README FILE: Question 1
Author: Ayush Jain
BITS ID: 2017A7PS0093P

Execution Instructions:

1. In directory q1, write **make** in the terminal.
2. Open two terminals in the same directory. First execute ./server in one terminal. In 2nd terminal please execute ./client.

Description of Files in Directory.

1. Client.c : code for client.
2. Server.c: code for server
3. Packet.h: Header file for packet.
4. Stop_and_wait.h: Header for client.c and server.c
5. Makefile
6. Input.txt: A large text file, to be transferred by client to server.
7. Input1.txt: A smaller subset of input.txt
8. Clientlog.txt: Log of terminal when ./client is executed
9. Serverlog.txt: Log of terminal when ./server is executed
10. Client.o: Binary for direct execution using ./client
11. Server.o : Binary for direct execution using ./server
12. Output.txt: file at server after transmission

Important #defines.

PACKET.h

1. #define PACKET_SIZE : modify this for changing the size of payload of packet (ignoring the other fields). This specifies the number of bytes of data transferred.

2. #define timeout: Amount of time after the packet timeouts if ACK is not received.

Client.c

1. #define PORT: refers to the server port to connect to.

Server.c

1. #define PORT: refers to server port which accepts connection from client.
2. #define PDR: Packet drop rate. Must be between 0 and 1. 0.1 means 10% drop rate.
3. #define BUFFER_SIZE: Size of a small buffer at server side. Its value is presently initialised to be 10 (payloads). Packets data are temporarily stored in a buffer. It is written to file, when the buffer is full, and has all the packets from 1-10 in order.

Execution status: Able to transfer a big file of approximately 150000 bytes. Tested correctly with different values of packet drop rates.

KEY DESIGN DECISIONS

1. Packets are dropped at server side with PDR probability. A message is shown “OOPS, Packet dropped at server” for convenience of tracking the execution of the code.
2. Multi channel implementation: Since it is a TCP communication, every channel needs its own connection with the server port. Hence on the client side, two TCP sockets. Server has initially a master socket on which it will be listening for connections. Each socket of the client sends a connection request to the server, which server accepts. While accepting it makes a new port for each of the connections, while the master socket is listening for further connections if any.

The connection and data transfer are dealt using select. We make use of the fact that if some data is sent to the master socket, it must

be for accepting a new connection. Hence a new socket is made and added to select so that we can keep track of any data transfer on the newly created socket. If data is received on some other socket, it must be file data which is appropriately stored in the buffer temporarily. This is done to handle out of order packet conditions.

3. Timeouts: Individual timeouts on each packet are not done. Instead a single timer for both the channels is implemented.
4. Packet dropout: If a packet is dropped, ack is not sent by the server. After the specified timeout period, the client resends the packets.
5. Out of order packets outside buffer size are simply dropped.