Networks

**RELIABLE DATA TRANSFER**

ASSIGNMENT 2

horizontal line

Alaa Samir Shehab  
ID (1)

## Requirements

## Server

### Overall implementation

* Create server socket
* Bind socket with Address
* Receive data using recvFrom from a client
* Fork a child to handle this client through reading the file name found in packet sent by the client, and sending back the number of packets expected.
* Start transferring packets to client using (checksum, congestion control, SR) principles.

## Client

### Overall implementation

* Create client socket
* Create a packet with the name of the file requested and sends the packet to the server
* Waits for the acknowledgement of this packet to determine the number of packets expected.
* Starts receiving packets from server and sending acks back.

## SR

### Server side

* New maps (one which contains the sent packets and the other for the received acks) are created
* Window base, window size and congestion window are initialized
* The lost packets are chosen and added to a new list
* 2 threads are created one for sending data packets and one for receiving acks from client.
  + Send data
    - Loop infinitely till all packets are sent, loop on all packets in current window and either send the packet to client or ignore if it a lost packet then if packet was already sent check timeout of the packet.
  + Recv acks
    - Check if an ack is sent then update congestion window and window base.

### Client side

* Loop infinitely until all packets are received.
* On receiving a new packet check if corrupt, if not then add packet to buffer and send ack to server.
* After all the packets are received the data is written to the file.

## Functions

*Calculate\_checksum*: adds data of packet, the seqno then add carry through 16 bit shifts and then returning the complement to get (1’s complement)

*Get\_data\_from\_file:* read a chunk of data from the file by using SEEK, so first set the file pointer to the position at which reading will start (number of the packet \* 500) then read a length of 500 bytes.

*Ack\_corrupt:* reads the checksum value of the packet and recalculates the checksum and returns true if they are not equal

*Fill\_lost\_packets:* find the number of lost packets using the plp parameter then randomly choose n (lost packets) numbers to be lost.

*Send\_data:* loop on window, check if packet is not sent then send it and add it to the map, if packet is from the lost ones then only add it to map without sending it and in both cases start the timer. If packet is already sent then check if it timed out.

## Data structures

#### Packet:

* Ack packet -> similar to what is defined in the assignment pdf
* Data packet -> similar to what is defined in the assignment pdf
* Server ack packet -> is the first packet sent by the server to client to set the expected number of packets that will be sent.

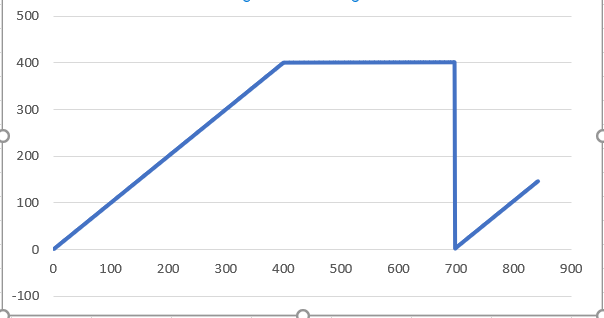
#### Map:

* Map for packets received on client side
* Map for packets sent on server side
* Map for acks sent by client on server side

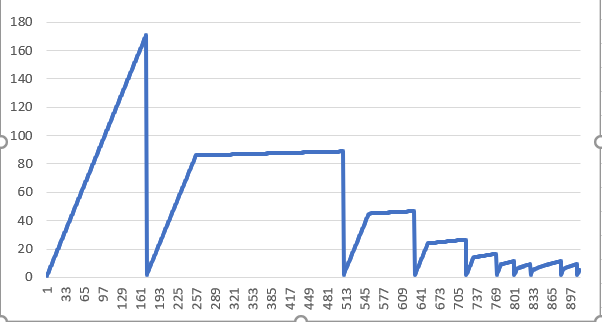
## Testing examples

* File size - 228370
* Number of packets - 457
* More packets could be lost other then the expected number due to real network congestion

PLP = 0 (0 packets lost)



PLP = 0.1 (46 packets lost)



PLP = 0.2 (92 packets lost)

