COMPUTER NETWORKS (CS F303)
SECOND SEMESTER 2019-20
Programming Assignment

Problem 1: File transfer using multi-channel stop-and-wait protocol

Consider the following modified version of stop-and-wait protocol.

Multi-channel stop-and-wait protocol without DATA or ACK losses.

1. The sender transmits packets through two different channels (TCP connections).
2. The server acknowledges the receipt of a packet via the same channel through which the corresponding packet has been received.
3. The sender transmits a new packet using the same channel on which it has received an ACK for its one of the previously transmitted packet. Note that, at a time, there can be at most two outstanding unacknowledged packets at the sender side.
4. On the server-side, the packets transmitted through different channels may arrive out of order. In that case, the server has to buffer the packets temporarily to finally construct in-order data stream.

Write client and server programs to upload a given file **("input.txt"**) from client to the server using ARQ protocol as described above by making TCP connections between the client and the server.

Your program MUST include following features.

1. The complete file should be divided into equal size chunks and each chunk should be transmitted separately using stop and wait for protocol by encapsulating it in the form of a packet structure. The size of each chunk should be defined as #define **PACKET_SIZE** macro. For submission purposes, the value of PACKET_SIZE should be kept as 100. However, while evaluating your code, we will test it for other values as well. Also, handle the situation when the file size is not multiple of PACKET_SIZE.

2. In addition to the payload, the packet structure should contain following information in form of a header.
    a. The size (number of bytes) of the payload
    b. The Seq. No. (in terms of byte) specifying the offset of the first byte of the packet with respect to the input file.
    c. Whether the packet is last packet or not?
    d. The packet is DATA or ACK. In this way, you can utilize the same packet structure for both DATA send by the client and ACK send by the server. The Seq. No. field in the ACK packet would correspond to it DATA packet with the same Seq. No. value received from the client.
    e. The channel information specifying the Id (either 0 or 1) of the channel through which the packet has been transmitted.

3. The packet loss should be implemented at the server. The server will randomly drop a received packet and would not send ACK back to the client. The packet drops rate (PDR) should again be defined as #define macro, and its value can be kept as 10% for the submission purpose. Again, while evaluating, we may test the working of your program for different values of PDR.

4. No need to implement ACK losses.

5. The client should handle the retransmission of lost packets. You can either use timers for this purpose or the "poll" system call. Again the duration of retransmission time should be defined as macro and can be as two seconds for the submission purpose. You should have kept a copy of the transmitted packet to facilitate re-transmission, instead of re-constructing a new packet from the input file again. Assume the ACKs are not lost.

6. The server program should implement a mechanism to handle the packets received out of order. You **should not** store the packets in a temporary buffer and write the entire buffer to the file at the end. Instead, directly write the received packets in the output file as and when they are received. However, a small amount of buffering would be required to handle out of order reception.

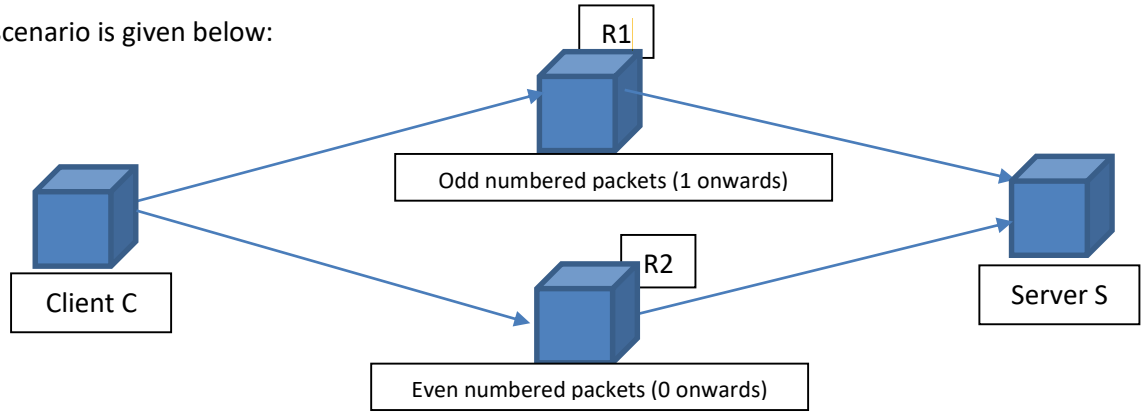
7. The client and server should produce the traces in the following format.

Client Trace	**Server Trace**
SENT PKT: Seq. No ---- of size ---- Bytes from channel ---	RCVD PKT: Seq. No ---- of size --- Bytes from channel ---
RCVD ACK: for PKT with Seq. No. --- from channel ---	SENT ACK: for PKT with Seq. No. --- from channel ---

Problem 2: File transfer using Selective Repeat protocol over UDP

Write client and server programs to upload a given file **("input.txt"**) from client to the server in a given scenario by implementing a reliable connection on top of UDP communication which uses Selective Repeat protocol.

The scenario is given below:



C uploads input.txt to S. All odd-numbered packets go through the relay node R1, while all even-numbered packets go through the relay node R2. R1 and R2 add a delay, which is a random number distributed uniformly between 0 to 2 ms for a given packet. Acknowledgments can take any route and **do not** suffer from delays or packet drops.

Your program MUST include the following features.

1. The complete file should be divided into equal-size chunks, and each chunk should be transmitted separately using SR protocol by encapsulating it in the form of a packet structure. The size of each chunk should be defined as #define **PACKET_SIZE** macro. For submission purpose, the value of PACKET_SIZE should be kept as 100 bytes. However, while evaluating your code, we shall test it for other values also. For simplicity, assume that file size is a multiple of PACKET_SIZE.

2. In addition to the payload, the packet structure should contain the following information in the form of a header.
    a. The size (number of bytes) of the payload
    b. Whether the packet is the last packet or not?
    c. You should also decide how you wish to implement and keep track of sequence numbers and ACKs, and include the necessary information in the header (or in the packet structure). The packet structure should be declared in a separate header file "packet.h". Both client and the server programs will make use of this structure.

3. The packet loss should be implemented at R1 and R2. The packet drops rate (PDR) should be defined as #define macro, and its value can be kept as 10% for the submission purpose. Again, while evaluating, we may test the working of your program for different values of PDR.
4. The client should handle the retransmission of lost packets. You can either use timers for this purpose or the "poll" system call. The value of the timer at the client should be defined as a macro and it should be more than the worst-case delay experienced by any packet in the network (so that the timer doesn't expire prematurely). You should keep a copy of the transmitted packet to facilitate retransmission instead of reconstructing a new packet from the input file again. Assume the ACKs are not lost.

5. The server should handle out-of-order delivery of packets, their reassembly, and be able to receive the entire file input.txt.

6. You may buffer the received packets (only payload) in an array of characters. However, you should assume that the buffer array is NOT large enough to store the entire input.txt (assume input.txt to be at least 10 times larger than the buffer). From time-to-time, you will need to write the contents to disk and empty the buffer. However, this should also handle out-of-order delivery.

7. Generate and write the log event using following format:

Node Name	Event Type	Timestamp	Packet Type	Seq. No	Source	Dest

Node Name: Name of the node where the event has happened
Possible values: CLIENT, SERVER, RELAY1, RELAY2
Event Type: S (SEND), R (RECV) , D (DROP), TO (TIME OUT), RE (Retransmission)

The relay nodes should also log the details when they receive as well as send a packet
Timestamp: The time of the event. Since all four processes would be running on the same machine. Later when we will sort the combined log based on the timestamp a clearer picture of the events can be seen.
Packet Type: DATA, ACK
Seq No: Sequence number of the DATA and the ACK packet.
Source: The transmitter of the packet over a link
Possible Values (CLIENT, SERVER, RELAY1, RELAY2)
Dest. The receiver of the packet **over a link**
Possible Values (CLIENT, SERVER, RELAY1, RELAY2)
Example of log data:

	Node Name	Event Type	Timestamp	Packet Type	Seq. No	Source	Dest
A packet sent event at CLIENT	CLIENT	S	01:35:20.312554	DATA	0	CLIENT	RELAY2
Reception of a packet at Relay1 from the CLIENT	RELAY1	R	01:35:20.313816	DATA	1	CLIENT	RELAY1
Relaying of the packet by relay2	RELAY1	S	01:35:20.315474	DATA	0	RELAY2	SERVER
Ack from server	SERVER	S	01:35:20.316236	ACK	0	SERVER	RELAY1

A diagram to illustrate the working of file transfer from client to server as discussed in this problem