3331 Assessment Report

z5261811

August 5, 2021

1 Implementation details

Python3 is used for this project

Modules

There are 5 modules in my implementation: Sender, Receiver, PacketLoss, PacketEncoder and PacketDecoder.

- Sender: responsible for read input data file, establish and terminate connection with Receiver, send data to receiver and write sender log.
- Receiver: responsible for receive data from sender, send ACK to sender, write the received data to a certain file and write receiver log
- PacketLoss: Simulate packet loss, ”randomly” drop packets according to the input seed.
- PacketEncoder: given a data segment and the packet information like sequence number and ACK number, pack data and header in to a byte string. Used in both sender and receiver
- PacketDecoder: unpack the received byte string to a dictionary, which contains the packet’s information and data segment. Used in both sender and receiver.

Achieved features

1. Initial sequence number (ISN) for both sender and receiver are set to 0
2. A three-way handshake (SYN, SYN+ACK, ACK) for the connection establishment. This process does not contains any data exchange, however ISN on both sides are increased by 1.
3. The four-segment connection termination (FIN, ACK, FIN, ACK). The Sender will initiate the connection close once the entire file has been successfully transmitted, and the Receiver to combine the ACK and FIN in one message.

4. Sender keep track of its sequence number, and simulate TCP sliding window when send data. MWS and MSS are given as inputs.

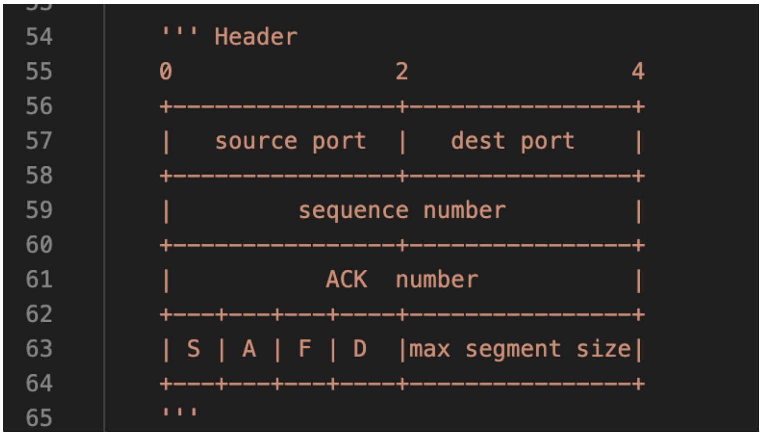
5. Sender maintain a single timer on the oldest not acked chunk. Timeout is a constant value given as an input. Sender uses cumulative ACK, where once an ACK is received, it assumes all the packets prior to the ACK is received by Receiver successfully, and slide the window to the latest acked position.

6. When 3 repeated ACKs are received, Sender simulate fast re-transmit. It will restart timer immediately and send all data in current window.

7. Receiver stores received data into a buffer, both in-order and out-of-order chunks. It makes an appropriate acknowledgement to Sender immediately after receiving a chunk, and write received data to a file in the correct order after connection terminates.

8. Packet loss module to decide whether to send a packet or simulate a packet loss.

2 Header design



```
''' Header
0                2                4
+----------------+----------------+
|   source port  |   dest port    |
+----------------+----------------+
|         sequence number         |
+----------------+----------------+
|           ACK  number           |
+---+---+---+----+----------------+
| S | A | F | D  |max segment size|
+---+---+---+----+----------------+
'''
```

Figure 1: Header design

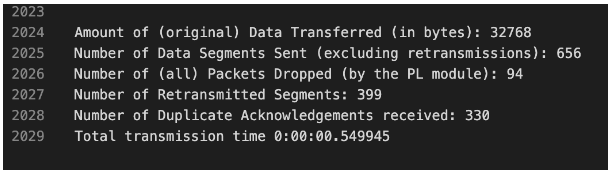
- byte 0 2: port number that this packet is sent from
- byte 3 4: port number that this packet is sent to
- byte 5 8: sequence number of this packet
- byte 9 12: ACK number of this packet

- byte 13 14: packet type formed by 4 4-bits integer. S has value 0 or 1, A has value 0 or 2, F has value 0 or 4, D has value 0 or 8. By computing the sum of these 4 integers, PacketDecoder can decide the packet type.
- byte 15 16: max segment size to let receiver know max segment size of the data chunk from sender

3 Question (a)

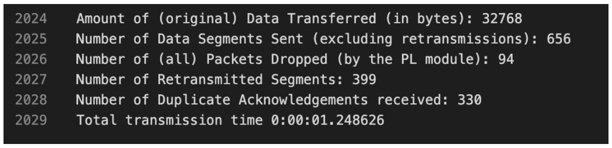
If timeout is large, the Sender will be wasting time on waiting for ACK for lost packets, so we should try to make timeout as small as possible. However, if timeout is too small, the Sender will re-transmit the packets before ACKs are received, which will cause a lot of unnecessary re-transmission.

For pdrop=0.1, seed=300, MWS=500, MSS=50, after a few experiments, I choose 5ms as the timeout value, which would complete transmission in less than 1 seconds, and a reasonable number of re-transmitted segments. For timeout value less than 1, it takes more than 1 minute to transfer the file due to the unnecessary re-transmission. For larger timeout value, it takes longer to complete transmission as timeout increases.



```
Amount of (original) Data Transferred (in bytes): 32768
Number of Data Segments Sent (excluding retransmissions): 656
Number of (all) Packets Dropped (by the PL module): 94
Number of Retransmitted Segments: 399
Number of Duplicate Acknowledgements received: 330
Total transmission time 0:00:00.549945
```

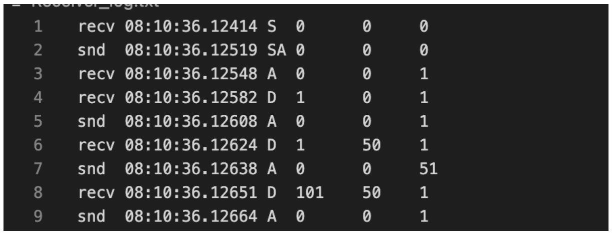
Figure 2: pdrop=0.1, pdrop=300, MWS=500, MSS=50, timeout=5, Sender



```
Amount of (original) Data Transferred (in bytes): 32768
Number of Data Segments Sent (excluding retransmissions): 656
Number of (all) Packets Dropped (by the PL module): 94
Number of Retransmitted Segments: 399
Number of Duplicate Acknowledgements received: 330
Total transmission time 0:00:01.248626
```

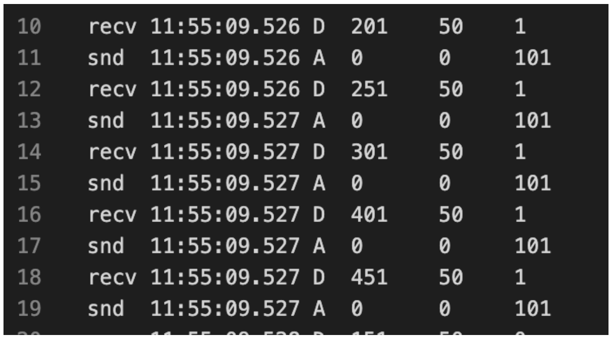
Figure 3: pdrop=0.1, pdrop=300, MWS=500, MSS=50, timeout=20, Sender

The packet drop occurs when the order-of-order packet is received. For example, with pdrop=0.1, in Figure 4, packet 101 is received before 51, which means 51 is dropped. With pdrop-0.3, in Figure 5, packet 401 is received before 351, which means packet 351 is dropped.



```
recv 08:10:36.12414 S  0      0     0
snd  08:10:36.12519 SA 0      0     0
recv 08:10:36.12548 A  0      0     1
recv 08:10:36.12582 D  1      0     1
snd  08:10:36.12608 A  0      0     1
recv 08:10:36.12624 D  1      50    1
snd  08:10:36.12638 A  0      0     51
recv 08:10:36.12651 D  101    50    1
snd  08:10:36.12664 A  0      0     1
```

Figure 4: pdrop=0.1, pdrop=300, MWS=500, MSS=50, timeout=5, Receiver



```
recv 11:55:09.526 D  201    50    1
snd  11:55:09.526 A  0      0     101
recv 11:55:09.526 D  251    50    1
snd  11:55:09.527 A  0      0     101
recv 11:55:09.527 D  301    50    1
snd  11:55:09.527 A  0      0     101
recv 11:55:09.527 D  401    50    1
snd  11:55:09.527 A  0      0     101
recv 11:55:09.527 D  451    50    1
snd  11:55:09.527 A  0      0     101
```

Figure 5: pdrop=0.3, pdrop=300, MWS=500, MSS=50, timeout=5, Receiver

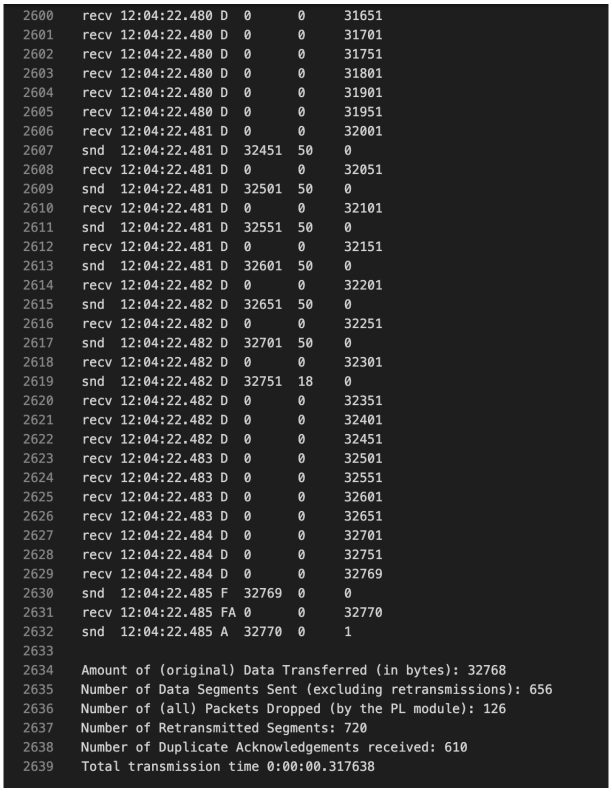
4 Question (b)

Tcurrent = 5ms
pdrop = 0.1
MWS = 500 bytes
MSS = 50 bytes
seed = 300

	number of total transmitted packets	overall transfer time
Tcurrent	10693	0:00:02.545976
4 × Tcurrent	10693	0:00:02.468946
Tcurrent/4	11099	0:00:02.525610

With timeout set to 5ms and 20ms, the number of total transmitted packets are the same, 10693. With timeout set to 1.25ms, the total number of transmitted packets are slightly larger than the other 2 timeout value. This is because with a lower timeout value, some packets are resent due to timeout before ACKs are received.
The overall transfer time are generally the same for 3 Tcurrent value, this is because the probability of packet drop is set to a low value. With low pdrop, as long as Tcurrent is greater than RTT, the overall transfer time will be quite close to each other.

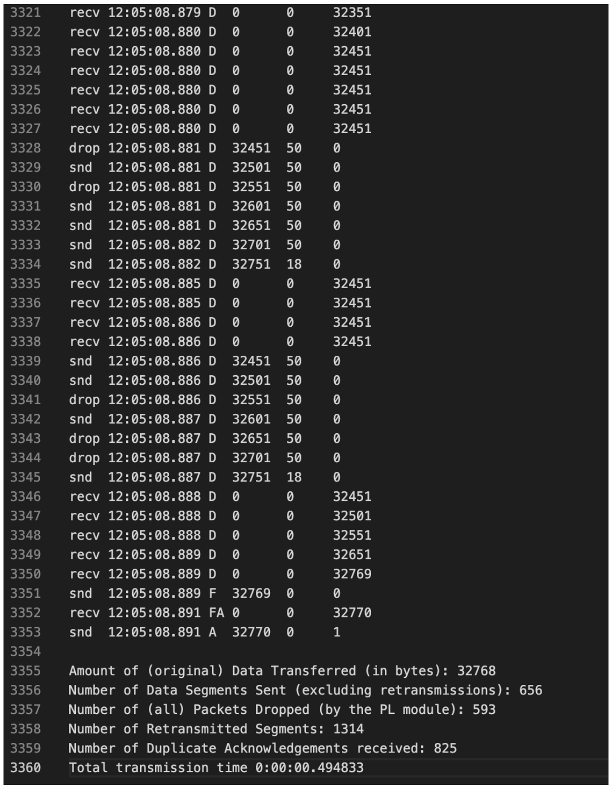
5 Appendix



```
recv 12:04:22.480 D  0      0     31651
recv 12:04:22.480 D  0      0     31701
recv 12:04:22.480 D  0      0     31751
recv 12:04:22.480 D  0      0     31801
recv 12:04:22.480 D  0      0     31901
recv 12:04:22.480 D  0      0     31951
recv 12:04:22.481 D  0      0     32001
snd  12:04:22.481 D  32451  50    0
recv 12:04:22.481 D  0      0     32051
snd  12:04:22.481 D  32501  50    0
recv 12:04:22.481 D  0      0     32101
snd  12:04:22.481 D  32551  50    0
recv 12:04:22.481 D  0      0     32151
snd  12:04:22.481 D  32601  50    0
recv 12:04:22.482 D  0      0     32201
snd  12:04:22.482 D  32651  50    0
recv 12:04:22.482 D  0      0     32251
snd  12:04:22.482 D  32701  50    0
recv 12:04:22.482 D  0      0     32301
snd  12:04:22.482 D  32751  18    0
recv 12:04:22.482 D  0      0     32351
recv 12:04:22.482 D  0      0     32401
recv 12:04:22.482 D  0      0     32451
recv 12:04:22.483 D  0      0     32501
recv 12:04:22.483 D  0      0     32551
recv 12:04:22.483 D  0      0     32601
recv 12:04:22.483 D  0      0     32651
recv 12:04:22.484 D  0      0     32701
recv 12:04:22.484 D  0      0     32751
recv 12:04:22.484 D  0      0     32769
snd  12:04:22.485 F  32769  0     0
recv 12:04:22.485 FA 0      0     32770
snd  12:04:22.485 A  32770  0     1

Amount of (original) Data Transferred (in bytes): 32768
Number of Data Segments Sent (excluding retransmissions): 656
Number of (all) Packets Dropped (by the PL module): 126
Number of Retransmitted Segments: 720
Number of Duplicate Acknowledgements received: 610
Total transmission time 0:00:00.317638
```

Figure 6: Some packet sequence for pdrop=0.1, pdrop=300, MWS=500, MSS=50, timeout=5, Receiver



```
recv 12:05:08.879 D  0      0     32351
recv 12:05:08.880 D  0      0     32401
recv 12:05:08.880 D  0      0     32451
recv 12:05:08.880 D  0      0     32451
recv 12:05:08.880 D  0      0     32451
recv 12:05:08.880 D  0      0     32451
recv 12:05:08.880 D  0      0     32451
drop 12:05:08.881 D  32451  50    0
snd  12:05:08.881 D  32501  50    0
drop 12:05:08.881 D  32551  50    0
snd  12:05:08.881 D  32601  50    0
snd  12:05:08.881 D  32651  50    0
snd  12:05:08.882 D  32701  50    0
snd  12:05:08.882 D  32751  18    0
recv 12:05:08.885 D  0      0     32451
recv 12:05:08.885 D  0      0     32451
recv 12:05:08.886 D  0      0     32451
recv 12:05:08.886 D  0      0     32451
snd  12:05:08.886 D  32451  50    0
snd  12:05:08.886 D  32501  50    0
drop 12:05:08.886 D  32551  50    0
snd  12:05:08.887 D  32601  50    0
drop 12:05:08.887 D  32651  50    0
drop 12:05:08.887 D  32701  50    0
snd  12:05:08.887 D  32751  18    0
recv 12:05:08.888 D  0      0     32451
recv 12:05:08.888 D  0      0     32501
recv 12:05:08.888 D  0      0     32551
recv 12:05:08.889 D  0      0     32651
recv 12:05:08.889 D  0      0     32769
snd  12:05:08.889 F  32769  0     0
recv 12:05:08.891 FA 0      0     32770
snd  12:05:08.891 A  32770  0     1

Amount of (original) Data Transferred (in bytes): 32768
Number of Data Segments Sent (excluding retransmissions): 656
Number of (all) Packets Dropped (by the PL module): 593
Number of Retransmitted Segments: 1314
Number of Duplicate Acknowledgements received: 825
Total transmission time 0:00:00.494833
```

Figure 7: Some packet sequence for pdrop=0.3, pdrop=300, MWS=500, MSS=50, timeout=5, Receiver