ASSIGNMENT 2

Omar Khairat Mohamed Abodeif - 19016063

Abdelaziz Mohamed Abdelaziz - 19015941

o

Assignment Statement

Since you've learned about the socket interface and how it is used by an application; by now, you're pretty much an expert in how to use the socket interface over a reliable transport layer, so now seems like a good time to implement your own socket layer and reliable transport layer! You'll get to learn how the socket interface is implemented by the kernel and how a reliable transport protocol like TCP runs on top of an unreliable delivery mechanism (which is the real world, since in real world networks nothing is reliable). This lab should be fun since your implementation will differ very little from what would be required in a real-world situation.

The network communication in last assignment was provided through a reliable transfer protocol (TCP/IP). In this assignment, you are required to implement a reliable transfer service on top of the UDP/IP protocol. In other words, you need to implement a service that guarantees the arrival of datagrams in the correct order on top of the UDP/IP protocol, along with congestion control.

Specifications

Suppose you've a file and you want to send this file from one side to the other (server to client). You will need to split the file into chunks of data of fixed length and add the data of one chunk to a UDP datagram packet in the data field of the packet. You need to implement TCP with congestion control.

Data Structures Used

- Array of characters (string) to save our request information.
- Vector of Strings to store our requests independent of each other.
- Struct socket_address to utilize our WebSocket.
- struct arg_struct { int client_socket; long long* timer; }.

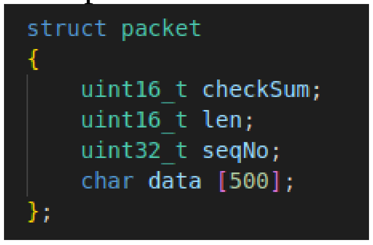
User Guide

- Run the server from terminal after changing the directory to its directory and type the following in terminal:
  - g++ -o HTTP-Server main.cpp
  - ./HTTP-Server
- Run the client from terminal after changing the directory to its directory and type the following in terminal:
  - g++ -o HTTP-Client main.cpp
  - ./HTTP-Client

Packet type and fields (Server) :

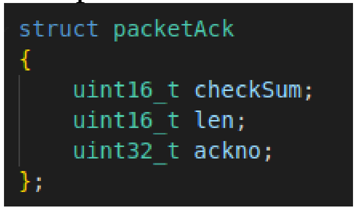
- There are two kinds of packets:

  Data packets :



```
struct packet
{
    uint16_t checkSum;
    uint16_t len;
    uint32_t seqNo;
    char data [500];
};
```

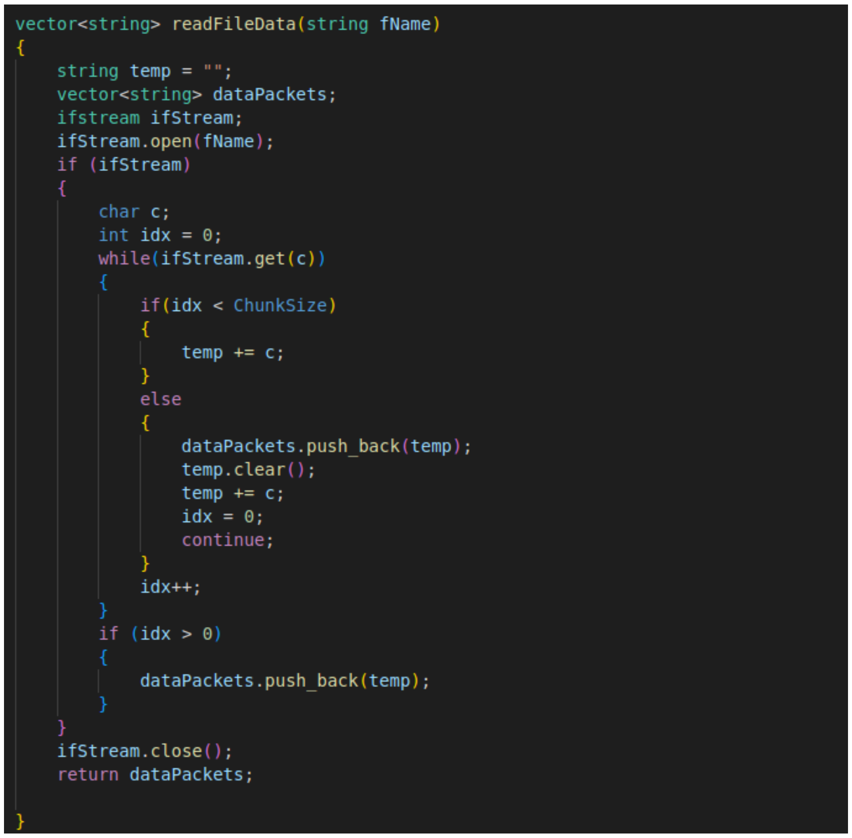
Ack packets :



```
struct packetAck
{
    uint16_t checkSum;
    uint16_t len;
    uint32_t ackno;
};
```

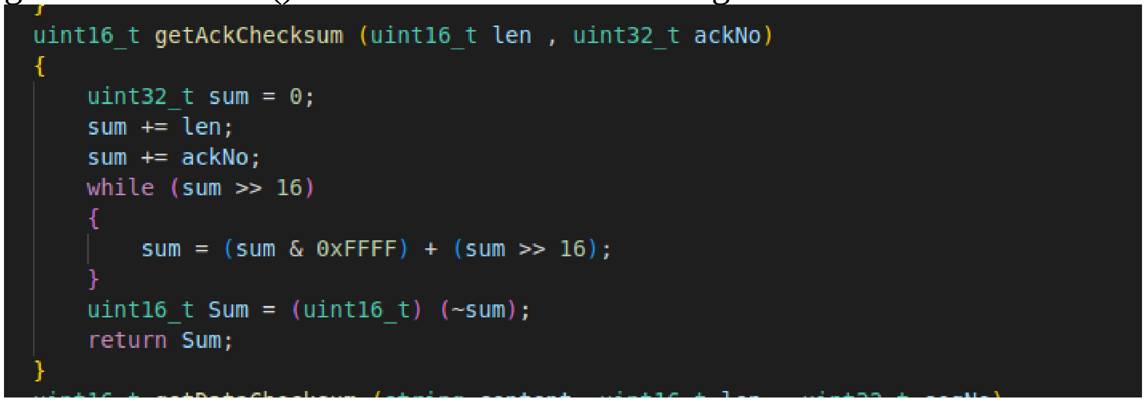
Main functions Server:

readFileData() : this function is used to read content of the file.



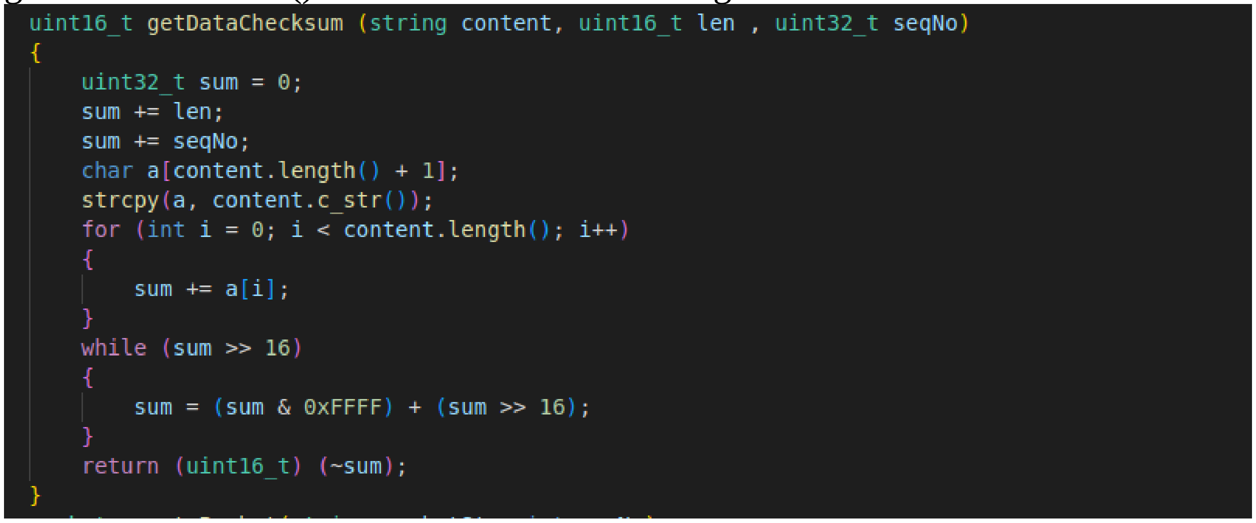
```
vector<string> readFileData(string fName)
{
    string temp = "";
    vector<string> dataPackets;
    ifstream ifStream;
    ifStream.open(fName);
    if (ifStream)
    {
        char c;
        int idx = 0;
        while(ifStream.get(c))
        {
            if(idx < ChunkSize)
            {
                temp += c;
            }
            else
            {
                dataPackets.push_back(temp);
                temp.clear();
                temp += c;
                idx = 0;
                continue;
            }
            idx++;
        }
        if (idx > 0)
        {
            dataPackets.push_back(temp);
        }
    }
    ifStream.close();
    return dataPackets;

}
```

getAckChecksum() = > this function is used to get Ack Checksum.



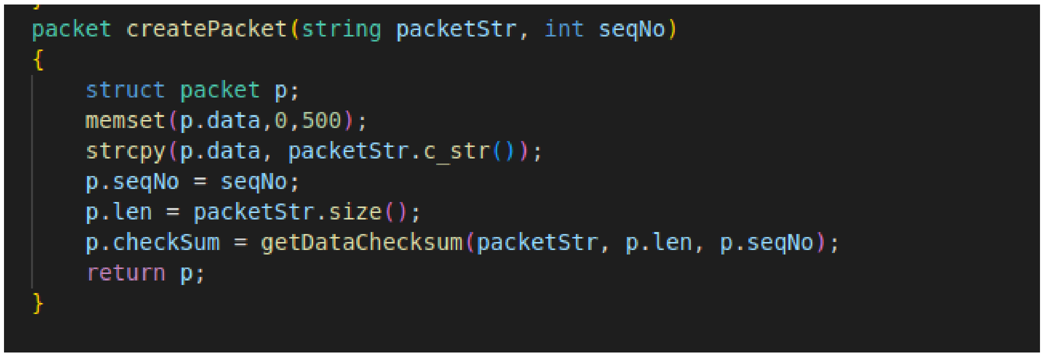
```
uint16_t getAckChecksum (uint16_t len , uint32_t ackNo)
{
    uint32_t sum = 0;
    sum += len;
    sum += ackNo;
    while (sum >> 16)
    {
        sum = (sum & 0xFFFF) + (sum >> 16);
    }
    uint16_t Sum = (uint16_t) (~sum);
    return Sum;
}
```

gerDataChecksum ()=> this function is used to get Data of Checksum.



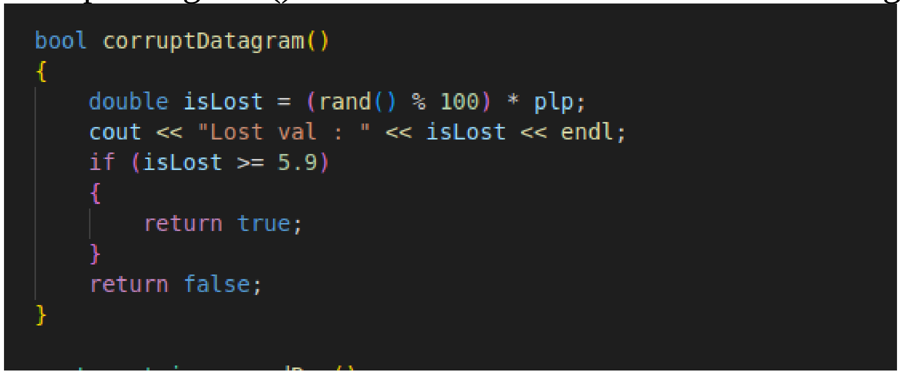
```
uint16_t getDataChecksum (string content, uint16_t len , uint32_t seqNo)
{
    uint32_t sum = 0;
    sum += len;
    sum += seqNo;
    char a[content.length() + 1];
    strcpy(a, content.c_str());
    for (int i = 0; i < content.length(); i++)
    {
        sum += a[i];
    }
    while (sum >> 16)
    {
        sum = (sum & 0xFFFF) + (sum >> 16);
    }
    return (uint16_t) (~sum);
}
```

createPacket() => in this function is used to create packet that will be send to be client



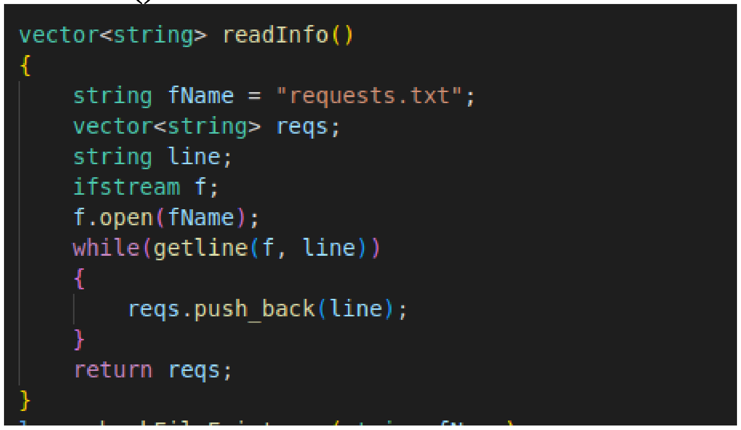
```
packet createPacket(string packetStr, int seqNo)
{
    struct packet p;
    memset(p.data,0,500);
    strcpy(p.data, packetStr.c_str());
    p.seqNo = seqNo;
    p.len = packetStr.size();
    p.checkSum = getDataChecksum(packetStr, p.len, p.seqNo);
    return p;
}
```

corruptDatagram ()=> this function is used to check if datagram is corrupted or not



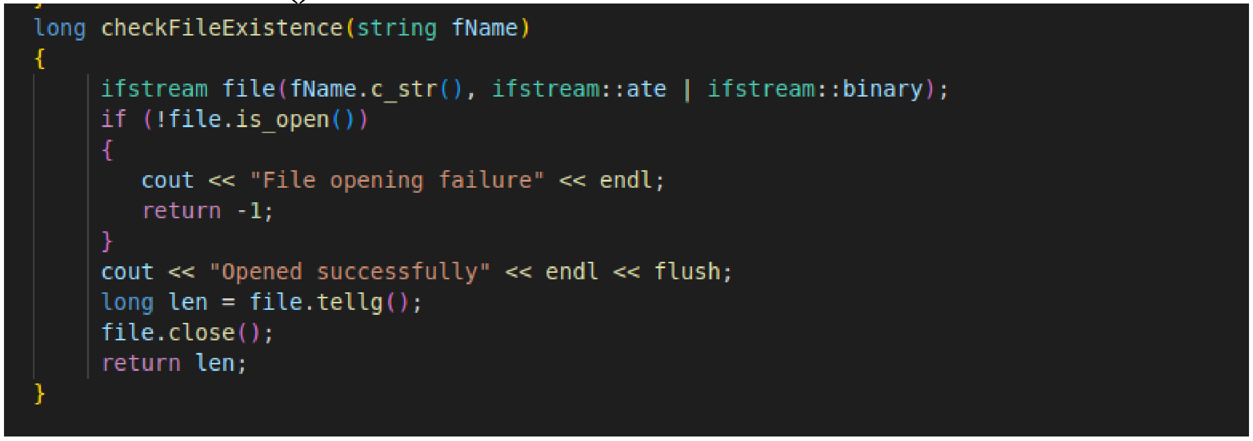
```
bool corruptDatagram()
{
    double isLost = (rand() % 100) * plp;
    cout << "Lost val : " << isLost << endl;
    if (isLost >= 5.9)
    {
        return true;
    }
    return false;
}
```

readInfo() => this function is used to readInfo of command file



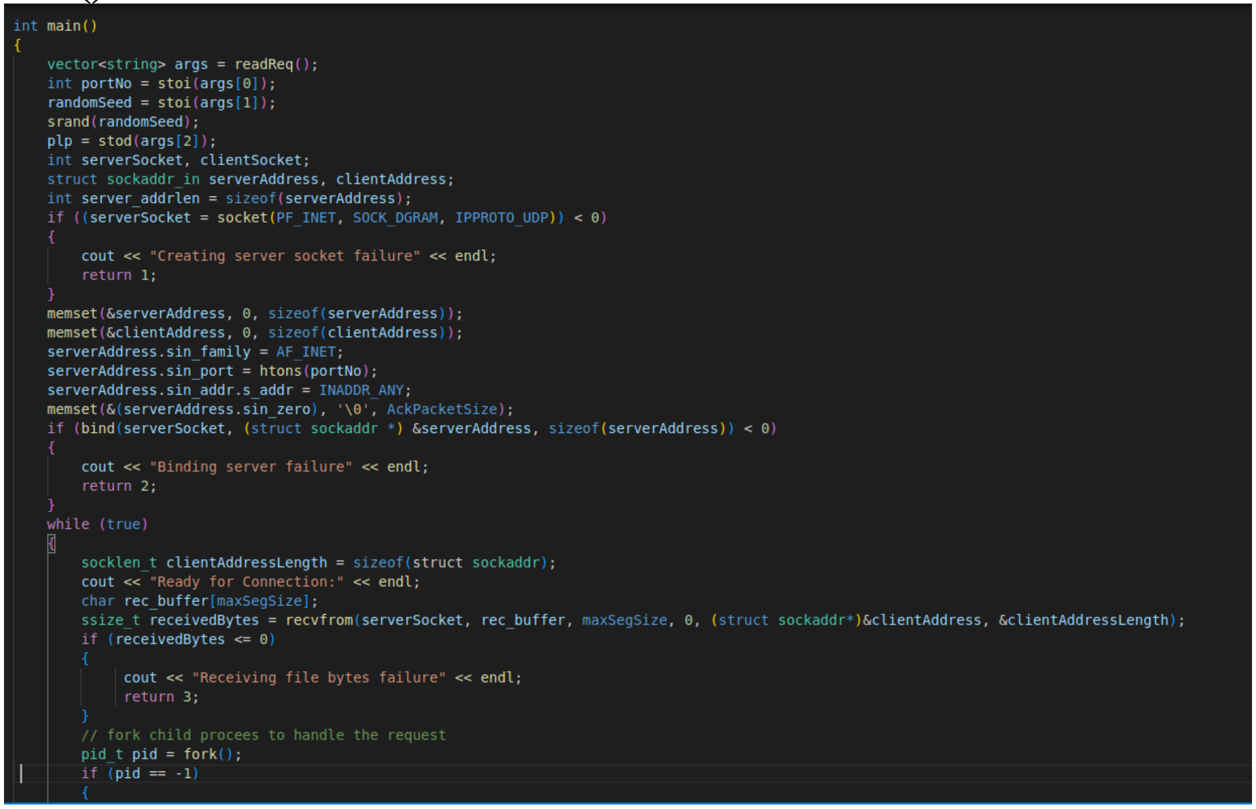
```
vector<string> readInfo()
{
    string fName = "requests.txt";
    vector<string> reqs;
    string line;
    ifstream f;
    f.open(fName);
    while(getline(f, line))
    {
        reqs.push_back(line);
    }
    return reqs;
}
```

checkfileExistence() => check if the file is exist or not

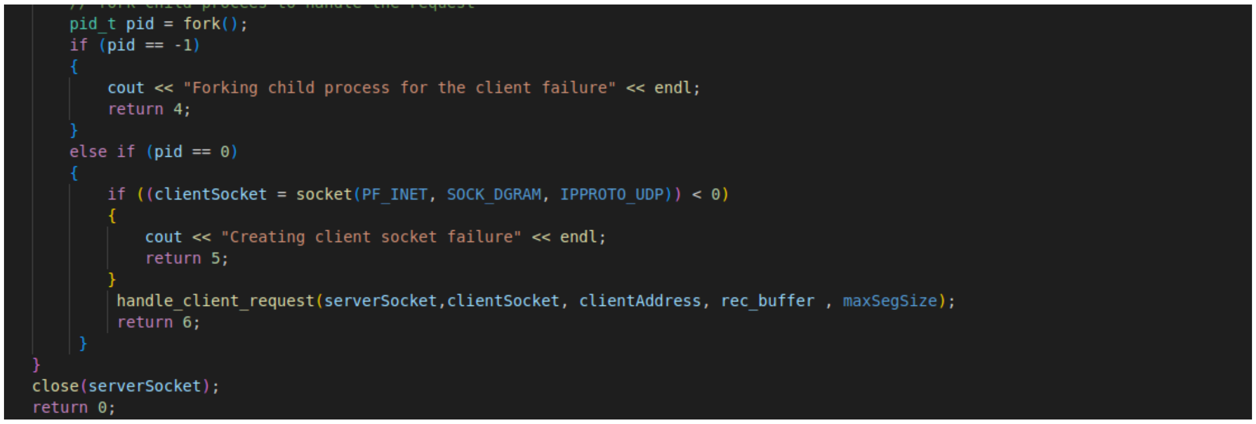


```
long checkFileExistence(string fName)
{
    ifstream file(fName.c_str(), ifstream::ate | ifstream::binary);
    if (!file.is_open())
    {
        cout << "File opening failure" << endl;
        return -1;
    }
    cout << "Opened successfully" << endl << flush;
    long len = file.tellg();
    file.close();
    return len;
}
```

main()

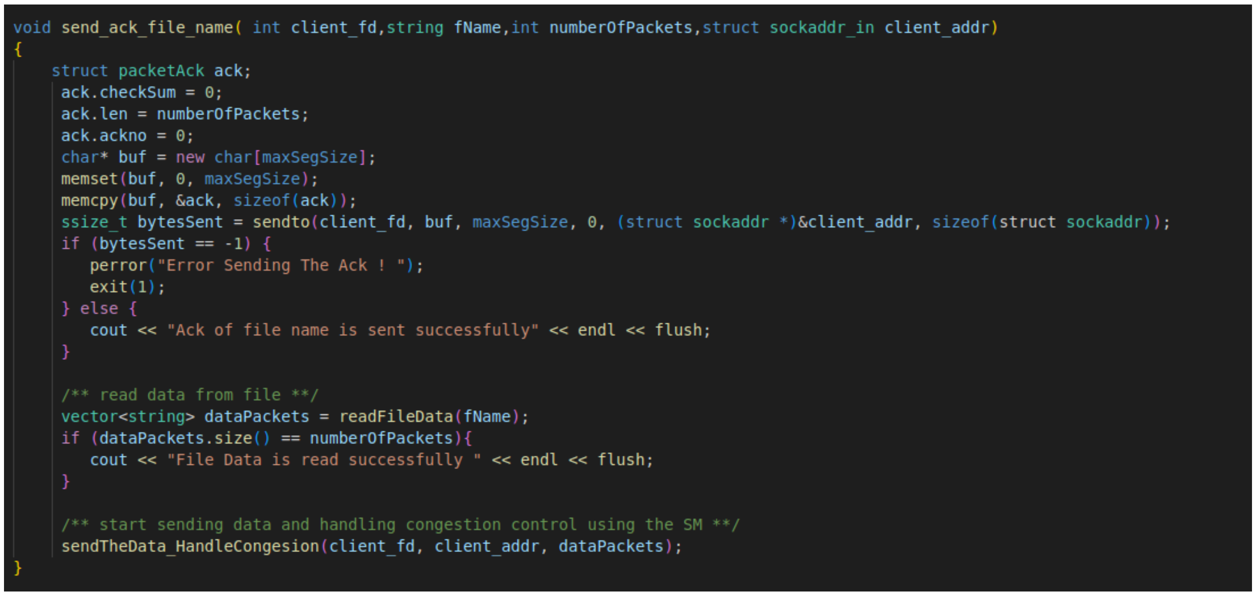


```
int main()
{
    vector<string> args = readReq();
    int portNo = stoi(args[0]);
    randomSeed = stoi(args[1]);
    srand(randomSeed);
    plp = stod(args[2]);
    int serverSocket, clientSocket;
    struct sockaddr_in serverAddress, clientAddress;
    int server_addrlen = sizeof(serverAddress);
    if ((serverSocket = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
    {
        cout << "Creating server socket failure" << endl;
        return 1;
    }
    memset(&serverAddress, 0, sizeof(serverAddress));
    memset(&clientAddress, 0, sizeof(clientAddress));
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_port = htons(portNo);
    serverAddress.sin_addr.s_addr = INADDR_ANY;
    memset(&(serverAddress.sin_zero), '\0', AckPacketSize);
    if (bind(serverSocket, (struct sockaddr *) &serverAddress, sizeof(serverAddress)) < 0)
    {
        cout << "Binding server failure" << endl;
        return 2;
    }
    while (true)
    {
        socklen_t clientAddressLength = sizeof(struct sockaddr);
        cout << "Ready for Connection:" << endl;
        char rec_buffer[maxSegSize];
        ssize_t receivedBytes = recvfrom(serverSocket, rec_buffer, maxSegSize, 0, (struct sockaddr*)&clientAddress, &clientAddressLength);
        if (receivedBytes <= 0)
        {
            cout << "Receiving file bytes failure" << endl;
            return 3;
        }
        // fork child procees to handle the request
        pid_t pid = fork();
        if (pid == -1)
        {
```



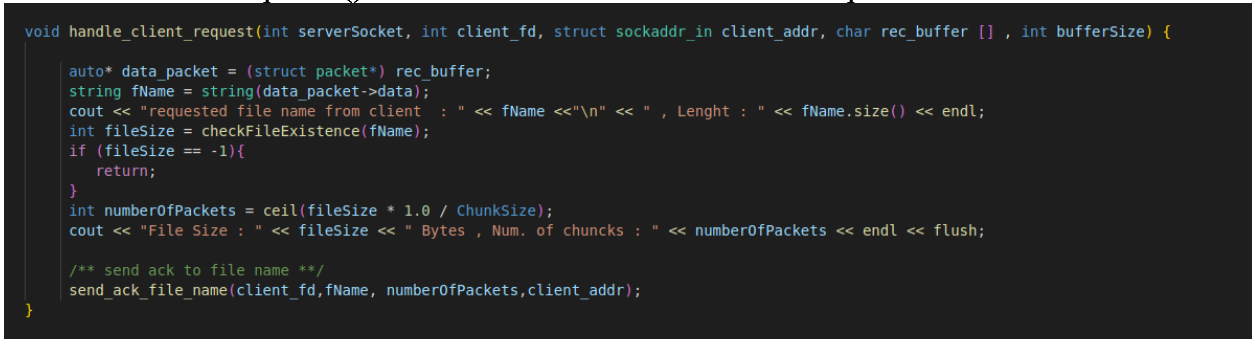
```
        pid_t pid = fork();
        if (pid == -1)
        {
            cout << "Forking child process for the client failure" << endl;
            return 4;
        }
        else if (pid == 0)
        {
            if ((clientSocket = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
            {
                cout << "Creating client socket failure" << endl;
                return 5;
            }
             handle_client_request(serverSocket,clientSocket, clientAddress, rec_buffer , maxSegSize);
             return 6;
         }
    }
    close(serverSocket);
    return 0;
```

send_ack_file_name(client,fname,numberofPackets,client_addr)=> it is used to send ack of file.



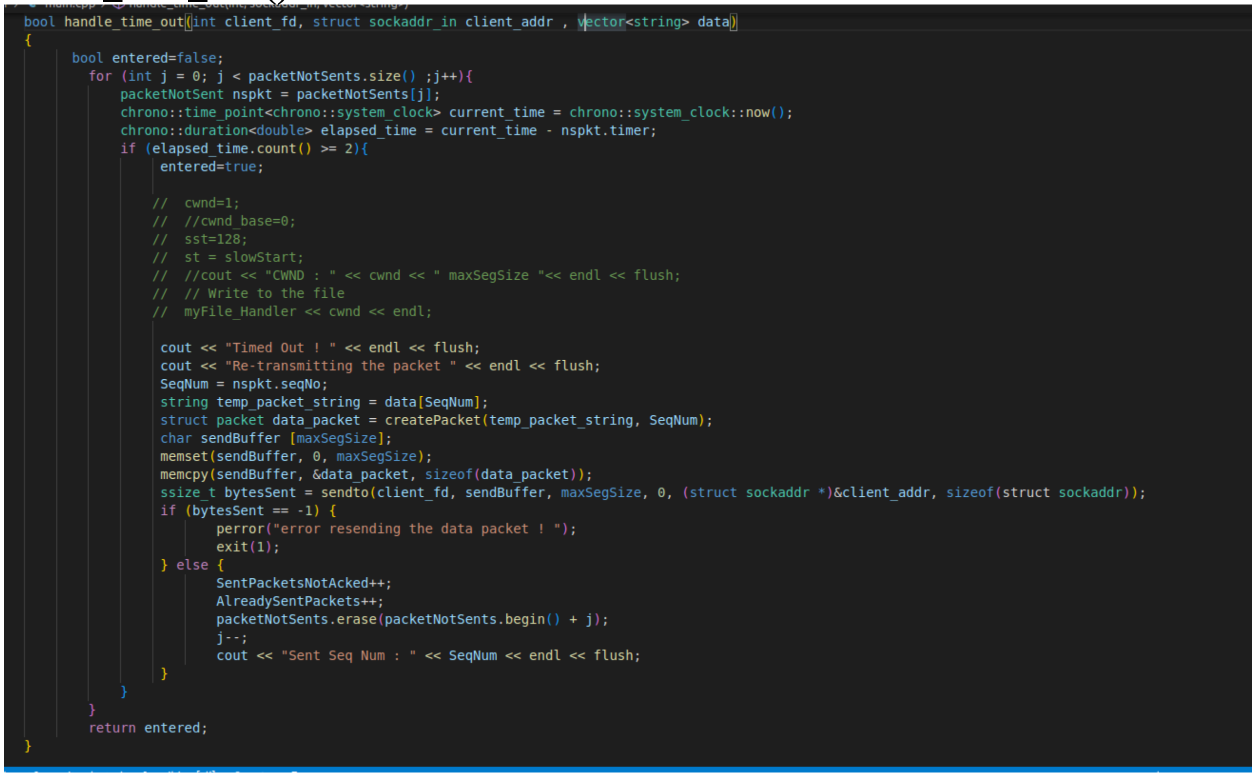
```
void send_ack_file_name( int client_fd,string fName,int numberOfPackets,struct sockaddr_in client_addr)
{
    struct packetAck ack;
     ack.checkSum = 0;
     ack.len = numberOfPackets;
     ack.ackno = 0;
     char* buf = new char[maxSegSize];
     memset(buf, 0, maxSegSize);
     memcpy(buf, &ack, sizeof(ack));
     ssize_t bytesSent = sendto(client_fd, buf, maxSegSize, 0, (struct sockaddr *)&client_addr, sizeof(struct sockaddr));
     if (bytesSent == -1) {
        perror("Error Sending The Ack ! ");
        exit(1);
     } else {
        cout << "Ack of file name is sent successfully" << endl << flush;
     }

     /** read data from file **/
     vector<string> dataPackets = readFileData(fName);
     if (dataPackets.size() == numberOfPackets){
        cout << "File Data is read successfully " << endl << flush;
     }

     /** start sending data and handling congestion control using the SM **/
     sendTheData_HandleCongesion(client_fd, client_addr, dataPackets);
}
```

handle_client_request () => it is used to handle client request .



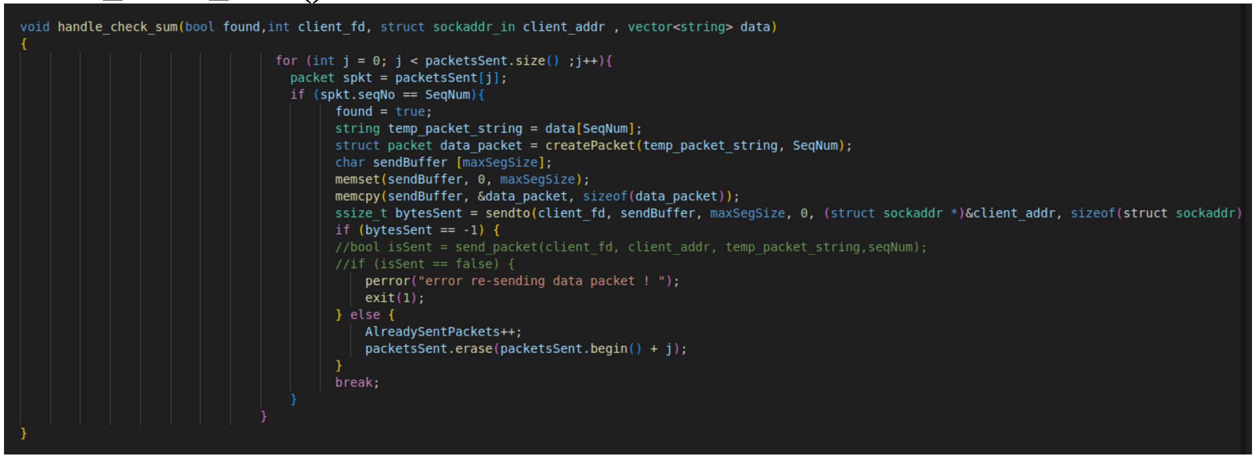
```
void handle_client_request(int serverSocket, int client_fd, struct sockaddr_in client_addr, char rec_buffer [] , int bufferSize) {

     auto* data_packet = (struct packet*) rec_buffer;
     string fName = string(data_packet->data);
     cout << "requested file name from client  : " << fName <<"\n" << " , Lenght : " << fName.size() << endl;
     int fileSize = checkFileExistence(fName);
     if (fileSize == -1){
        return;
     }
     int numberOfPackets = ceil(fileSize * 1.0 / ChunkSize);
     cout << "File Size : " << fileSize << " Bytes , Num. of chuncks : " << numberOfPackets << endl << flush;

     /** send ack to file name **/
     send_ack_file_name(client_fd,fName, numberOfPackets,client_addr);
}
```

handle_time_out ()



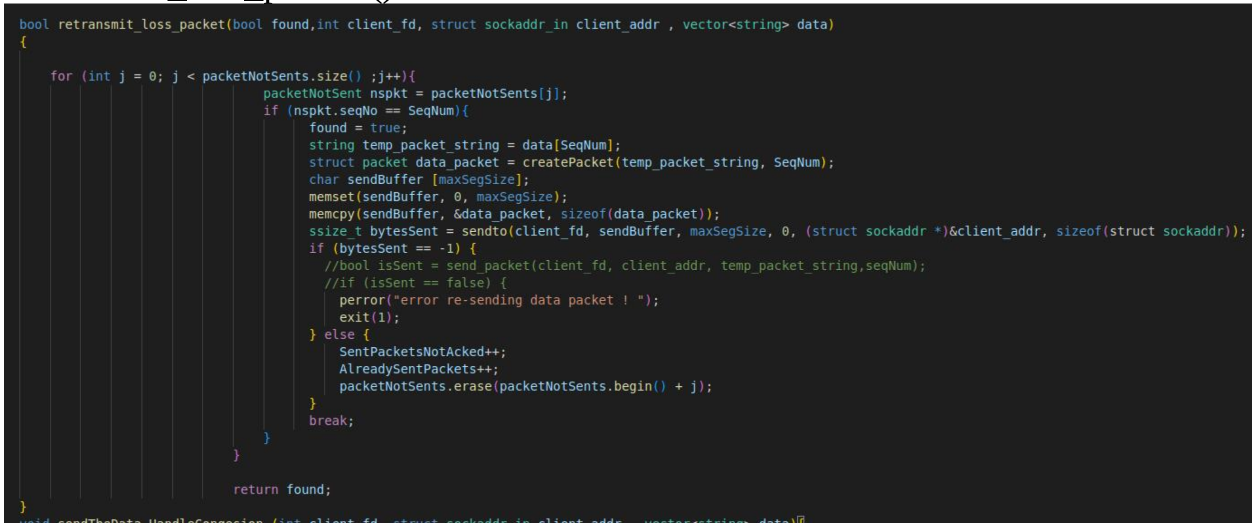
```
bool handle_time_out(int client_fd, struct sockaddr_in client_addr , vector<string> data)
{
        bool entered=false;
          for (int j = 0; j < packetNotSents.size() ;j++){
              packetNotSent nspkt = packetNotSents[j];
              chrono::time_point<chrono::system_clock> current_time = chrono::system_clock::now();
              chrono::duration<double> elapsed_time = current_time - nspkt.timer;
              if (elapsed_time.count() >= 2){
                   entered=true;

                  //  cwnd=1;
                  //  //cwnd_base=0;
                  //  sst=128;
                  //  st = slowStart;
                  //  //cout << "CWND : " << cwnd << " maxSegSize "<< endl << flush;
                  //  // Write to the file
                  //  myFile_Handler << cwnd << endl;

                   cout << "Timed Out ! " << endl << flush;
                   cout << "Re-transmitting the packet " << endl << flush;
                   SeqNum = nspkt.seqNo;
                   string temp_packet_string = data[SeqNum];
                   struct packet data_packet = createPacket(temp_packet_string, SeqNum);
                   char sendBuffer [maxSegSize];
                   memset(sendBuffer, 0, maxSegSize);
                   memcpy(sendBuffer, &data_packet, sizeof(data_packet));
                   ssize_t bytesSent = sendto(client_fd, sendBuffer, maxSegSize, 0, (struct sockaddr *)&client_addr, sizeof(struct sockaddr));
                   if (bytesSent == -1) {
                          perror("error resending the data packet ! ");
                          exit(1);
                   } else {
                          SentPacketsNotAcked++;
                          AlreadySentPackets++;
                          packetNotSents.erase(packetNotSents.begin() + j);
                          j--;
                          cout << "Sent Seq Num : " << SeqNum << endl << flush;
                   }
              }
          }
          return entered;
}
```

handle_check_sum ()



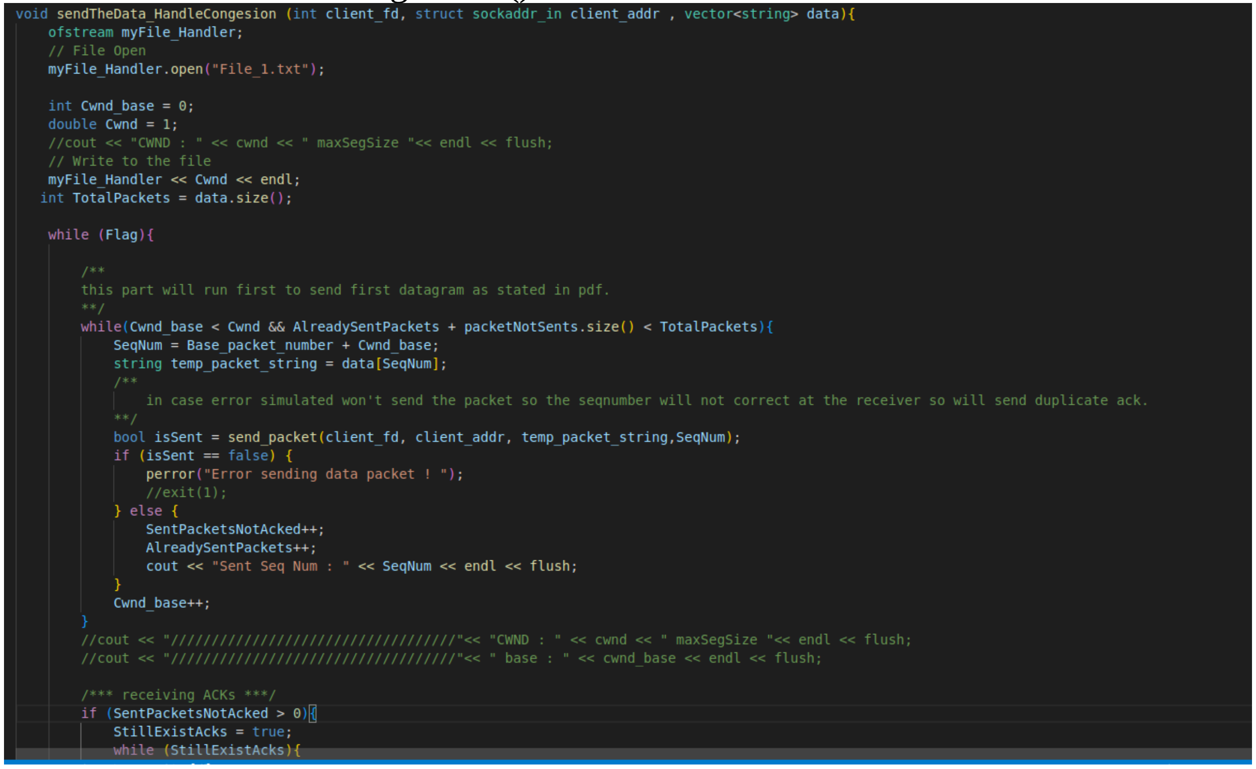
```
void handle_check_sum(bool found,int client_fd, struct sockaddr_in client_addr , vector<string> data)
{
                                  for (int j = 0; j < packetsSent.size() ;j++){
                                    packet spkt = packetsSent[j];
                                    if (spkt.seqNo == SeqNum){
                                          found = true;
                                          string temp_packet_string = data[SeqNum];
                                          struct packet data_packet = createPacket(temp_packet_string, SeqNum);
                                          char sendBuffer [maxSegSize];
                                          memset(sendBuffer, 0, maxSegSize);
                                          memcpy(sendBuffer, &data_packet, sizeof(data_packet));
                                          ssize_t bytesSent = sendto(client_fd, sendBuffer, maxSegSize, 0, (struct sockaddr *)&client_addr, sizeof(struct sockaddr)
                                          if (bytesSent == -1) {
                                          //bool isSent = send_packet(client_fd, client_addr, temp_packet_string,seqNum);
                                          //if (isSent == false) {
                                              perror("error re-sending data packet ! ");
                                              exit(1);
                                          } else {
                                              AlreadySentPackets++;
                                              packetsSent.erase(packetsSent.begin() + j);
                                          }
                                          break;
                                    }
                                  }
}
```

Retransmit_loss_packet ()

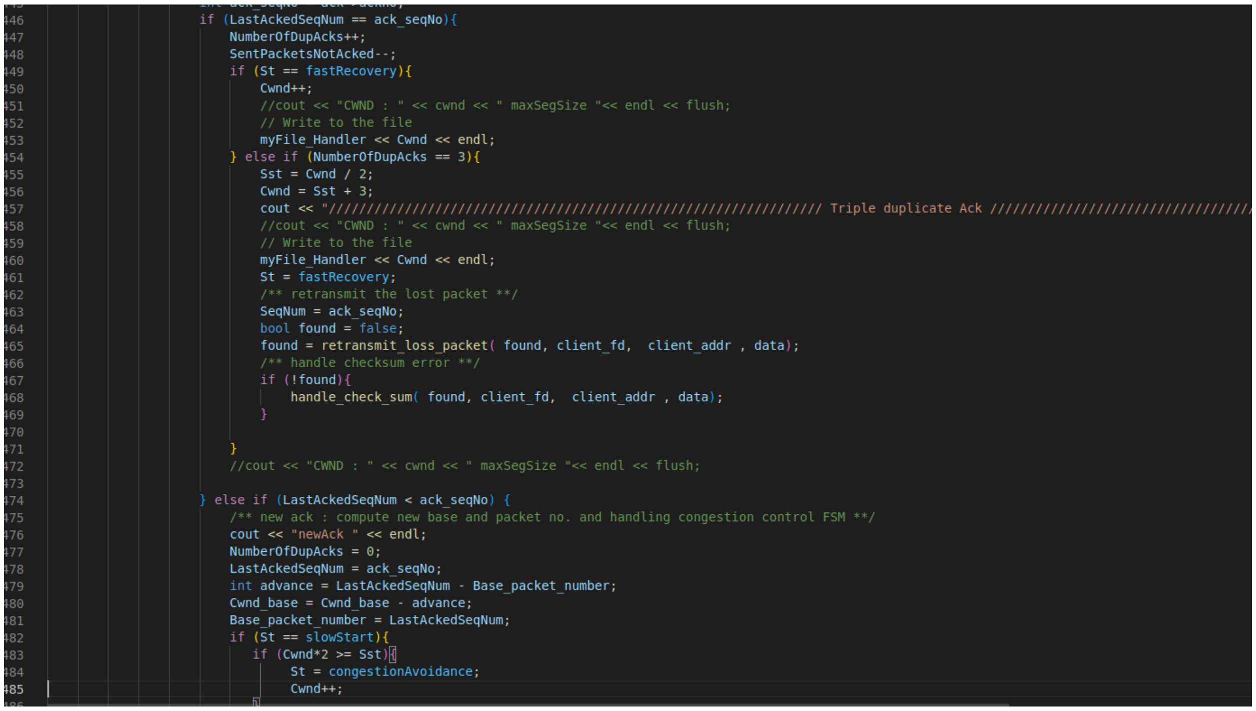


```
bool retransmit_loss_packet(bool found,int client_fd, struct sockaddr_in client_addr , vector<string> data)
{

    for (int j = 0; j < packetNotSents.size() ;j++){
                                packetNotSent nspkt = packetNotSents[j];
                                if (nspkt.seqNo == SeqNum){
                                      found = true;
                                      string temp_packet_string = data[SeqNum];
                                      struct packet data_packet = createPacket(temp_packet_string, SeqNum);
                                      char sendBuffer [maxSegSize];
                                      memset(sendBuffer, 0, maxSegSize);
                                      memcpy(sendBuffer, &data_packet, sizeof(data_packet));
                                      ssize_t bytesSent = sendto(client_fd, sendBuffer, maxSegSize, 0, (struct sockaddr *)&client_addr, sizeof(struct sockaddr));
                                      if (bytesSent == -1) {
                                        //bool isSent = send_packet(client_fd, client_addr, temp_packet_string,seqNum);
                                        //if (isSent == false) {
                                          perror("error re-sending data packet ! ");
                                          exit(1);
                                      } else {
                                          SentPacketsNotAcked++;
                                          AlreadySentPackets++;
                                          packetNotSents.erase(packetNotSents.begin() + j);
                                      }
                                      break;
                                }
                            }

                            return found;
}
```

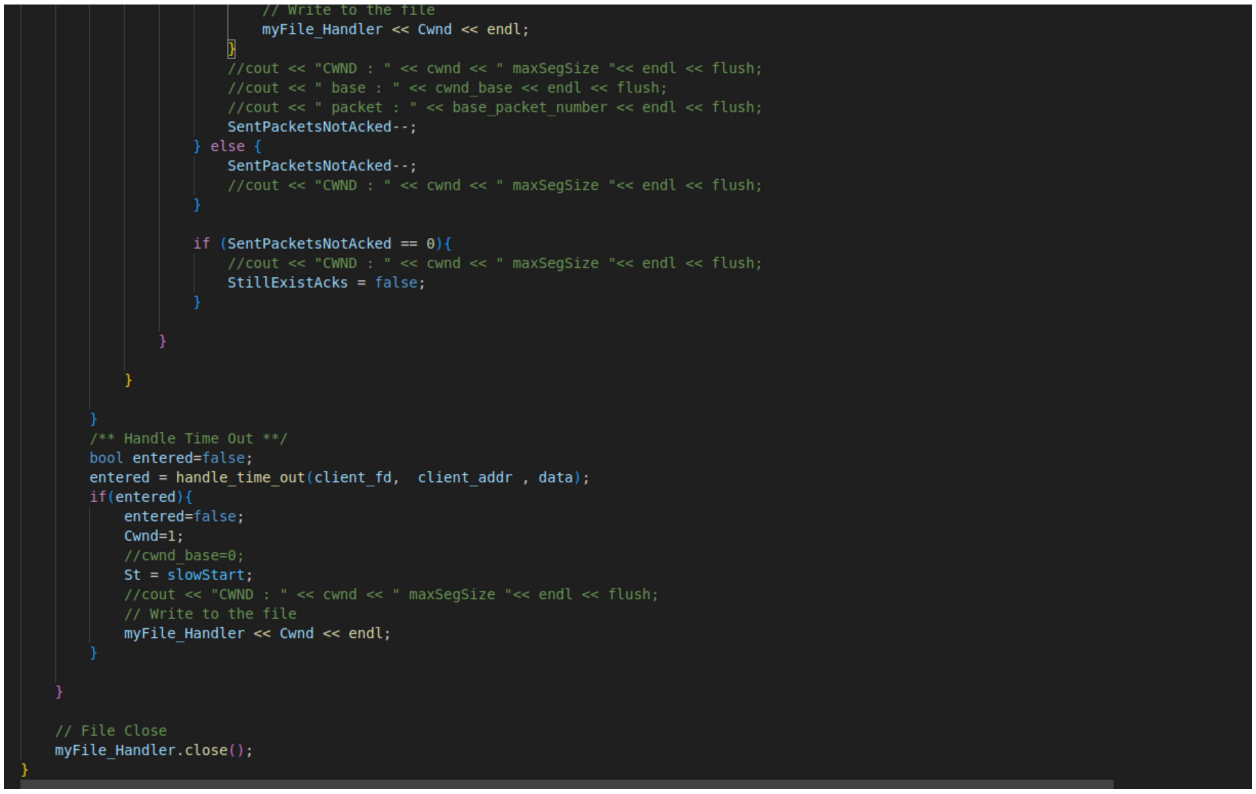
sendTheData_HandleCongestion ()



```
void sendTheData_HandleCongesion (int client_fd, struct sockaddr_in client_addr , vector<string> data){
    ofstream myFile_Handler;
    // File Open
    myFile_Handler.open("File_1.txt");

    int Cwnd_base = 0;
    double Cwnd = 1;
    //cout << "CWND : " << cwnd << " maxSegSize "<< endl << flush;
    // Write to the file
    myFile_Handler << Cwnd << endl;
   int TotalPackets = data.size();

    while (Flag){

        /**
        this part will run first to send first datagram as stated in pdf.
        **/
        while(Cwnd_base < Cwnd && AlreadySentPackets + packetNotSents.size() < TotalPackets){
            SeqNum = Base_packet_number + Cwnd_base;
            string temp_packet_string = data[SeqNum];
            /**
                in case error simulated won't send the packet so the seqnumber will not correct at the receiver so will send duplicate ack.
            **/
            bool isSent = send_packet(client_fd, client_addr, temp_packet_string,SeqNum);
            if (isSent == false) {
                perror("Error sending data packet ! ");
                //exit(1);
            } else {
                SentPacketsNotAcked++;
                AlreadySentPackets++;
                cout << "Sent Seq Num : " << SeqNum << endl << flush;
            }
            Cwnd_base++;
        }
        //cout << "///////////////////////////////////"<< "CWND : " << cwnd << " maxSegSize "<< endl << flush;
        //cout << "///////////////////////////////////"<< " base : " << cwnd_base << endl << flush;

        /*** receiving ACKs ***/
        if (SentPacketsNotAcked > 0){
            StillExistAcks = true;
            while (StillExistAcks){
```

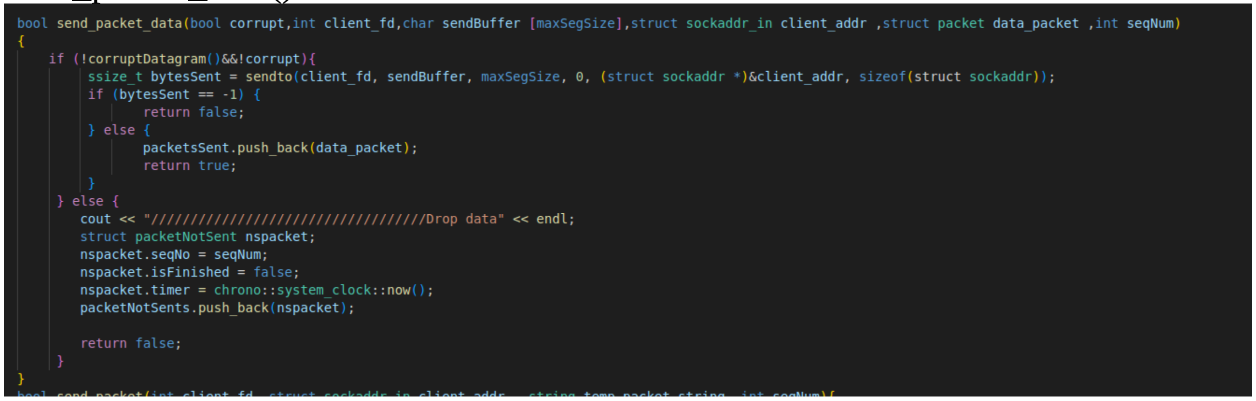


```
                    int ack_seqNo = ack->ackno;
                    if (LastAckedSeqNum == ack_seqNo){
                        NumberOfDupAcks++;
                        SentPacketsNotAcked--;
                        if (St == fastRecovery){
                            Cwnd++;
                            //cout << "CWND : " << cwnd << " maxSegSize "<< endl << flush;
                            // Write to the file
                            myFile_Handler << Cwnd << endl;
                        } else if (NumberOfDupAcks == 3){
                            Sst = Cwnd / 2;
                            Cwnd = Sst + 3;
                            cout << "/////////////////////////////////////////////////////////////////// Triple duplicate Ack ///////////////////////////////////
                            //cout << "CWND : " << cwnd << " maxSegSize "<< endl << flush;
                            // Write to the file
                            myFile_Handler << Cwnd << endl;
                            St = fastRecovery;
                            /** retransmit the lost packet **/
                            SeqNum = ack_seqNo;
                            bool found = false;
                            found = retransmit_loss_packet( found, client_fd,  client_addr , data);
                            /** handle checksum error **/
                            if (!found){
                                handle_check_sum( found, client_fd,  client_addr , data);
                            }

                        }
                        //cout << "CWND : " << cwnd << " maxSegSize "<< endl << flush;

                    } else if (LastAckedSeqNum < ack_seqNo) {
                        /** new ack : compute new base and packet no. and handling congestion control FSM **/
                        cout << "newAck " << endl;
                        NumberOfDupAcks = 0;
                        LastAckedSeqNum = ack_seqNo;
                        int advance = LastAckedSeqNum - Base_packet_number;
                        Cwnd_base = Cwnd_base - advance;
                        Base_packet_number = LastAckedSeqNum;
                        if (St == slowStart){
                           if (Cwnd*2 >= Sst){
                                St = congestionAvoidance;
                                Cwnd++;
```



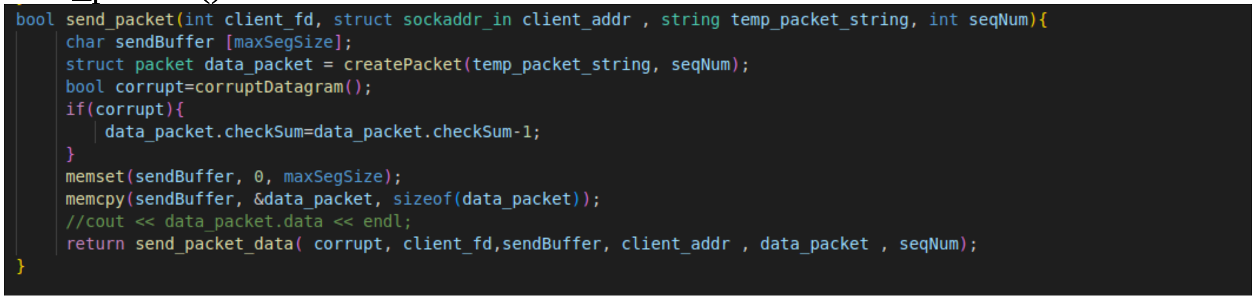
```
                            // Write to the file
                            myFile_Handler << Cwnd << endl;
                        }
                        //cout << "CWND : " << cwnd << " maxSegSize "<< endl << flush;
                        //cout << " base : " << cwnd_base << endl << flush;
                        //cout << " packet : " << base_packet_number << endl << flush;
                        SentPacketsNotAcked--;
                    } else {
                        SentPacketsNotAcked--;
                        //cout << "CWND : " << cwnd << " maxSegSize "<< endl << flush;
                    }

                    if (SentPacketsNotAcked == 0){
                        //cout << "CWND : " << cwnd << " maxSegSize "<< endl << flush;
                        StillExistAcks = false;
                    }

                }

            }

        }
        /** Handle Time Out **/
        bool entered=false;
        entered = handle_time_out(client_fd,  client_addr , data);
        if(entered){
            entered=false;
            Cwnd=1;
            //cwnd_base=0;
            St = slowStart;
            //cout << "CWND : " << cwnd << " maxSegSize "<< endl << flush;
            // Write to the file
            myFile_Handler << Cwnd << endl;
        }

    }

    // File Close
    myFile_Handler.close();
}
```

send_packet_data()



```
bool send_packet_data(bool corrupt,int client_fd,char sendBuffer [maxSegSize],struct sockaddr_in client_addr ,struct packet data_packet ,int seqNum)
{
    if (!corruptDatagram()&&!corrupt){
        ssize_t bytesSent = sendto(client_fd, sendBuffer, maxSegSize, 0, (struct sockaddr *)&client_addr, sizeof(struct sockaddr));
        if (bytesSent == -1) {
            return false;
        } else {
            packetsSent.push_back(data_packet);
            return true;
        }
     } else {
        cout << "///////////////////////////////////Drop data" << endl;
        struct packetNotSent nspacket;
        nspacket.seqNo = seqNum;
        nspacket.isFinished = false;
        nspacket.timer = chrono::system_clock::now();
        packetNotSents.push_back(nspacket);

        return false;
     }
}
```

send_packet ()

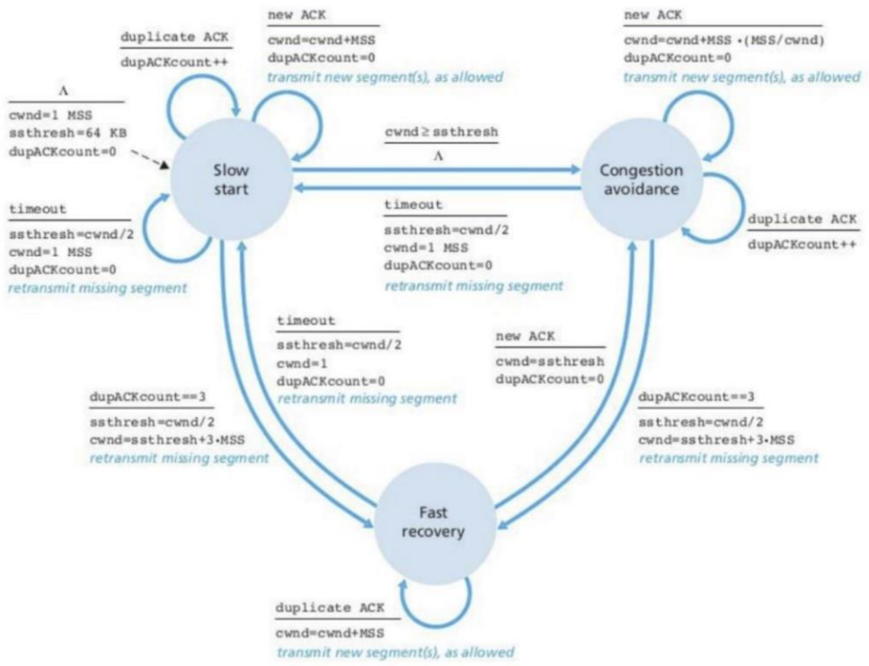


```
bool send_packet(int client_fd, struct sockaddr_in client_addr , string temp_packet_string, int seqNum){
    char sendBuffer [maxSegSize];
    struct packet data_packet = createPacket(temp_packet_string, seqNum);
    bool corrupt=corruptDatagram();
    if(corrupt){
        data_packet.checkSum=data_packet.checkSum-1;
    }
    memset(sendBuffer, 0, maxSegSize);
    memcpy(sendBuffer, &data_packet, sizeof(data_packet));
    //cout << data_packet.data << endl;
    return send_packet_data( corrupt, client_fd,sendBuffer, client_addr , data_packet , seqNum);
}
```

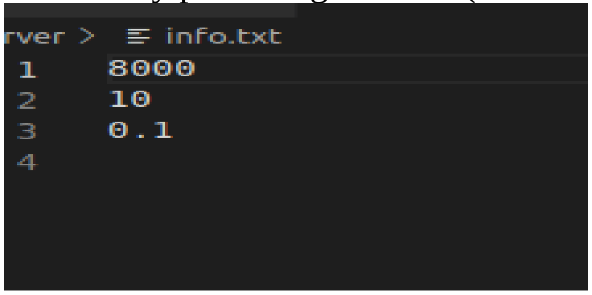
Design Decisions Server :

- Multiple users the server forks off a child process to handle the client.
- The server (child) creates a UDP socket to handle file transfer to the client.
- Handling client requests as follows :
  -The client sends the name of file that he wants the server to send to him.
  - when the server receives the file name checks if file is available if not it appear error message

  - if file is available the server send the number of packet need to client and transmission will be as following:



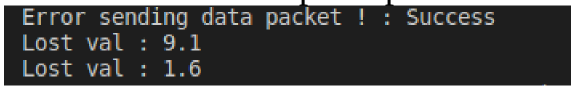
- To set the Arguments of the server we make commands file (info.txt)
  Where we put the following arguments :
  Well - known port number .
  Random generator seed value .
  Probability p of datagram loss (real number in the range [0.0,1.0]).



```
rver > ☰ info.txt
8000
10
0.1

```

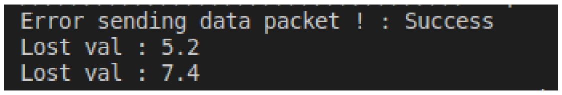
- Simulating corrupted packets we subtract one from the check sum of the packet so that client knows that the packet is corrupted.

  In first one corruption packet as lost val > 5.9
  In second one Corruption packet as lost <5.9



```
Error sending data packet ! : Success
Lost val : 9.1
Lost val : 1.6
```

- Simulating lost packets the server does not send them so , no ack for them from the client will be received
  In first one corruption packet as lost val < 5.9
  In second one Corruption packet as lost >5.9



```
Error sending data packet ! : Success
Lost val : 5.2
Lost val : 7.4
```

For the Congestion Window:

- The CWND increasing exponentially in the stage of slow start .
- When CWND value is more than the value of ssthresh which we set initially 128 , it starts to increase linearly in the stage of congestion window avoidance.
- When there is packet lost the CWND is drop to equal 1 and start increasing again.
- When there is triple duplicate Ack the CWND size drop to half of its size and the ssthresh will equal the half of CWND just before losing the packet.

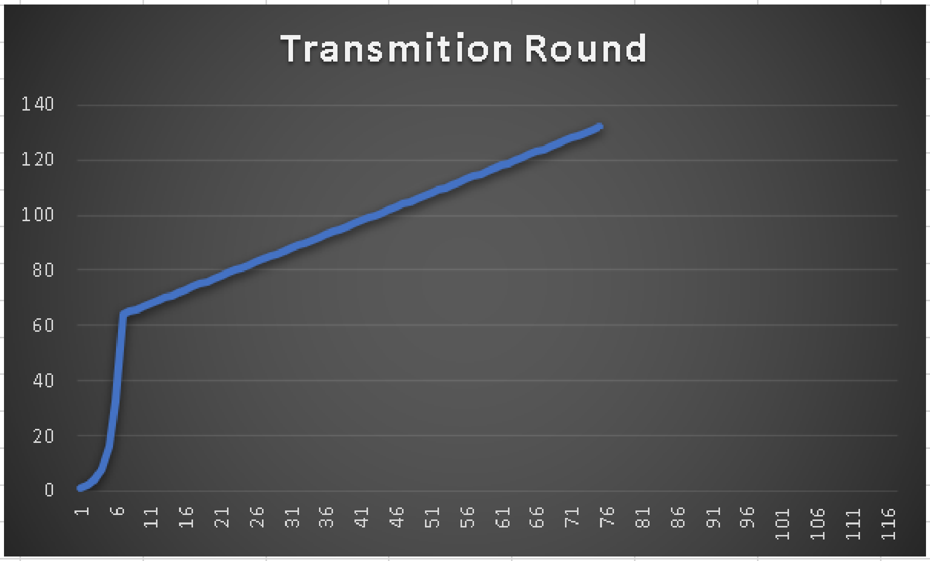
Network System analysis:

Points of Comparison	Stop -AND-WAIT	Selective Repeat
Basic	Retransmits all the frames after frame suspect to be lost	Retransmits only frames that are suspect to be lost
Complexity	Less complicated	More Complex

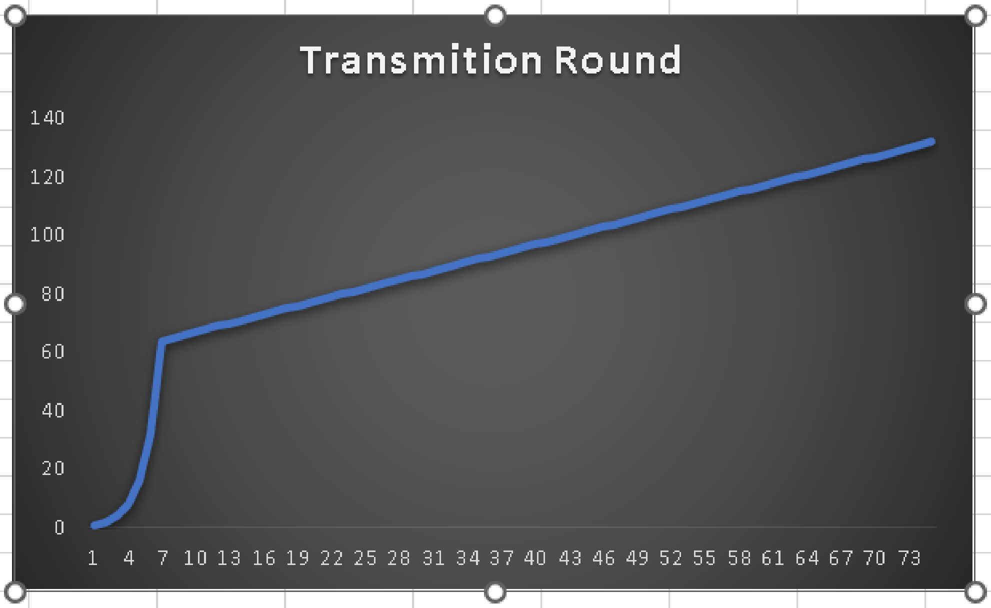
Window Size	N-1	<= (N+1)/2
Sorting	Sorting is neither need in sender or receiver	Receiver must be able sort .
Storing	Receiver do not stored frames after damaged until the frame is retransmit	Receiver stores the frames received after damaged frame is replaced.
Searching	No searching	Sender must be able to search.
Bandwidth Utilization	If error rate is high, it waste bandwidth	Less bandwidth is wasted In retransmission

Congestion Window Analysis:

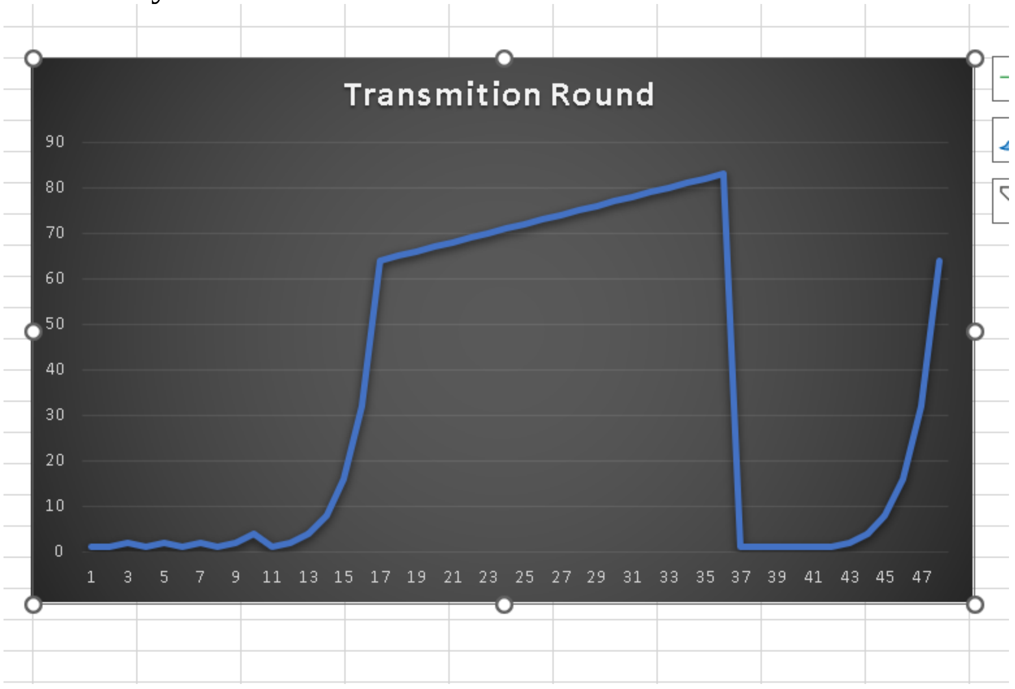
Probability is 0.01:



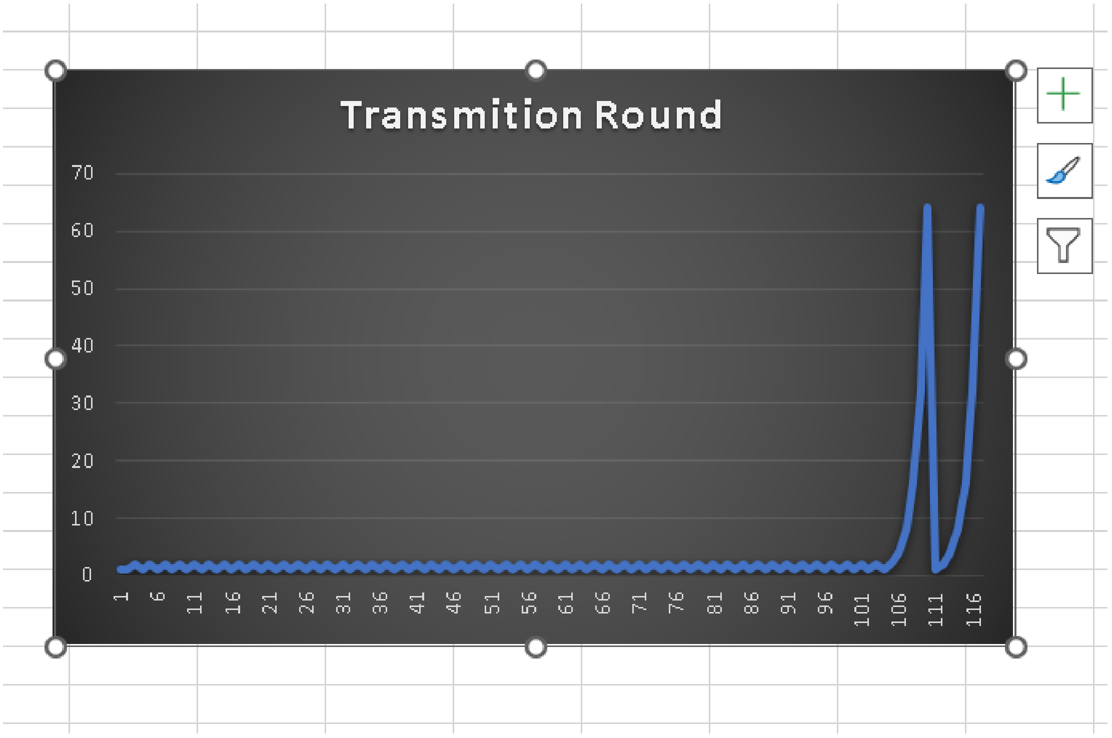
Probability is 0.05:



Probability is 0.1:



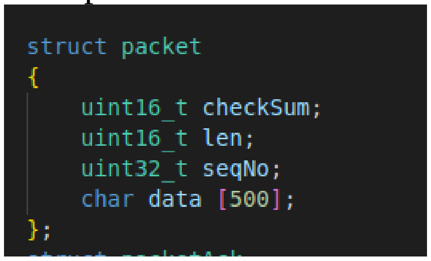
Probability is 0.3:



Packet type and fields (Client) :

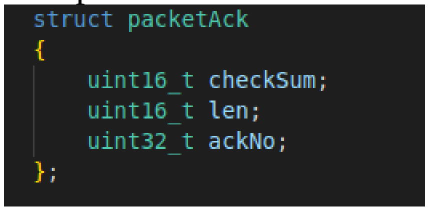
- There are two kinds of packets:

  Data packets :



```
struct packet
{
    uint16_t checkSum;
    uint16_t len;
    uint32_t seqNo;
    char data [500];
};
```

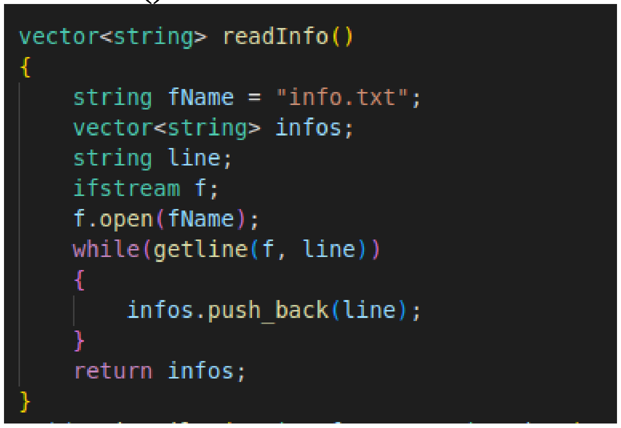
Ack packets:



```
struct packetAck
{
    uint16_t checkSum;
    uint16_t len;
    uint32_t ackNo;
};
```

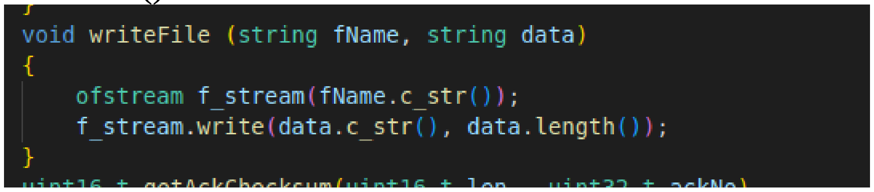
Main functions Client:

readInfo ()



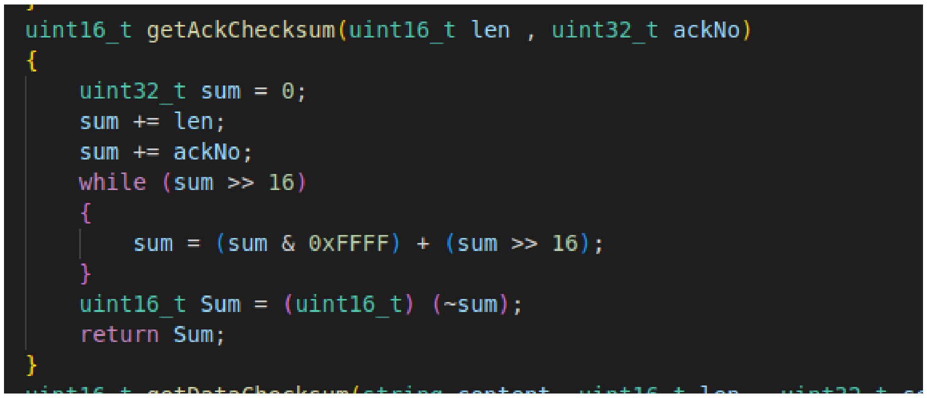
```
vector<string> readInfo()
{
    string fName = "info.txt";
    vector<string> infos;
    string line;
    ifstream f;
    f.open(fName);
    while(getline(f, line))
    {
        infos.push_back(line);
    }
    return infos;
}
```

writefile ()



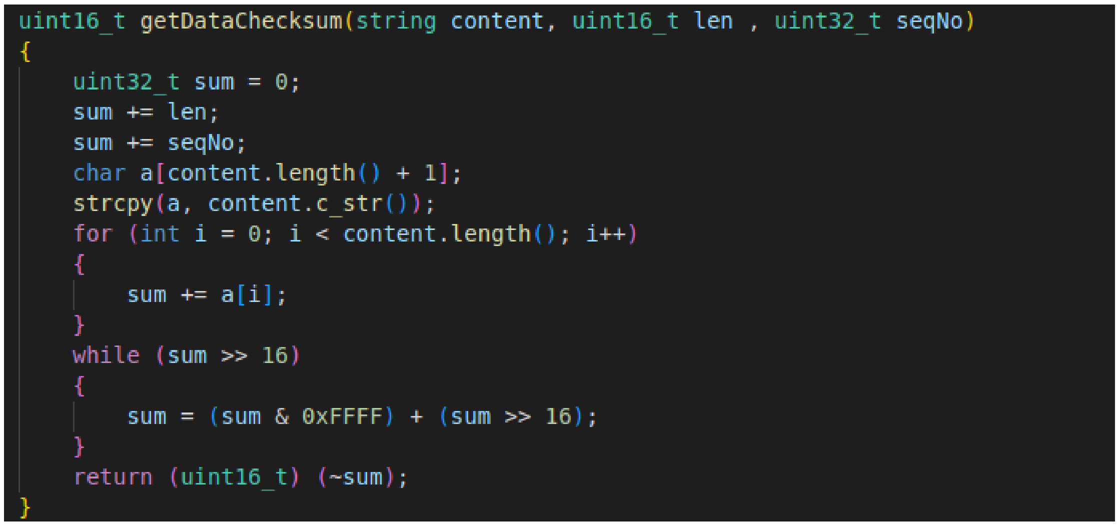
```
void writeFile (string fName, string data)
{
    ofstream f_stream(fName.c_str());
    f_stream.write(data.c_str(), data.length());
}
```

getAckChecksum ()



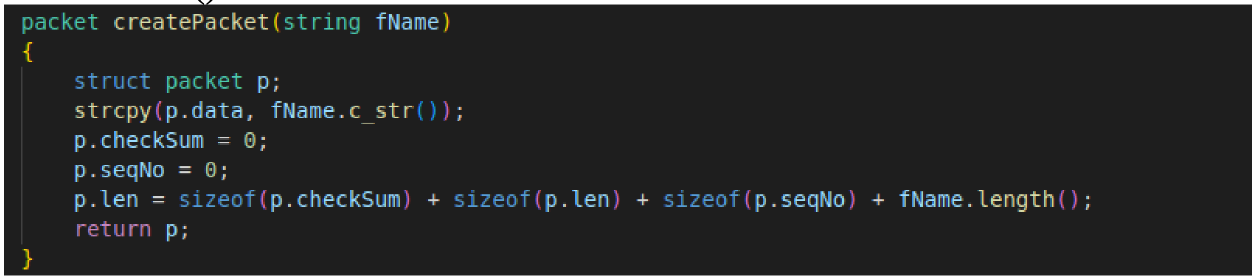
```
uint16_t getAckChecksum(uint16_t len , uint32_t ackNo)
{
    uint32_t sum = 0;
    sum += len;
    sum += ackNo;
    while (sum >> 16)
    {
        sum = (sum & 0xFFFF) + (sum >> 16);
    }
    uint16_t Sum = (uint16_t) (~sum);
    return Sum;
}
```

getDataChecksum ()



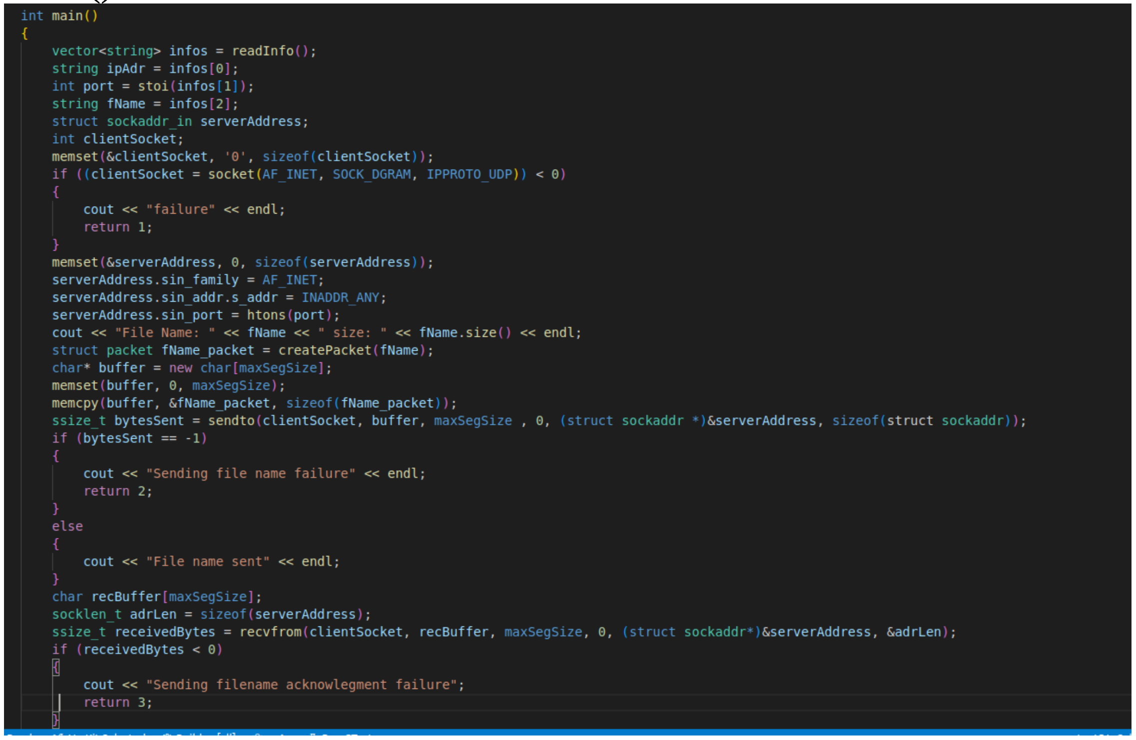
```
uint16_t getDataChecksum(string content, uint16_t len , uint32_t seqNo)
{
    uint32_t sum = 0;
    sum += len;
    sum += seqNo;
    char a[content.length() + 1];
    strcpy(a, content.c_str());
    for (int i = 0; i < content.length(); i++)
    {
        sum += a[i];
    }
    while (sum >> 16)
    {
        sum = (sum & 0xFFFF) + (sum >> 16);
    }
    return (uint16_t) (~sum);
}
```

createPacket()

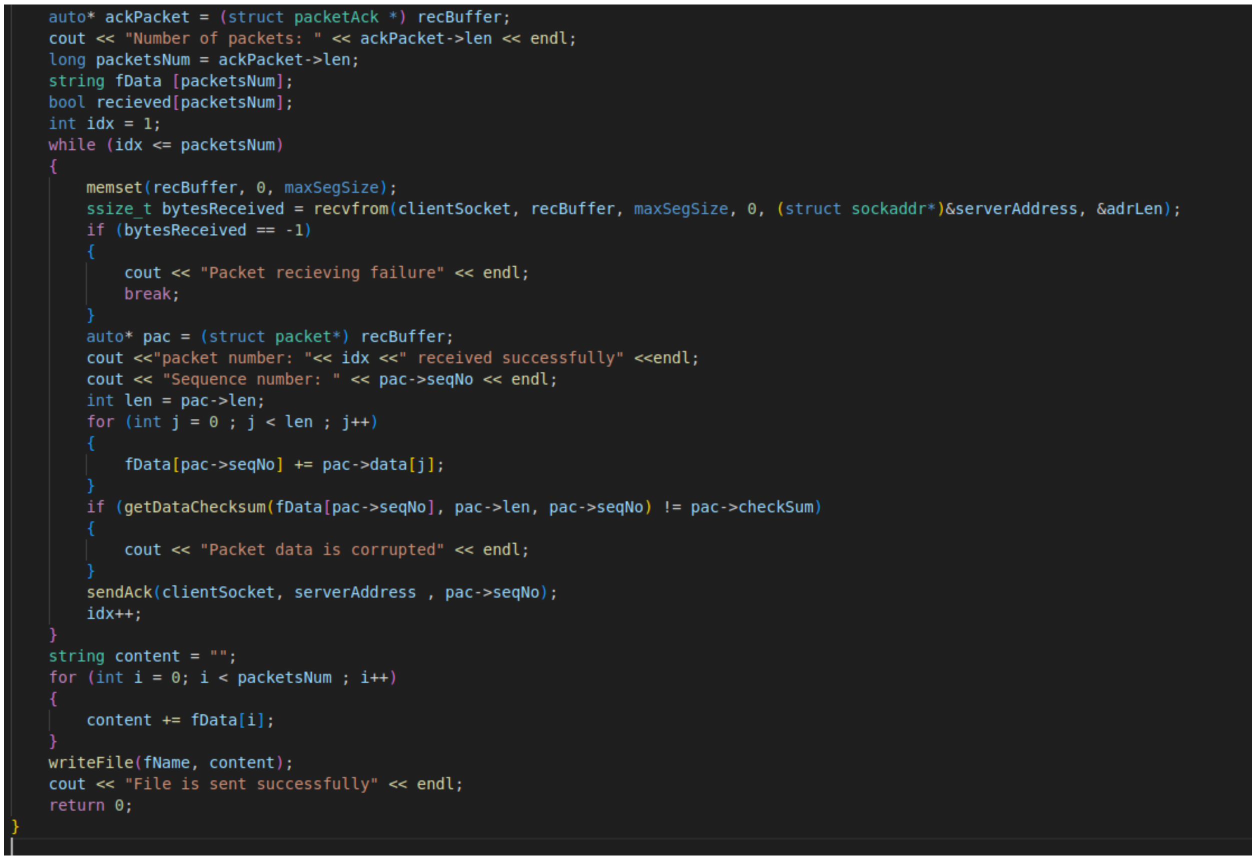


```
packet createPacket(string fName)
{
    struct packet p;
    strcpy(p.data, fName.c_str());
    p.checkSum = 0;
    p.seqNo = 0;
    p.len = sizeof(p.checkSum) + sizeof(p.len) + sizeof(p.seqNo) + fName.length();
    return p;
}
```

main ()



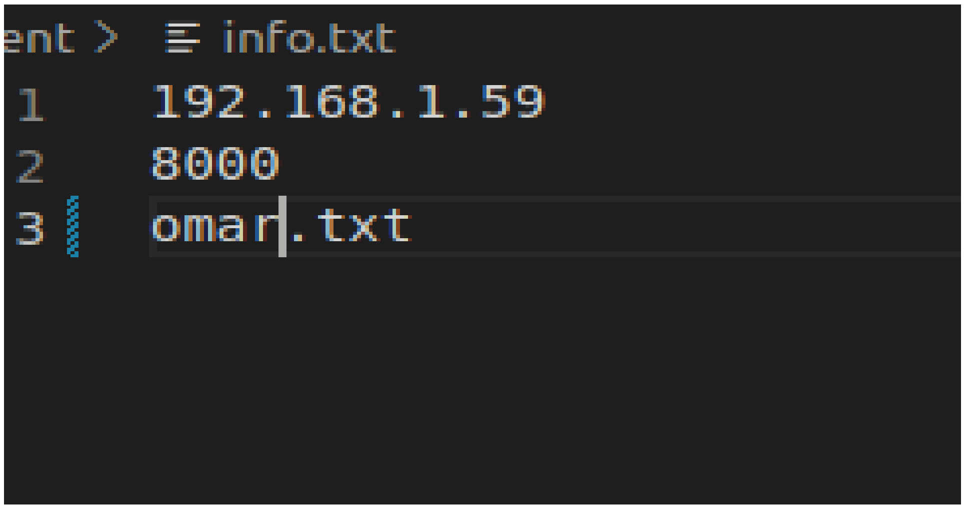
```cpp
int main()
{
    vector<string> infos = readInfo();
    string ipAdr = infos[0];
    int port = stoi(infos[1]);
    string fName = infos[2];
    struct sockaddr_in serverAddress;
    int clientSocket;
    memset(&clientSocket, '0', sizeof(clientSocket));
    if ((clientSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
    {
        cout << "failure" << endl;
        return 1;
    }
    memset(&serverAddress, 0, sizeof(serverAddress));
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_addr.s_addr = INADDR_ANY;
    serverAddress.sin_port = htons(port);
    cout << "File Name: " << fName << " size: " << fName.size() << endl;
    struct packet fName_packet = createPacket(fName);
    char* buffer = new char[maxSegSize];
    memset(buffer, 0, maxSegSize);
    memcpy(buffer, &fName_packet, sizeof(fName_packet));
    ssize_t bytesSent = sendto(clientSocket, buffer, maxSegSize , 0, (struct sockaddr *)&serverAddress, sizeof(struct sockaddr));
    if (bytesSent == -1)
    {
        cout << "Sending file name failure" << endl;
        return 2;
    }
    else
    {
        cout << "File name sent" << endl;
    }
    char recBuffer[maxSegSize];
    socklen_t adrLen = sizeof(serverAddress);
    ssize_t receivedBytes = recvfrom(clientSocket, recBuffer, maxSegSize, 0, (struct sockaddr*)&serverAddress, &adrLen);
    if (receivedBytes < 0)
    {
        cout << "Sending filename acknowlegment failure";
        return 3;
    }
```



```
    auto* ackPacket = (struct packetAck *) recBuffer;
    cout << "Number of packets: " << ackPacket->len << endl;
    long packetsNum = ackPacket->len;
    string fData [packetsNum];
    bool recieved[packetsNum];
    int idx = 1;
    while (idx <= packetsNum)
    {
        memset(recBuffer, 0, maxSegSize);
        ssize_t bytesReceived = recvfrom(clientSocket, recBuffer, maxSegSize, 0, (struct sockaddr*)&serverAddress, &adrLen);
        if (bytesReceived == -1)
        {
            cout << "Packet recieving failure" << endl;
            break;
        }
        auto* pac = (struct packet*) recBuffer;
        cout <<"packet number: "<< idx <<" received successfully" <<endl;
        cout << "Sequence number: " << pac->seqNo << endl;
        int len = pac->len;
        for (int j = 0 ; j < len ; j++)
        {
            fData[pac->seqNo] += pac->data[j];
        }
        if (getDataChecksum(fData[pac->seqNo], pac->len, pac->seqNo) != pac->checkSum)
        {
            cout << "Packet data is corrupted" << endl;
        }
        sendAck(clientSocket, serverAddress , pac->seqNo);
        idx++;
    }
    string content = "";
    for (int i = 0; i < packetsNum ; i++)
    {
        content += fData[i];
    }
    writeFile(fName, content);
    cout << "File is sent successfully" << endl;
    return 0;
}
```

Design Decisions Client :

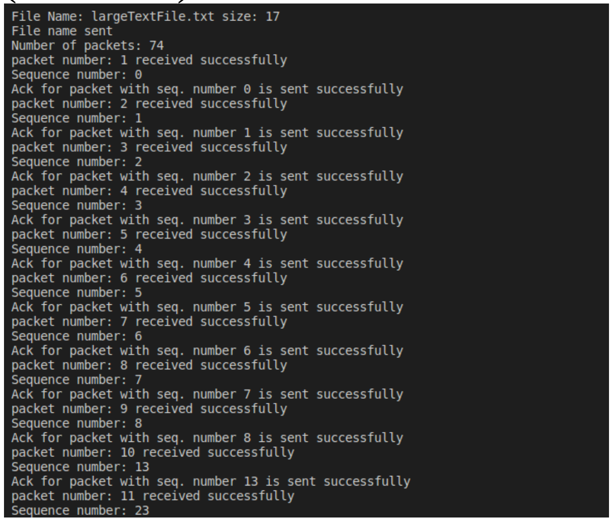
- To set the Arguments of the server we make commands file (info.txt)
  Where we put the following arguments :
  IP address of server .
  Well known port number in server .
  Filename to be transferred (large file).



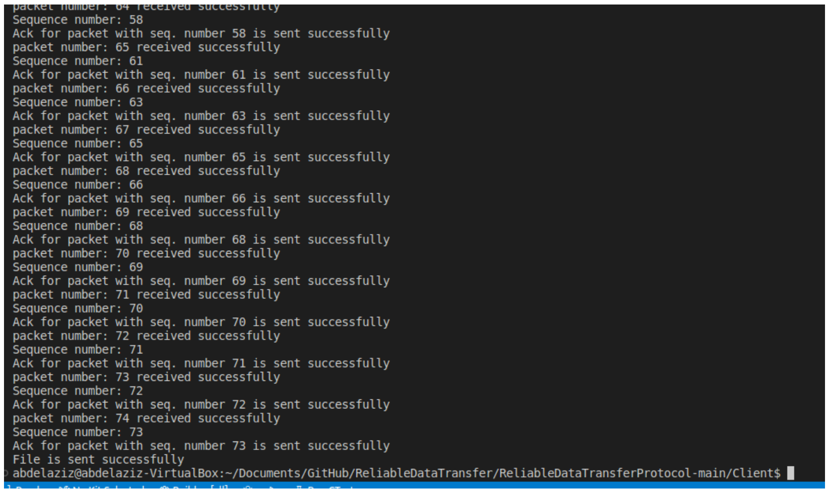
```
192.168.1.59
8000
omar.txt
```

Sample Runs :

(Client side)

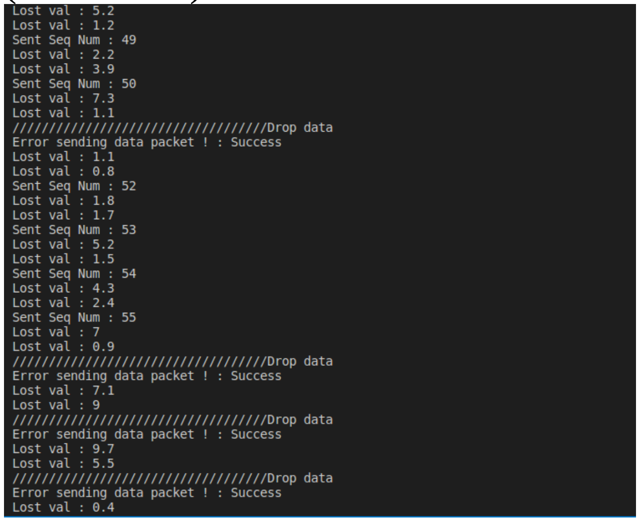


```
File Name: largeTextFile.txt size: 17
File name sent
Number of packets: 74
packet number: 1 received successfully
Sequence number: 0
Ack for packet with seq. number 0 is sent successfully
packet number: 2 received successfully
Sequence number: 1
Ack for packet with seq. number 1 is sent successfully
packet number: 3 received successfully
Sequence number: 2
Ack for packet with seq. number 2 is sent successfully
packet number: 4 received successfully
Sequence number: 3
Ack for packet with seq. number 3 is sent successfully
packet number: 5 received successfully
Sequence number: 4
Ack for packet with seq. number 4 is sent successfully
packet number: 6 received successfully
Sequence number: 5
Ack for packet with seq. number 5 is sent successfully
packet number: 7 received successfully
Sequence number: 6
Ack for packet with seq. number 6 is sent successfully
packet number: 8 received successfully
Sequence number: 7
Ack for packet with seq. number 7 is sent successfully
packet number: 9 received successfully
Sequence number: 8
Ack for packet with seq. number 8 is sent successfully
packet number: 10 received successfully
Sequence number: 13
Ack for packet with seq. number 13 is sent successfully
packet number: 11 received successfully
Sequence number: 23
```

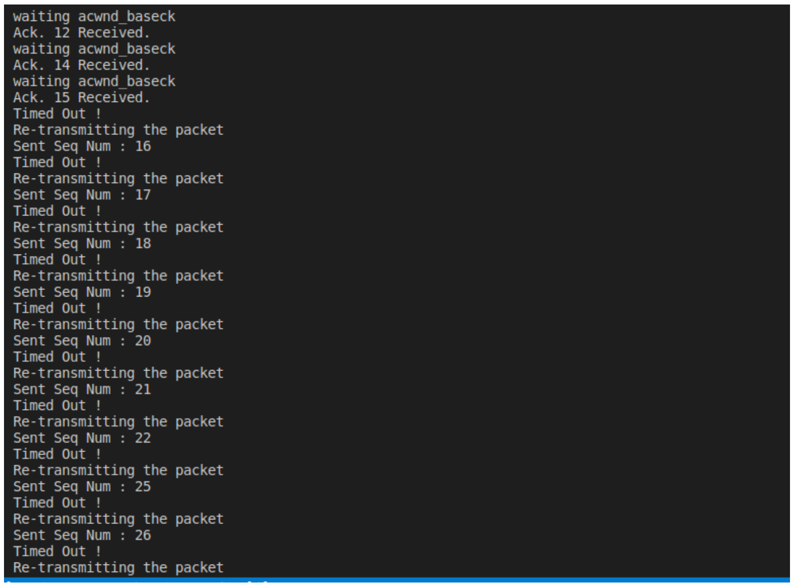


```
packet number: 64 received successfully
Sequence number: 58
Ack for packet with seq. number 58 is sent successfully
packet number: 65 received successfully
Sequence number: 61
Ack for packet with seq. number 61 is sent successfully
packet number: 66 received successfully
Sequence number: 63
Ack for packet with seq. number 63 is sent successfully
packet number: 67 received successfully
Sequence number: 65
Ack for packet with seq. number 65 is sent successfully
packet number: 68 received successfully
Sequence number: 66
Ack for packet with seq. number 66 is sent successfully
packet number: 69 received successfully
Sequence number: 68
Ack for packet with seq. number 68 is sent successfully
packet number: 70 received successfully
Sequence number: 69
Ack for packet with seq. number 69 is sent successfully
packet number: 71 received successfully
Sequence number: 70
Ack for packet with seq. number 70 is sent successfully
packet number: 72 received successfully
Sequence number: 71
Ack for packet with seq. number 71 is sent successfully
packet number: 73 received successfully
Sequence number: 72
Ack for packet with seq. number 72 is sent successfully
packet number: 74 received successfully
Sequence number: 73
Ack for packet with seq. number 73 is sent successfully
File is sent successfully
abdelaziz@abdelaziz-VirtualBox:~/Documents/GitHub/ReliableDataTransfer/ReliableDataTransferProtocol-main/Client$ 
```

(Server Side)



```
Lost val : 5.2
Lost val : 1.2
Sent Seq Num : 49
Lost val : 2.2
Lost val : 3.9
Sent Seq Num : 50
Lost val : 7.3
Lost val : 1.1
//////////////////////////////////Drop data
Error sending data packet ! : Success
Lost val : 1.1
Lost val : 0.8
Sent Seq Num : 52
Lost val : 1.8
Lost val : 1.7
Sent Seq Num : 53
Lost val : 5.2
Lost val : 1.5
Sent Seq Num : 54
Lost val : 4.3
Lost val : 2.4
Sent Seq Num : 55
Lost val : 7
Lost val : 0.9
//////////////////////////////////Drop data
Error sending data packet ! : Success
Lost val : 7.1
Lost val : 9
//////////////////////////////////Drop data
Error sending data packet ! : Success
Lost val : 9.7
Lost val : 5.5
//////////////////////////////////Drop data
Error sending data packet ! : Success
Lost val : 0.4
```



```
waiting acwnd_baseck
Ack. 12 Received.
waiting acwnd_baseck
Ack. 14 Received.
waiting acwnd_baseck
Ack. 15 Received.
Timed Out !
Re-transmitting the packet
Sent Seq Num : 16
Timed Out !
Re-transmitting the packet
Sent Seq Num : 17
Timed Out !
Re-transmitting the packet
Sent Seq Num : 18
Timed Out !
Re-transmitting the packet
Sent Seq Num : 19
Timed Out !
Re-transmitting the packet
Sent Seq Num : 20
Timed Out !
Re-transmitting the packet
Sent Seq Num : 21
Timed Out !
Re-transmitting the packet
Sent Seq Num : 22
Timed Out !
Re-transmitting the packet
Sent Seq Num : 25
Timed Out !
Re-transmitting the packet
Sent Seq Num : 26
Timed Out !
Re-transmitting the packet
```