Lab 4a: Build Your Own Transport

In this assignment we need to build a transport layer on top of IP layer. In which I am implementing reliable, stop-and-wait and sliding window. For sliding window I am using Selective Repeat. We read the input from conn_input() and write the output to conn_output() which are defined in the given code. So basically we get the data from conn_input() to send on to network and write the data to conn_output() at the receiver end and send back the acknowledgment to the sender.

I send a segment over conn_send() which internally sends the segment to receiver. So in segment I start with segno 1 and increment by data size for every next segno. I advertise the sender window size to the receiver so that receiver know the capacity of the sender and even receiver advertise its window size to sender. While advertising it advertise the current window available by subtracting the current buffer it's already holding. May be waiting for ACK or to output to stdout.

I have kept a buffer on sender side so as soon as I send a segment to receiver I push it to buffer, and once I receive the acknowledgment for that packet I remove the segment from the sender buffer. This sender buffer is a special type of structure which is a linked list of packet segment, time at which the packet was sent and retransmission count. So when i send a data to conn_data() I save the current time in the buffer with segment and plus I make retransmission count as 0 and save this too. Retransmission_count can be used when we don't get the ACK for the segment in retransmission timeout.

When I receive a packet I have basically differentiated whether it's a ACK message or the data message by the data field of the segment and handled both different. For ACK I take the ackno and compare with the sender's buffer and remove all the packets which are before that ackno. Which is a aggregated ACKs implementation.

At the receiver end for data part of segment I kept a linked list of segments which are received and then I check the received packet and if its in the correct order then I output the data to conn_output(). Here I handle the flow control by checking if conn_output is empty enough to output the data else we drop the packet. I handle the cases when you get a duplicate segments, then I check the incoming segment with receiver buffer and discard if I have the same segment. I check for corrupt packet by checking the checksum of the received packet by calculating new checksum and comparing with the one we got and drop if the checksums are incorrect.

I am handling the retransmission in case of packet drop, ACK drop or conn_output is too busy to output the buffer. This is handled on sender side, In sender side I have a buffer which also keeps the sent time of the packet. So ctcp_timer() is called for every some time, whenever it is called I check for all the connections and for all the sender's buffer in each connection whose current_time() minus sent time of the segment is greater than the retransmission timeout and retransmission count is less than 5. If so then I send the segment again to receiver and wait for retransmission timeout for next retransmission. After 5 times, if we dont get the ACK we teardown the connection by calling ctcp_destroy() which frees data structure used and removes the connection.

Finally, handled the closing of connection when we get EOF while reading a input. When I get EOF I send a FIN segment to the receiver if all the previous packets have received the ACK, if not I will wait till all the packets in sender buffer gets ACK and then send FIN to receiver. And then receiver ACK's to it and wait for all the packets in the receiver buffer segments data to output to stdout. Once buffer is empty we send the FIN back to sender. Then sender ACK to FIN and closes the connection and frees the memory. Once receiver gets ACK for FIN it also closes connection and does clean up.

Challenge faced:

I have faced many challenges but closing a connection was a tricky one. I was not sure how to handle the FIN and ACK-FIN in one message and I was not clear about the closing connection on TCP. I looked for online content to understand the TCP connection close and implemented in the same way.

Testing:

I have tested using single window with small data and then with large data bigger than window size. And then tested with multiple window with small, large and very large data. Even checked with the google web server for HTTP get, which is not very reliable as it does not respond every time. Testing with binary files and ran the test cases provided tester-student.py file.

I don't remember or know any bugs so far. I have implemented it as per what I have understood about the sliding window and stop and wait for TCP.