**Question 1**

I first wrote a **skeleton class for both Sender and Receiver** and also an **STPPacket class** which would act as the base segment structure for all my segments.

* I added in the necessary variables for both Sender and Receiver as defined by the assignment
* I also added in extra variables for tracking state, seq numbers, ack numbers and more.

I then wrote methods for **creating packets, receiving segments, sending packets over UDP** and **closing the connection** for both the sender / receiver

* I tested by “receiving” a file from the app-layer, creating a segment with all the file bytes, sending the whole payload over UDP to the receiver.
* I made sure UDP was functioning properly by checking the difference between test1.txt and the final file receiver.txt. I repeated the same experiment with test2.txt with receiver.txt

Then I added in a method **split\_data() to test** **Max Segment Size feature**, extracting only the max size number of bytes from the app-layer file then appending it to the segment payload before sending it over UDP.

* I tested this by choosing various values of MSS and running the two programs to make sure each packet was being split properly by the MSS value, payload added to the segment then sent over UDP.
* I double checked that all the split packets arrived correct at the receiver side, by again checking diff between test.txt with receiver.txt

Then I added in methods for **creating SYN SYNACK ACK FIN segments** and logic to **increment Sequence / Acknowledgement numbers** for both the Sender and Receiver. On top of this, I created a method for **updating sender\_log.txt / receiver\_log.txt files** so that I can keep track of what is going on in the program.

* I tested this many times, so that the seq / ack numbers were correct on both sides.
* I made sure that the **3-way-handshake worked correctly**
* I made sure that the **FIN – ACK FIN – ACK** **worked correctly**

I created a **Packet Loss Drop** **Class** for the PLD feature and also a **method to** **take in packets as an argument and** **generate a pseudorandom number to drop or transmit the packet**

* I tested this by choosing various **Packet Loss Drop values** and checked in my log output that packets were being dropped.

I created a **Timer feature**, by calling the **time.clock()** function to keep track of current time and assist with **timeout and retransmissions**

* I tested this against dropped packages, where all dropped packages would have a given constant timeout value before retransmitting.

Everything was held together by **main methods on both the Sender / Receiver side**, where there are **several events / states** which keep track of what and when to send / receive packets and when to close the connection.

* Sender states: CLOSED, SYN\_SENT, TIMEOUT, ESTABLISHED, END

Receiver states: LISTEN, SYN\_RECV, SYNACK\_SENT, ETABLISHED, END

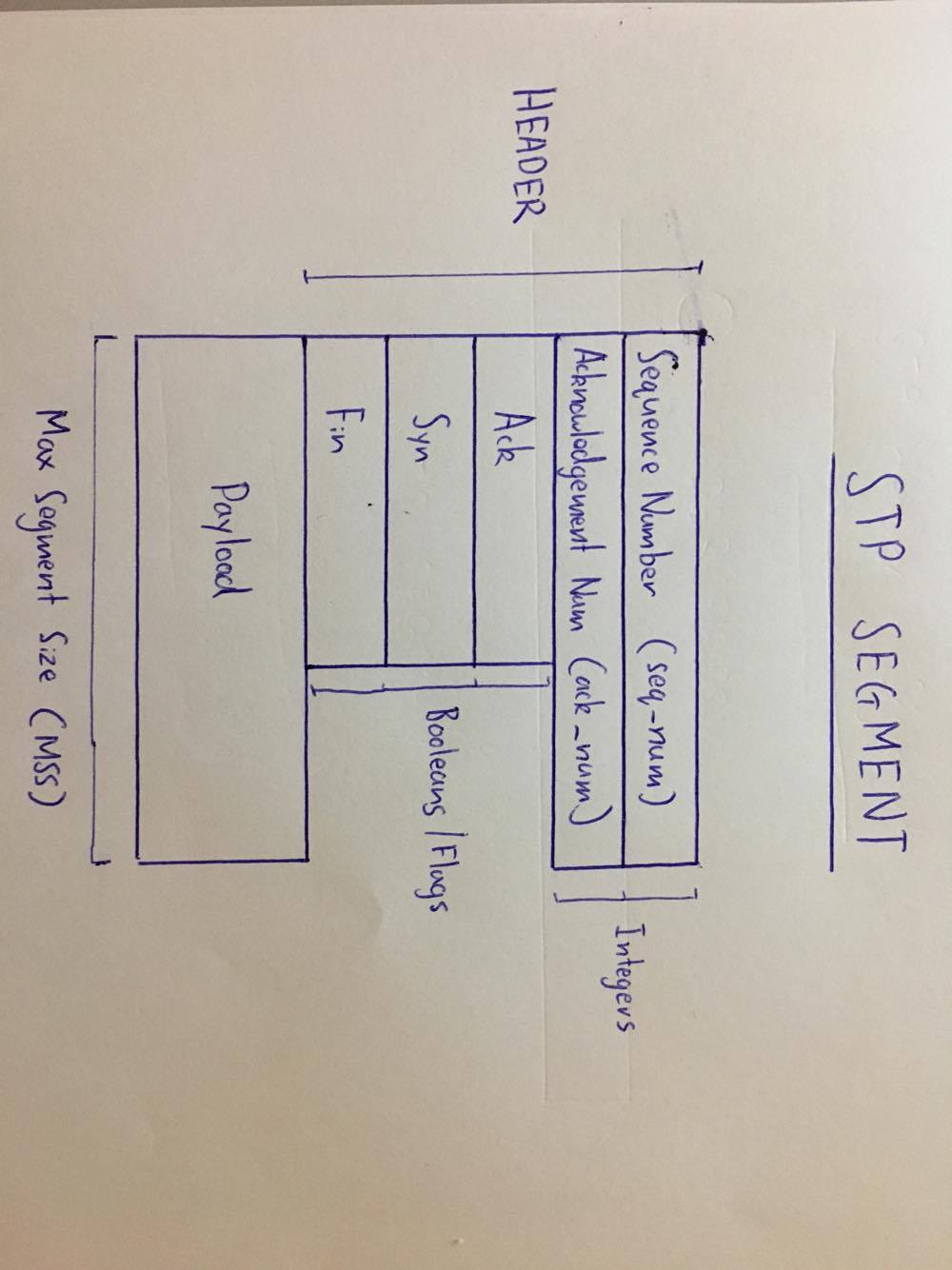
* On the both Sender / Receiver sides, they will go through the **3-way-handshake** before moving to an **ESTABLISHED CONNECTION** state.
* During the established connection state, the Sender will grab data from the app-layer file, create new packets based on MSS, parse it through the PLD system, transmit to the Receiver for processing.
* Likewise, the Receiver will be **listening for any Data segments** and as soon as it receives them, it will **generate a corresponding ACK**

SENDER: App-layer file 🡪 cut out MSS bytes 🡪 create segment 🡪 parse segment through PLD 🡪 send over UDP 🡪 wait for ACK

RECEIVER: Listen for packet 🡪 receive packet 🡪 determine if correct seq\_num 🡪 send ACK 🡪 extract payload 🡪 append to file

|  |  |
| --- | --- |
| **Implemented Features** | **Not Implemented** |
| **Both:** Three-way-handshake (SYN SYNACK ACK) | **Sender:** Fast retransmit |
| **Both:** Four-segment cnnt termination (FIN ACK FIN ACK) | **Sender:** Maximum Window Size (MWS) |
| **Sender:** Single-timer for timeout operation | **Sender:** Packet Delay |
| **Sender:** Simple timeout retransmit |  |
| **Receiver:** Immediate acknowledgement / ACKs |  |
| **Both:** Sequence Numbers, Acknowledgement Numbers |  |
| **Sender:** Maximum Segment Size (MSS) |  |
| **Sender:** Packet Loss |  |
| **Sender:** Constant timeout |  |

**Question 2**



**Question 3**

(a)

A suitable timeout value is 2 \* the average Round-Trip Time, which is calculated based off running average RTT.

**Test1.txt with timeout = 100ms**

### FINAL RECEIVER LOG ###

rcv 52.608 S 0 0 0

snd 53.391 SA 0 0 1

rcv 53.929 A 1 0 1

snd 54.624 A 1 0 51

rcv 54.908 D 1 50 1

snd 55.498 A 51 0 101

rcv 55.732 D 51 50 1

snd 56.287 A 101 0 151

rcv 56.542 D 101 50 1

snd 57.108 A 151 0 201

rcv 57.354 D 151 50 1

snd 57.916 A 201 0 251

rcv 58.155 D 201 50 1

snd 58.742 A 251 0 301

rcv 58.981 D 251 50 1

snd 59.706 A 301 0 351

rcv 60.271 D 301 50 1

snd 61.075 A 351 0 401

rcv 61.352 D 351 50 1

snd 61.946 A 401 0 451

rcv 62.204 D 401 50 1

snd 62.787 A 451 0 501

rcv 63.042 D 451 50 1

snd 63.607 A 501 0 551

rcv 63.842 D 501 50 1

snd 64.405 A 551 0 601

rcv 64.648 D 551 50 1

snd 65.207 A 601 0 651

rcv 65.438 D 601 50 1

snd 65.932 A 651 0 701

rcv 66.162 D 651 50 1

snd 66.7 A 701 0 751

rcv 66.954 D 701 50 1

snd 67.5 A 751 0 801

rcv 67.73 D 751 50 1

snd 68.282 A 801 0 851

rcv 68.508 D 801 50 1

snd 69.057 A 851 0 901

rcv 69.299 D 851 50 1

snd 69.855 A 901 0 951

rcv 70.096 D 901 50 1

snd 70.699 A 951 0 1001

rcv 70.929 D 951 50 1

snd 71.494 A 1001 0 1051

rcv 71.737 D 1001 50 1

snd 72.281 A 1051 0 1101

rcv 72.516 D 1051 50 1

snd 72.993 A 1101 0 1151

rcv 73.193 D 1101 50 1

snd 73.697 A 1151 0 1201

rcv 73.912 D 1151 50 1

snd 74.601 A 1201 0 1251

rcv 74.943 D 1201 50 1

snd 75.577 A 1251 0 1301

rcv 75.812 D 1251 50 1

snd 76.418 A 1301 0 1351

rcv 76.66 D 1301 50 1

snd 77.203 A 1351 0 1401

rcv 77.443 D 1351 50 1

snd 78.012 A 1401 0 1451

rcv 78.24 D 1401 50 1

snd 78.794 A 1451 0 1501

rcv 79.017 D 1451 50 1

snd 79.54 A 1501 0 1551

rcv 79.779 D 1501 50 1

snd 80.397 A 1551 0 1594

rcv 80.693 D 1551 43 1

rcv 81.228 F 1594 0 1

snd 82.173 FA 1594 0 1595

rcv 82.876 A 1594 0 2

**Test1.txt with timeout = 100ms with pdrop = 0.3**

### FINAL RECEIVER LOG ###

rcv 80.197 S 0 0 0

snd 80.591 SA 0 0 1

rcv 80.989 A 1 0 1

snd 81.572 A 1 0 51

rcv 81.792 D 1 50 1

snd 82.181 A 51 0 101

rcv 82.405 D 51 50 1

snd 82.936 A 101 0 151

rcv 83.158 D 101 50 1

snd 83.689 A 151 0 201

rcv 83.893 D 151 50 1

snd 84.397 A 201 0 251

rcv 84.606 D 201 50 1

snd 85.134 A 251 0 301

rcv 85.384 D 251 50 1

snd 85.884 A 301 0 351

rcv 86.1 D 301 50 1

snd 86.605 A 351 0 401

rcv 86.841 D 351 50 1

snd 87.372 A 401 0 451

rcv 87.611 D 401 50 1

snd 88.103 A 451 0 501

rcv 88.316 D 451 50 1

snd 88.79 A 501 0 551

rcv 89.018 D 501 50 1

snd 89.521 A 551 0 601

rcv 89.754 D 551 50 1

snd 90.299 A 601 0 651

rcv 90.52 D 601 50 1

snd 91.122 A 651 0 701

rcv 91.45 D 651 50 1

snd 92.048 A 701 0 751

rcv 92.546 D 701 50 1

snd 93.274 A 751 0 801

rcv 93.513 D 751 50 1

snd 94.033 A 801 0 851

rcv 94.277 D 801 50 1

snd 94.812 A 851 0 901

rcv 95.051 D 851 50 1

snd 95.604 A 901 0 951

rcv 95.871 D 901 50 1

snd 96.441 A 951 0 1001

rcv 96.7 D 951 50 1

snd 97.217 A 1001 0 1051

rcv 97.44 D 1001 50 1

snd 98.143 A 1051 0 1101

rcv 98.547 D 1051 50 1

snd 99.176 A 1101 0 1151

rcv 99.531 D 1101 50 1

snd 100.241 A 1151 0 1201

rcv 100.508 D 1151 50 1

snd 101.136 A 1201 0 1251

rcv 101.454 D 1201 50 1

snd 102.074 A 1251 0 1301

rcv 102.31 D 1251 50 1

snd 102.847 A 1301 0 1351

rcv 103.073 D 1301 50 1

snd 103.657 A 1351 0 1401

rcv 103.955 D 1351 50 1

snd 104.609 A 1401 0 1451

rcv 104.971 D 1401 50 1

snd 105.729 A 1451 0 1501

rcv 106.001 D 1451 50 1

snd 106.668 A 1501 0 1551

rcv 106.939 D 1501 50 1

snd 107.679 A 1551 0 1594

rcv 108.224 D 1551 43 1

rcv 108.6 F 1594 0 1

snd 109.08 FA 1594 0 1595

rcv 109.553 A 1594 0 2

(b)

|  |  |
| --- | --- |
| TCurrent | **36/40** transmitted, 120.41 – 84.474 = **35ms** |
| 4 \* TCurrent | **35/40** transmitted, 119.783 – 85.928 = **33.85ms** |
| Tcurrent / 4 | **38/40** transmitted, 82.323 – 50.282 = **32.041ms** |