**STP Report**

**Brief:**

My version of STP uses a tuple as the packet containing all the header fields and the data. This is sent from the sender to the receiver as a string, which is then converted back into a tuple on the receiver side. Both the sender and receiver keeps track of the local ack and sequence number (this is the value the local program expects to use), and received ack and received sequence (value used as a request from the other side for a specific packet). The local and received values are used so that if a segment is interrupted either due to a timeout/fast retransmit, the program will know where to start sending again.

The sender and receiver uses socket blocking (turned off) to overcome the stop and wait implementation. The sender will continuously send packets until a response is received from the receiver. A dictionary is used on both sender and receiver to save the segments according to their sequence numbers.

There is a bug (unknown causes) which causes the sender to not receive the acknowledgement packets fast enough causing it’s own unstable packet delay from the receiver end.

**Features:**

Sender:

* Three way handshake to establish connection
* Four segment connection termination
* Send segmented file even in situations created by the PLD
* Handles packet timeouts based on timeout intervals
  + Timeout intervals updated every time a packet is sent/correct ack is received
* File segments sent based on MSS
* MWS was not implemented
* PLD:
  + Dropped packets
  + Duplicate packets
  + Corrupted packets
  + Reordered packets
    - Bug with
  + Delayed packets (simplified)
    - Only allows for 1 packet to be delayed at a time (similar to re-order)

Receiver:

* Three way handshake to establish connection
* Four segment connection termination
* Handles out of order packets
* Handles corrupted packets accordingly
* Requests correct packets from the sender which are needed (ie, for fast retransmit)
* Does not have a dynamic range to receive from socket (static number of bytes to read from socket)

**STP Header Fields:**

|  |  |
| --- | --- |
| Flag | |
| Acknowledgement Number | Sequence Number |
| Data | |
| Checksum | |

* Flag:
  + Holds the type of packet (SYN,SYN-ACK,ACK,FIN) (stored as a string)
  + Indicates to the destination as to what type of packet it is
    - SYN: Used to start the three way handshake
    - SYN-ACK used to acknowledge the SYN received
    - ACK used to acknowledge a packet received
    - FIN used to start the connection termination
* Acknowledgement number (integer)
  + Mostly used by the receiver to tell the sender what packet it requires next
* Sequence number (integer)
  + Indicates which data segment the packet is hold information for
  + From the sender, it represents the current data segment being sent
  + From receiver, represents that the particular data segment has been received successfully
* Data (size of MSS)(Integer)
  + Contains the specific data segment being sent
* Checksum calculated from data (integer)
  + Calculated by both the sender before it sends and the receiver when it receives
  + The two are then compared to see if the packet has been corrupted during the process of being transmitted

The header and the data are stored in a tuple which can can be accessed with packet[0….4]. The values are stored inside the tuple when the packet is created and converted into a string when sent.

**Design Tradeoffs:**

One of the biggest design tradeoffs was using the format of a string(tuple) to use to send the segments. This was used solely for the ease of retrieving values from the packet. The size for the packet *string(tuple(header + data))*  is always larger than the packet *string(8bits of 0/1 + data).*  I was not able to get the fast retransmit working along with the MWS so I decided the fast retransmit was more important as it would guarantee the file to be sent correctly.

**Improvements and Extensions:**

* Use single bits (0/1) for the header fields instead of using sending a tuple
  + Utilizes memory more effective (less bytes need to be sent)
  + Less overhead
* Dynamic receiver socket size
  + This will allow any value of MSS to be used
  + MSS can be included in the header field
* If threading were to be used, the packet delay function will be able to be implemented correctly

- Overall code style

* + PLD should have been its own encapsulated function whereas the separate function only determines which PLD error to use
  + Less global variables to be used as it becomes hard to keep track of
  + Although there are functions are used, the functions used should have been more specific and narrow as a general function becomes messy when there are a lot of specific cases

**References to used code**

Line 72/80:





Used in the PLD module (corrupt) to convert a string into bits so that a single bit can be flipped

<https://stackoverflow.com/questions/10237926/convert-string-to-list-of-bits-and-viceversa>

Line 41:

Used to calculate the checksum of the data contained inside the packet

<https://code.activestate.com/recipes/52251-simple-string-checksum/>

**Questions:**

5(a)

|  |  |  |
| --- | --- | --- |
| pDrop | Dropped packets (sequence) | Number of drops |
| 0.1 | 201, 2001,2701, 2801 | 4 |
| 0.3 | 1, 1(retransmit), 301, 401, 301(retransmit), 701, 301 (retransmit), 1201, 1401, 1501, 701 (retransmit), 2101……. | 29 |

(Appendix: 5(a))

There is a significant increases in drops From 4 to 29.due to the bug with the delayed acknowledgement packets, it causes the sender to sometimes retransmit more packets than needed, this also means that more packets will be handled by the PLD module and increasing the chances of dropped packets, which could explain why a large amount of retransmits are dropped as well. The packet sequences for pDrop = 0.1 were almost all in order except for the ones which were dropped compared to pDrop =0.3 where they are more out of order due to the amount of packets dropped, retransmitted, and timed out.

5(b)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Gamma | Packets transmitted | Total Time (ms) | Packets delayed | Timeouts |
| 2 | 17682 | 176200 | 306 | 382 |
| 4 | 18034 | 171406 | 295 | 28 |
| 6 | 18275 | 386803 | 299 | 51 |

(Appendix: 5(b))

The values for gamma = 4 and 6 follow a trend. The total time and packets transmit should increase along with the gamma value (takes longer to timeout = longer time to retransmit). However, gamma = 2 is the anomaly. I believe this is due to the bug I mentioned earlier in the brief where sometimes the acknowledgements from the receiver won’t be received on time. This explains the high value for number of timeout retransmits. Which in turn, increase the total time for transmission to be more than gamma = 4.

5 (c)

Time taken: 1896840 ms (31minutes)

File transfer successful: Yes

Packet dropped: 4215

Packet duplicated: 3726

Packet corrupted: 3292

Packet reordered: 4476

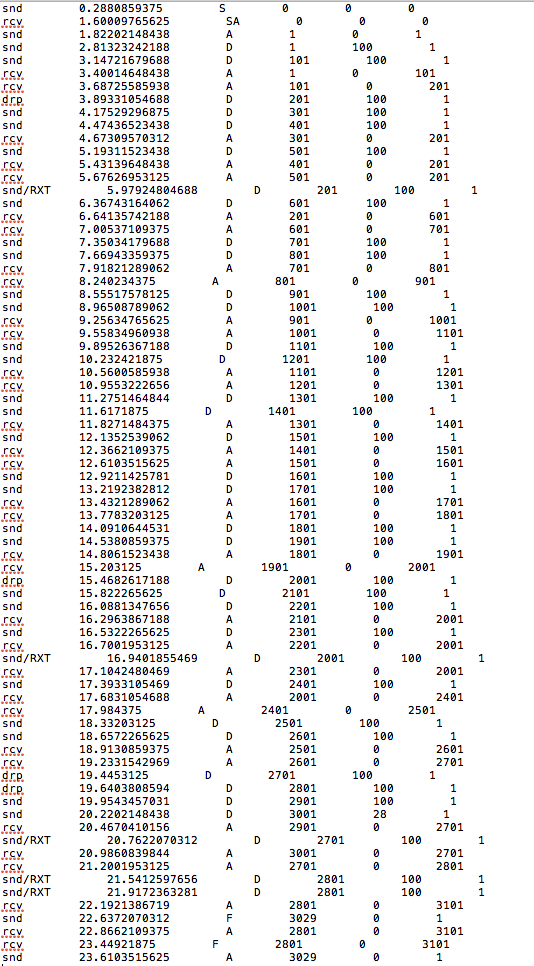
(Appendix 5(c))

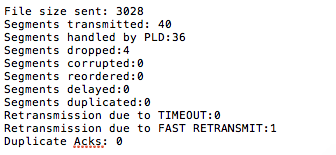
The most significant errors would be packet dropping and corruption. This is because although packet reordering has the highest frequency, sending the out of order packet bypasses the PLD module, so although some of the packets that are reordered might be retransmitted (due to fast retransmit), the frequencies for time outs (adds to the total time the most) will be less than packet corruption and packet dropping.

These two are in the same category as from the sender’s view, it has sent the packet and the ack is never received. So although the errors are inherently different, the methods in which the sender and receiver handles these packets remain the same

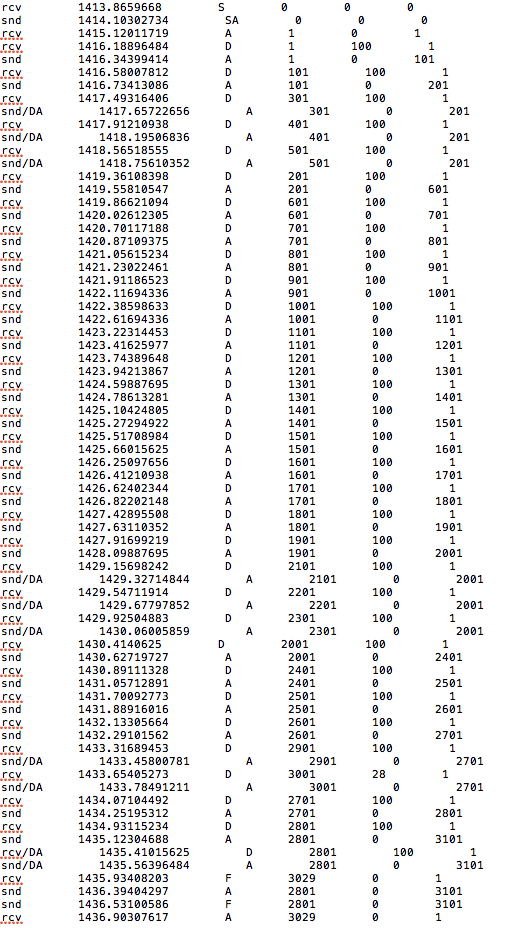
**Appendix**

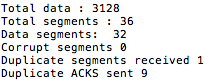
**5(a) Sender: pDrop = 0.1**

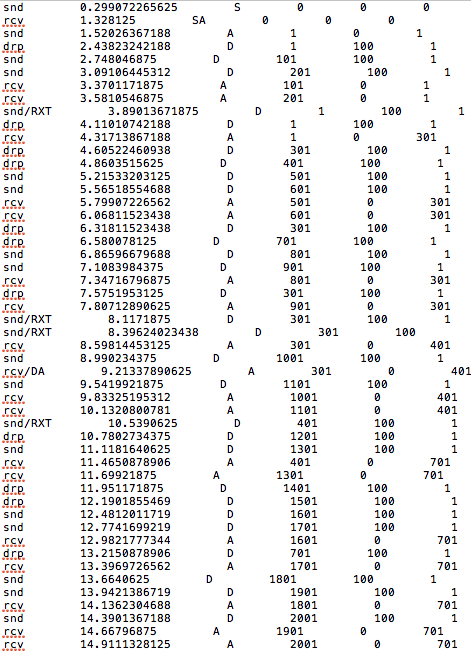
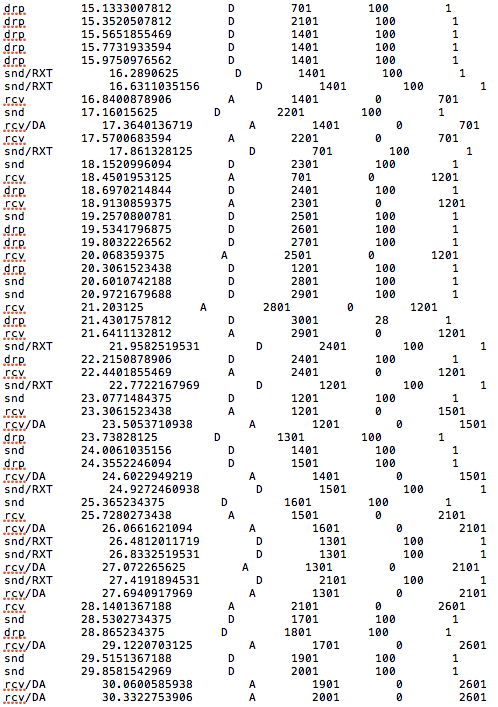


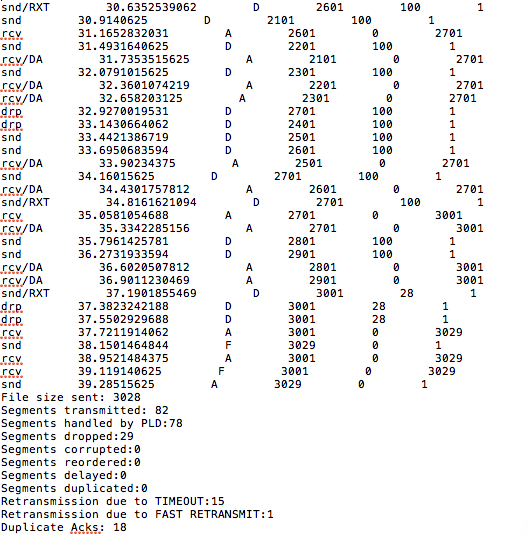


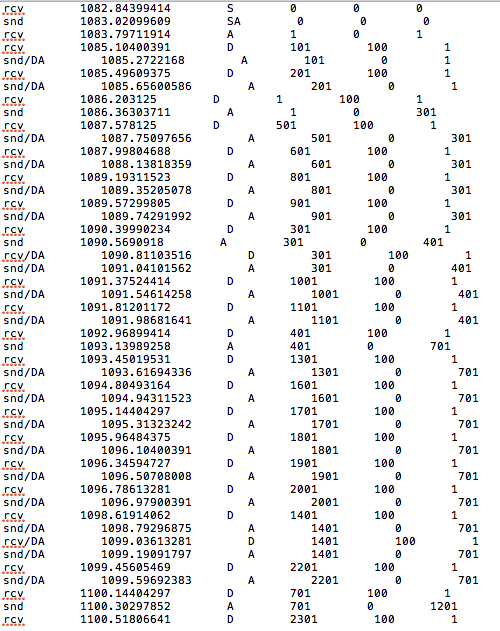
**5(a) Receiver: pDrop = 0.1**

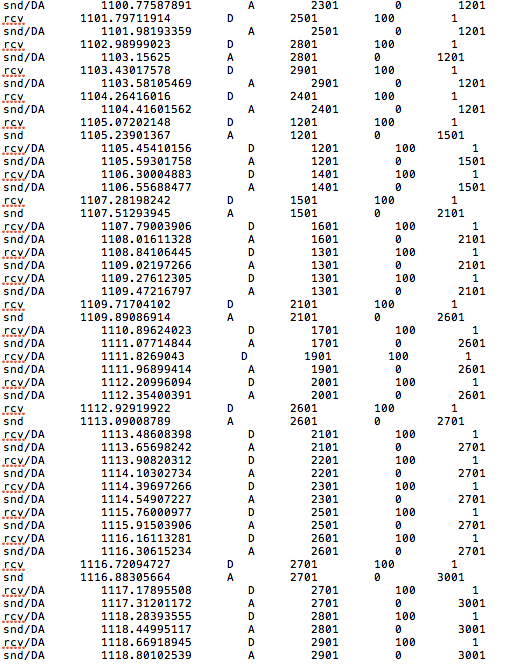


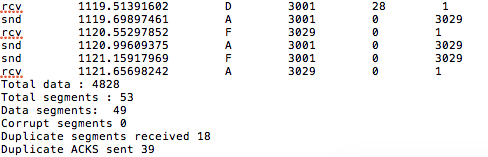


**5(a) Sender : pDrop = 0.3**

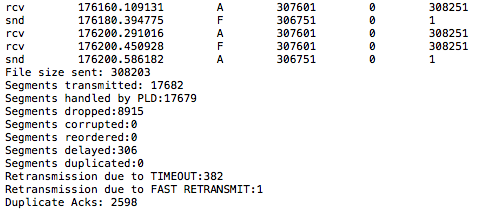


**5(a) Receiver: pDrop = 0.3**

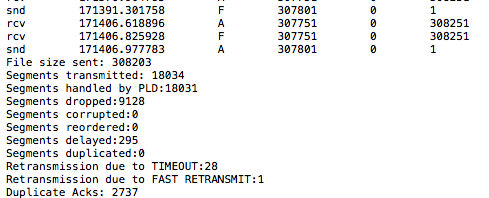




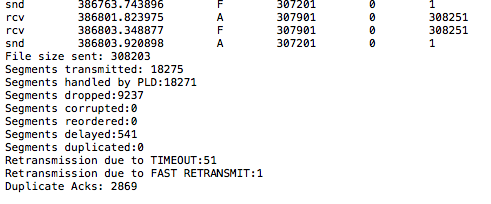
**5(b) gamma = 2**



**5(b) gamma = 4**



**5(b) gamma = 6**

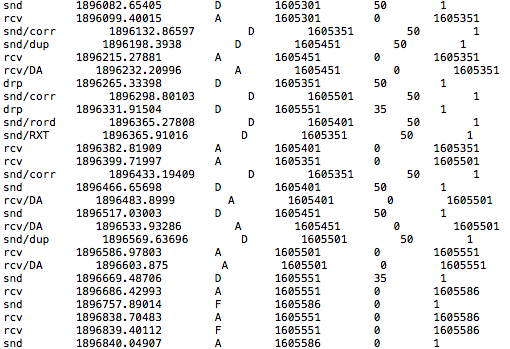
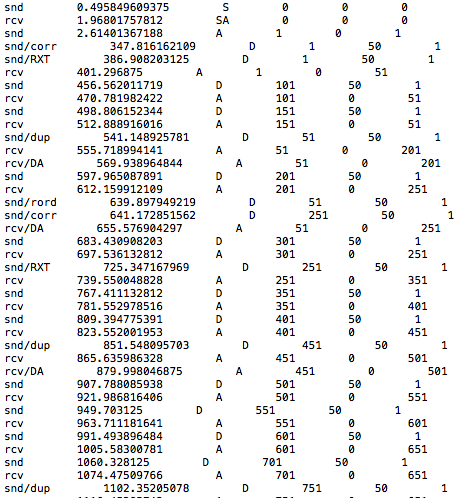


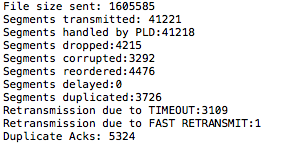
**5 (c) Sender**

Ending sequences

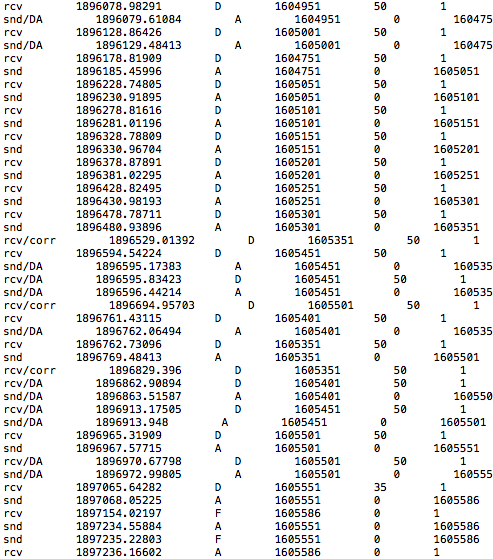
itial

Initial Sequences





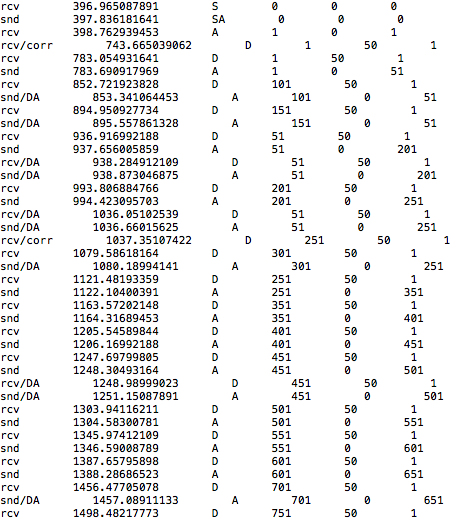
Stats

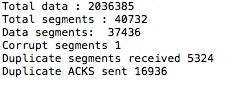
**5(c) Receiver**

Ending sequences

itial

Initial Sequences

****



Stats