRC
4. Flow control
5. Access control
6. Bellman ford algorithm
7. DES encryption and decryption
8. Analysis of code with respect to:

* A client/server program
* Execute localhost
* Implement the basic chat application on peer machine

Server side



Client side:



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);

}

}

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';

puts(client\_reply);

printf ("Me: ");

gets (msg);

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

}

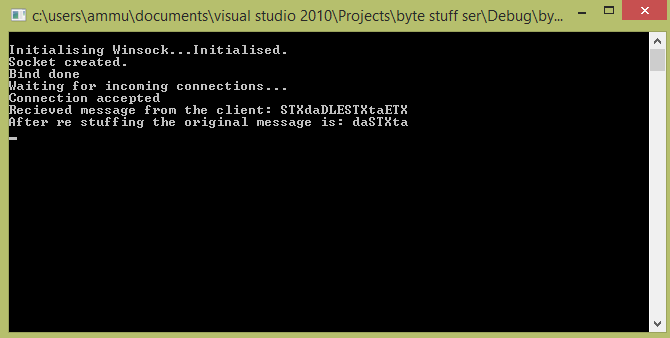
}

1. Write a client/server program for the given input to perform byte stuffing and calculate the overhead cost.

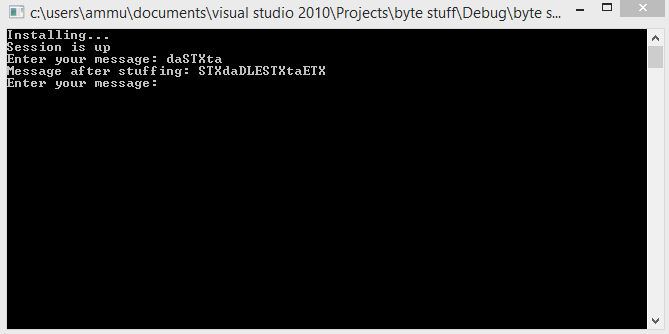
Algorithm:

* Read the data to be sent.
* If STX/ETX/DLE is encountered in data section then perform byte stuffing by embedding DLE before STX/ETX/DLE to avoid conflicts.
* Send this byte stuffed data to server.
* Remove stuffed DLE and get the original data

Server side:



Client side:



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);

}

}

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);

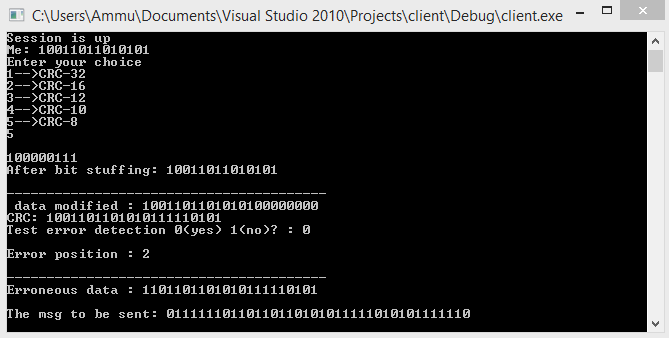
}

1. Write a client/server program for the standard polynomial used for the internet and detect how many number of bits are corrupted.

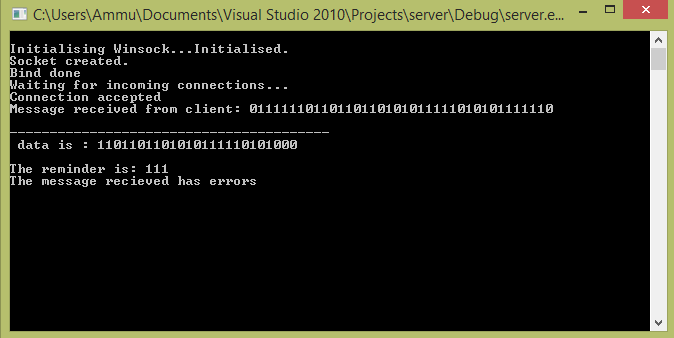
Algorithm:

* The input message will read from user
* Additional bits of 0 is added to the message
* Then the standard polynomial is taken as divisor, dividing the message (XOR).
* The remainder is appended with the message and is sent to server.
* Server performs division on the received data, if the remainder got is 0 then there is no error in transmission, else the data is rejected.

Server side:



Client side:



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);

}

}

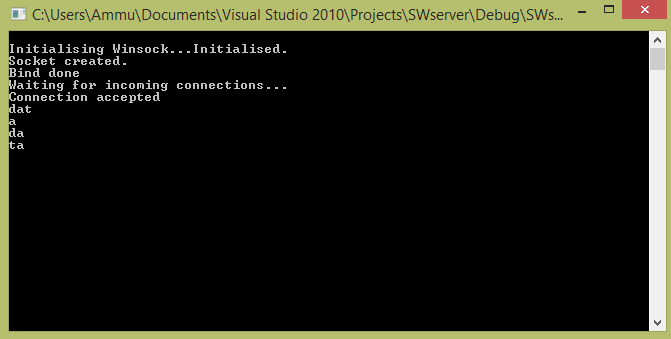
1. Create a client/server environment for realizing flow control of data link layer , where receiver doesn’t have negative acknowledgment feature, declare the entire variable at both sender and receiver with respect to sliding window in line with textbook referred during the course, program should allow user to study abnormal situation

Activity: formula for efficiency of all the ARQ protocols

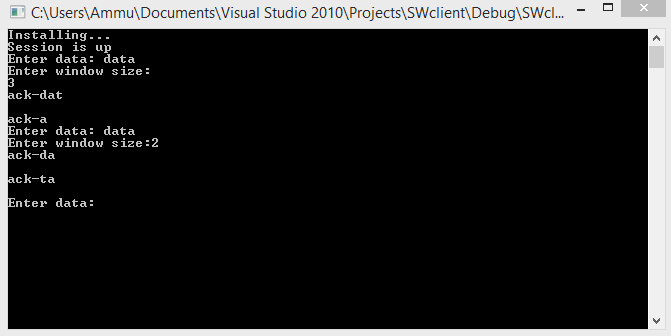
1. Plot the graph of efficiency vs probability of error of all ARQ packets by varying probability of error. Assume other relevant parameters.

Algorithm:

* Read the data to be sent
* Set sender’s window size as 2^m-1 and receiver’s window size as 1
* Send cumulative acknowledgement to the client after sending the packets equal to window size.
* If the acknowledgment is not received in time, then re-send the packets.

Server side code: 

Client side code:



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);

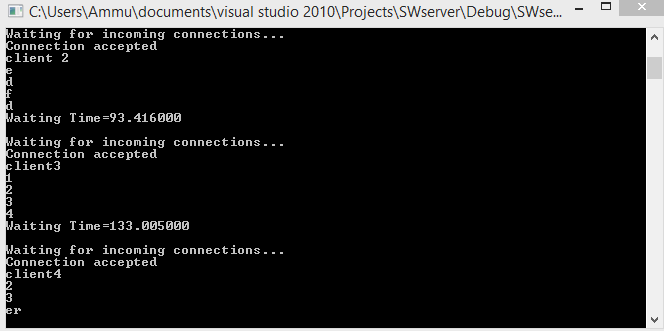
send(new\_socket , msg1 , strlen(msg1) , 0);

}

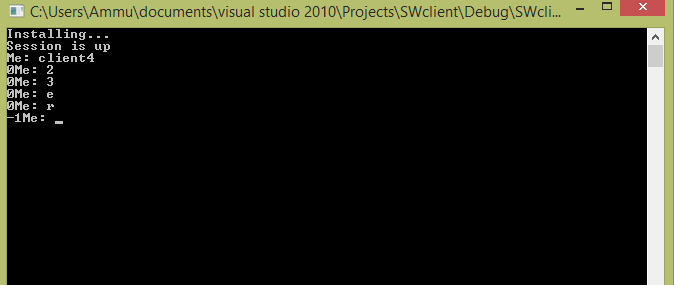
}

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 successful transmissions and failed transmissions by assuming appropriate number of frames to be transmitted from sender.

Server side code:



Client side code:



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;

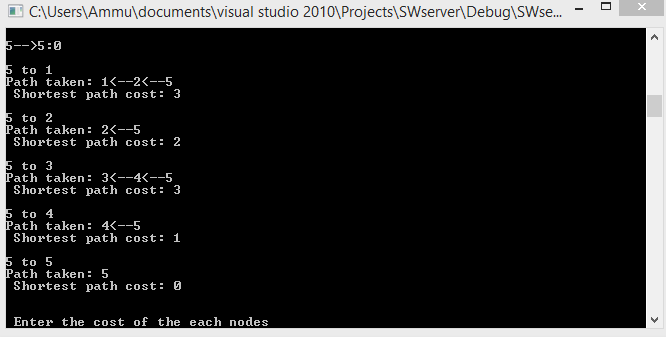
}

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.

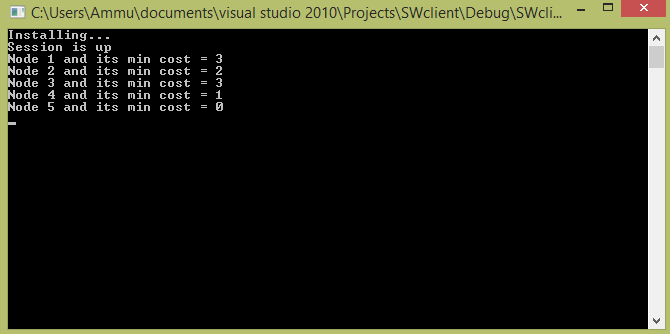
Algorithm:

* Initialize graph.
* Replace with shortest weights repeatedly. In every subsequent iteration of the algorithm it tries to relax each edge in the graph by minimizing the cost of the incident. Repeat this step |v|-1 times, in the last iteration, we get the shortest path from source to destination.
* Check for negative weight cycle.

Server side code:



Client side code:



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);

}

1. Create client/server environment sing socket programming with following specification.

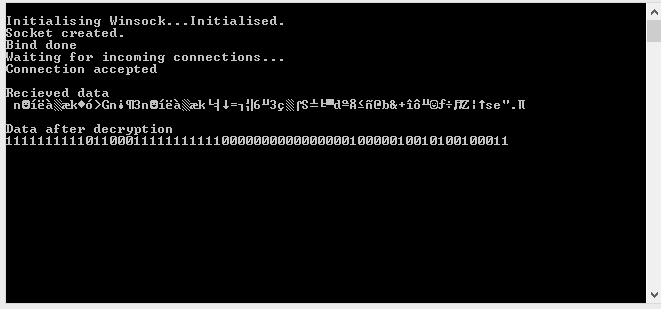
Consider any jpeg image of your choice and perform encryption and decryption of an image using DES algorithm.

Algorithm:

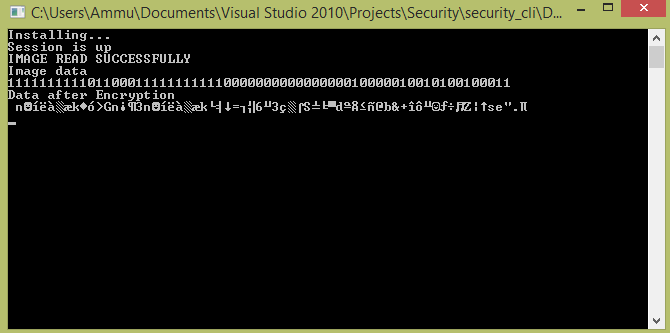
DES is a symmetric, 64 bit block cipher as it uses the same key for both encryption and decryption and only operates on 64bit blocks of data at a time (be they plaintext or ciphertext). The key size used is 56 bits, however a 64bit (or eight-byte) key is actually input.

**Snapshots:**

Server side:



Client side:



Server side code:

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 code:

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);