NETWORK REPORT

1.DIFFERENCE BETWEEN MY IMPLEMENTATION AND ACTUAL TCP

1.1. DATA SEQUENCING: Since UDP has no inbuilt sequencing mechanism so it has been explicitly handled here. In my implementation data sequencing is done by dividing data in many packets and assigning serial sequence numbers to each packet by the sender. In my method the nth packet will have the sequence number as n-1. While in actual TCP protocol, first number for sequencing is chosen on random basis , meaning packets are not assigned starting from 0. Instead they assign random numbers as first sequence number for different users. They keep track of these sequence numbers by mapping them. The sequence number in our implementation has no information about the number of bytes being sent. While in actual TCP they keep track of number of bytes too while sequencing.

In our implementation we have predetermined size of bytes to be sent and received by the server and the client in each packet. This value does not change during run-time and thus flow control and congestion control is not performed.
While in TCP flow control and congestion control is handled properly.

1.2. RETRANSMISSION: The key difference in retransmission in actual TCP and in my implementation is the timeout interval in TCP.

In actual TCP timeout is calculated using RTO formula , hence the timeout interval changes dynamically there.
While in our case timeout interval is same (0.1) always and it doesn't change dynamically.
Also we retransmit a data packet only when all data packets are sent atleast once. We don't stop to receive acknowledgement for a particular packet , instead we move on to send next packets . Once we are done sending all packets then we receive acknowledgements for the sent packets , and if acknowledgement for some packet is not received yet then we retransmit it in next loop.

2. Extending my implementation to account for flow control

TCP uses a `sliding window` flow control protocol. In each TCP segment, the receiver specifies in the receive window field the amount of additionally received data (in bytes) that it is willing to buffer for the connection. The sending host can send only up to that amount of data before it must wait for an acknowledgement and receive window update from the receiving host.

To do similar implementation , we would not fix the number of bytes being sent during whole communication. The number of bytes to be sent would depend on the response of receiver. We would add one more parameter to the packets being sent known as byte_size, this parameter will hold the value of bytes being sent in that particular packet. Also while sending acknowledgement , we will send a struct containing sequence

number received and window size. Window size will tell sender that how many bytes receiver wants to receive. Then the sender will adjust it’s byte size on the basis of demand by receiver. Thus, a simple form of flow control has been implemented using an `advertised window` specified by the receiver in the ACK packet it sends back, specifying the amount of additionally received data (in bytes) that it is willing to buffer for the connection.