

1.

First, I want to say that, my program works perfectly on my own computer, but has probability to fail in vlab. It's so ,wired. Actually during this assignment I met so many wired problems, such as with the same arguments, sometimes it can work perfectly and in other times the program get stuck. If it doesn't work during the test, run more times if it's possible.

I have implemented seq, ack, timeout resend, fast resend, receiver buffer, pipeline, accumulative confirmation mechanism, three way handshake and four way termination, all of the features mentioned in spec, but my programs can't hold the situation that the same segment drop three times in row.

In PTPSocket.py, it has some basic establishment that for sender and receiver, class 'Header', class 'Segment' and class 'PTPSocket'. PTPSocket has some common features and method of sender and receiver, the sender and receiver class are both extend from PTPSocket.

When send a segment, I transfer all the information of a segment to bytes and send it by function 'struct.pack'.

When receive a segment, I transfer bytes stream to a segment also by 'struct.pack'.

The class segment and header has methods to do them.

Sender

Sender has a send buffer which is implemented by deque, and a unack buffer.

First, do the three-way-handshake, then sender will read the source file, get all the data and encapsulated into the segments, finally put a fin segment in the send buffer, as a signal for receiver to quit threading. Then sender start two threading. The send threading keep getting out segments from send buffer and send it. If the segment is resend(sequence number equal to head of unack buffer), just send it; if the segment is new, wait until the unack buffer is not full, send it and put it in unack buffer. If all the data segment is acked, send the last segment in the send buffer, the fin segment, to tell the receiver there is no more data. The receiving thread keep receiving ack segment from receiver. After get a ack segment, if unack buffer is empty, means no unack segment, do nothing. If there is unack segments, remove them from the unack buffer if their sequence number is smaller than the ack number of ack segment received. Finally, if receive three identical ack three times in row, put the segment which at head of unack buffer(the oldest unack segment)to the head of send buffer(send buffer is a deque), in order to resend it as soon as possible. If all segment is sent and acked, send fin segment as the first step of four way termination, kill both send and receive threading and do the least steps of termination. The logic of timeout resend is: (1) every time send a segment, if the timer is not working, start it; if it's working, leave it there, (2) when **new** ack received, if there is unack segments refresh the timer; if not, delete the timer. (3) when all the segment acked, delete the timer.

Receiver

Receiver has only one thread. After three times handshake, it start the receiving

threading. The threading keeps receiving segments from sender. When a segment received, if the segment is fin, kill the thread, if not go next part. Then see if the segment is duplicated, if it is, do nothing ;if it's not ,determine it's order or disorder. If the received segment's sequence number is not equal to the seq that the receiver want now, just put the segment into disorderbuffer. If the segment is which the receiver want now, put it in orderbuffer, then determine if there is disorder received segment. If there is, try to remove all the segments which is ordered now from disorder buffer to orderbuffer. For example, the disorder buffer has now [2,3,4,6,7,9] six segments with their seq, and the receiver get 1, first put 1 in order buffer and get 234 from disorder buffer to order buffer, so now there are [1,2,3,4] in order buffer and [6,7,9] in disorder buffer, waiting for 5. Finally send new ack segment to tell the sender what is the sequence number of next segment it wants. After, it received fin segment, the threading will be terminated, and then do the four way termination. After all, go through the order buffer, get all the data in each segment and write them into target file.

Header

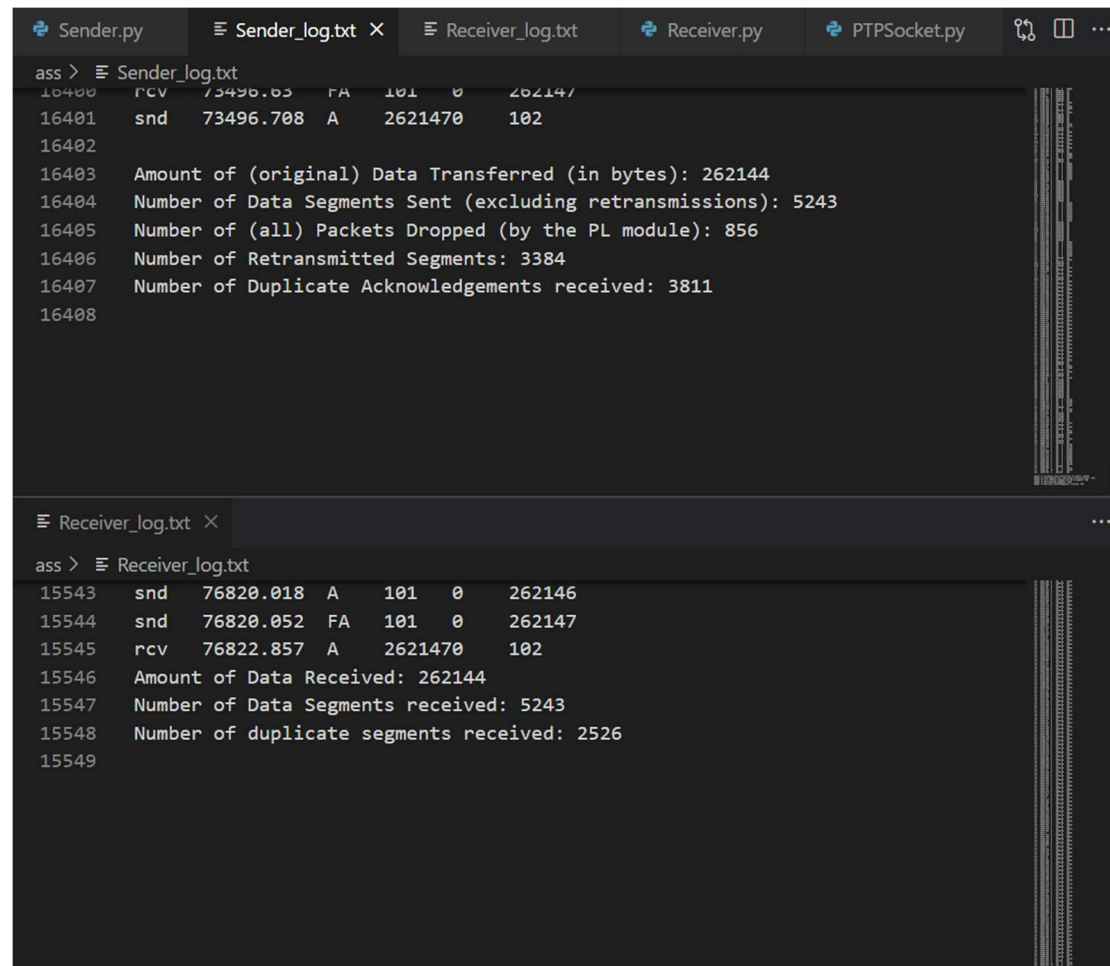
My PTP header has five elements, two are numbers, sequence number and ack number, three are flags, ACK, SYN, FIN., they has the same meaning with real TCP.

Improvements

I think the logic of my code is not clear enough, it could be improved.

Here is two result I get by this set of arguments

```
# 127.0.0.1 10086 256KB.txt 500 50 100 0.1 300  
# 10086 target.txt
```



The screenshot shows a terminal window with several tabs: Sender.py, Sender_log.txt, Receiver_log.txt, Receiver.py, and PTPSocket.py. The active tab is Sender_log.txt, which displays the following log entries:

```
ass > Sender_log.txt  
16400 rcv 73496.63 FA 101 0 262147  
16401 snd 73496.708 A 2621470 102  
16402  
16403 Amount of (original) Data Transferred (in bytes): 262144  
16404 Number of Data Segments Sent (excluding retransmissions): 5243  
16405 Number of (all) Packets Dropped (by the PL module): 856  
16406 Number of Retransmitted Segments: 3384  
16407 Number of Duplicate Acknowledgements received: 3811  
16408
```

The next tab is Receiver_log.txt, which displays the following log entries:

```
ass > Receiver_log.txt  
15543 snd 76820.018 A 101 0 262146  
15544 snd 76820.052 FA 101 0 262147  
15545 rcv 76822.857 A 2621470 102  
15546 Amount of Data Received: 262144  
15547 Number of Data Segments received: 5243  
15548 Number of duplicate segments received: 2526  
15549
```

```
Sender.py  Sender_log.txt X  Receiver_log.txt  Receiver.py  PTPSocket.py  ? ? ...
ass > Sender_log.txt
16854 rcv 60720.885 A 101 0 262145
16855 snd 60720.704 F 2621450 101
16856 rcv 60721.085 A 101 0 262146
16857 rcv 60721.104 FA 101 0 262147
16858 snd 60721.307 A 2621470 102
16859
16860 Amount of (original) Data Transferred (in bytes): 262144
16861 Number of Data Segments Sent (excluding retransmissions): 5243
16862 Number of (all) Packets Dropped (by the PL module): 873
16863 Number of Retransmitted Segments: 3623
16864 Number of Duplicate Acknowledgements received: 4148
16865

Receiver_log.txt X
ass > Receiver_log.txt
15982 rcv 67341.258 F 2621450 101
15983 snd 67341.596 A 101 0 262146
15984 snd 67341.62 FA 101 0 262147
15985 rcv 67341.683 A 2621470 102
15986 Amount of Data Received: 262144
15987 Number of Data Segments received: 5243
15988 Number of duplicate segments received: 2746
15989
```