Comp3331 Ass1

Riley Sutton, z5164867, UNSW October 2018

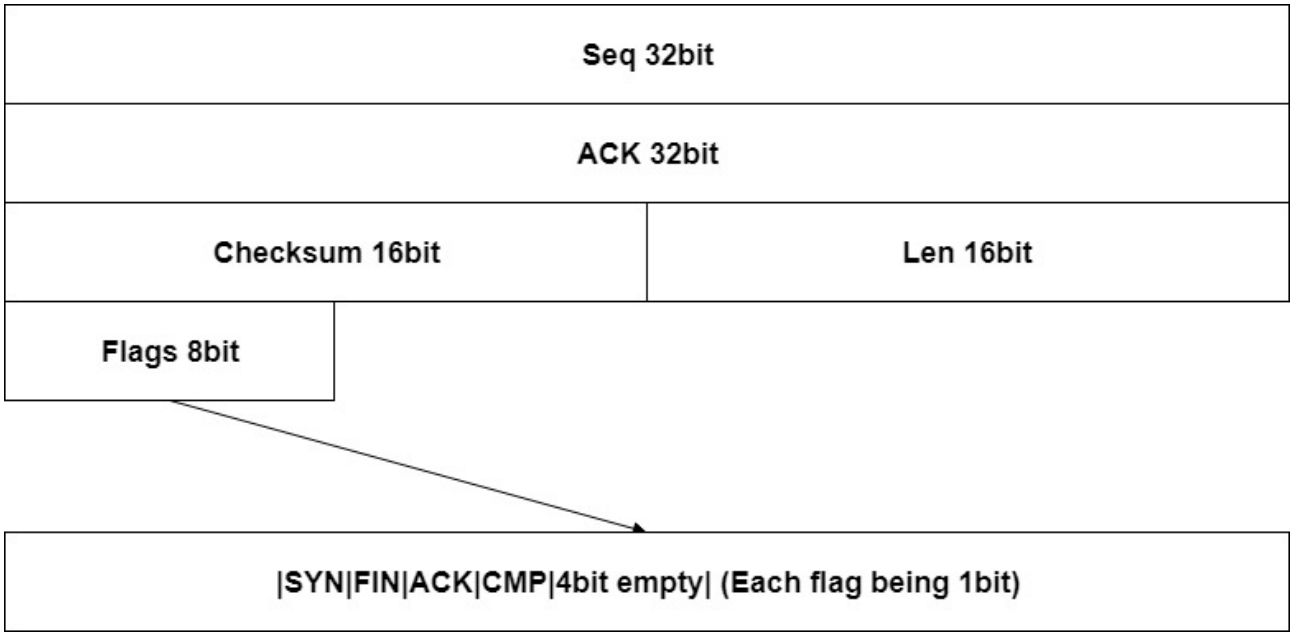
Contents

1 Part 1

I have implemented my version of STP using C. I made this choice since I am most comfortable at using C in comparison to Java and Python, although I was convinced to use Python early on but decided against it. As I started writing the project I realised that choosing C may have made implementing STP quite a lot harder. I started my design with an abstract packet and an RTP main class for handling the send and receive functionality. I implemented status codes into the RTP header and an abstract socket to handle the variables I would need to have for my protocol. The status codes I settled on were ACK, SYN, FIN, CMP. Most of these are self explanatory except for CMP which my protocol sends after a file has been fully transmitted, allowing for the possibility of multiple transmissions in one session. It also made it a lot easier to program in sections, which is what I did. The abstract socket handles local address, peer address, timeout, logging and the PLD. On initilisation of the socket the client is required to also initialise the PLD with all attached variables only because its not relevant to make the receiver do so. The packet header contains the definitions for the packet rep, packet head and functions to handle packets including the checksum. The head and body of the packet are allocated together in memory to save space and to also make sending and receiving packets easier, since they will also need to be adjacent in memory. The packet header also contains definitions for functions able to read, copy and allocate packets from a buffer, which also comes in handy. After some more designing of the program I decided I would need a list ADT, capable of handling the packets I receive and send since C is the only language among the allowed languages that does not have a list natively. Fortunately I wrote an arraylist in C before which the one in this program is heavily inspired from, although lacking in some of the features. ListRTP can expand, add elements, remove elements, insert elements and print diagnostics. I also wrote a timer for my program which similarly came in quite handy since every section which relies on waiting for an ACK uses it. The timer can be reset, set, checked and can run a sleep function for the duration of the timer. The logging section of the program allows for outputting the log file (with every variable attached) to stdout and saving the sender, receiver or all log files to a specific file. It also stores a list of logs which contain the time and certain aspects of the packet header in the order in which they occurred. The PLD module runs only on the sender side. It basically contains a wrapper for the send function which logs what is going on and also sends the packets (depending on rand). The main functions in the RTP files contain functions to connect, bind, send, receive and to close with respect the the socket. the send function will send a SYN, which will be replied with a SYN,ACK which is then replied with an ACK. The two programs then enter the send and recv routines in which the sender will initially send the entire window and wait for the timeout period for any ACKs, if it receives any ACKs then the timer is reset and the program will send the next packet in the sequence. The window is similar to GBN although it can split like SR, which allows for faster transfer. When an ACK is received on the senders side the corresponding packet is removed and freed from the list of packets to be sent. The first window size packets in the list are the ones currently waiting for an ACK. If the sender does not receive an ACK in time, it will re-transmit the entire window. I also included a check at the beginning of the wait loop to see if the packets have already timed out before they reach the loop. In this case the timeout will increase and all packets will immediately be re-transmitted. This improvement dramatically increased the effective bandwidth available for the protocol. The receiver does not use a PLD like the sender and as such when it receives a packet it will immediately reply with an ACK. The receiver has no timeout. All functions handle corrupted and oversized packets. Once the senders packet list is emptied, it will send a CMP to indicate the file is finished and that the receiver should start the process of outputting the data of the packets to a buffer. It is possible that the sender will enter the close method immediately, in which case the receiver will handle the close and then process the file. Otherwise the receiver will enter the await close method and close after the file is processed. As stated before, there is no requirement to close after a file is sent, only that the CMP is sent at the end of each of the files so the framework could be used to send multiple files. As per the spec the close will send a FIN, receive and ACK, receive a FIN and send an ACK. Fast re-transmit and retransmits showing in log files are two features I did not get working. The design of my project would not allow for fast retransmit to be functional without logging the acks for each packet. Logging the retransmits was something I forgot until it was too late and unfortunately would

have required a rewrite of the PLD and Logger since the retransmitted packets go through the PLD and there is no way to log it is a retransmit as well as another type of packet (eg drop or corr). It would mean I would need a buffer which I add strings to for each packet and the feature that the PLD can receive an indicator of whether the packet was a retrasmit. The count of retransmitted packets is sound as well as the protocols handling of them. ACK and SEQ numbers also slightly differ from TCP. Since connection, sending and receiving, and closing are all handled separately, I saw no reason to include a sequence number that progresses through the entirety of the connection. Instead each section starts at zero and ACKs are exactly seq + data for the packet, not seq + 1 if there is no data.

2 Part 2



SEQ holds the sequence number for the packet in question. ACK holds the acknowledgment number for use when the ACK flag is 1, otherwise it is unused. Checksum holds the checksum for the packet, the header is included in this calculation. Len holds the length of the data in the packet (not including the header) Flags hold the flags used in the packet, each being one bit wide. There is a section of 4 unused bits in the flag section.

3 Part 3

As stated before I could have changed the re-transmit logging and included fast re-transmit. Re-transmit could be realised if I changed the way PLD worked and Logging worked. The sending section of the code could be easily changed but the others would be significantly harder. It is also possible to scan through the log list before it is written for repeats in the sequence numbers but this could take a considerable amount of time. fast retransmit would require a major rework of the sending code to be able to detect a number of duplicate ACKS.

4 Part 4

As stated before I used some code I had written before (and not used in any assignment) which is located here `https://github.com/rileysut8991/rstd/blob/master/src/arrlist.c`. I also used checksum code which was inspired from urlhttps://locklessinc.com/articles/tcp$_{c}hecksum/, although I have modifie$

5 Part 5

All tests were done on my home computer.

5.1 Test0

In Test0 in both sections you can the see the effects of the sliding window in progress. Packets quickly become out of order since some are being dropped and the sender utilises the window space to send new packets after others have not been dropped. Since the delay is non existent the timeout drops to barely anything and the sender re-transmits the entire active window, although this only happens in part ii in my examples. Since part i was small and it was hard to see the effect of dropping in the first and last 20 packets, I included the whole logs to show it is dropping. Note that part ii also takes longer to transmit although the startup times of both programs make it difficult to determine by how much (around 3 seconds I think).

5.2 Test1

Test	i	ii	iii
Time	217	239	265
Packets	8318	7446	7067

In Test1 in all 3 tests I had some interesting results. Increasing gamma I believe increased the timeout which correspondingly raised the time taken proportionally but surprisingly lowered the amount of packets sent (minus the ones dropped but they were lower too). I believe that this is because with a higher timeout the protocol is more forgiving for extremely slow ACKs to be received and lowers the amount of packets needed but also increases the time taken for the test to complete. I also believe that the relationship between gamma and the time taken and packets sent is linearly proportional and another proportionality that resembles a logarithmic curve respectively, but there isn't enough information to make it extremely clear.

5.3 Test3

In Test3 the log files looked quite messy and long although it only took around 4 seconds to complete! A total of 41592 segments were sent in that short time-frame due to the protocol taking advantage of the non existent delay. Looking at the results, I would say that the corruption is the most expensive error to deal with since my protocol does not have a NAK and must then wait for a timeout, where every packet in the window is sent. From the sender log file it is apparent that drops and corruptions are followed closely with window re-transmits. Corruptions are worse because they take up more time to process on the receiver than dropped packets, which take no time. Reorders are not helpful but since the timer is reset whenever an ACK is received, the client can just have a window size of one smaller until it receives a reply for the really slow packet. Duplicates are simply ignored by the receiver and don't slow down the transfer much at all.

6 Appendix

6.1 Test0 i receiver

```
1  rcv 1.326773 S 0 0 0
2  snd 1.326813 SA 0 0 0
3  rcv 1.326846 A 0 0 0
4  rcv 2.084192  0 100 0
5  snd 2.084191 A 0 0 100
6  rcv 2.084200  200 100 0
7  snd 2.084200 A 0 0 300
8  rcv 2.084208  300 100 0
9  snd 2.084208 A 0 0 400
10 rcv 2.084222  400 100 0
11 snd 2.084221 A 0 0 500
12 rcv 2.084228  500 100 0
13 snd 2.084227 A 0 0 600
14 rcv 2.084233  600 100 0
15 snd 2.084233 A 0 0 700
16 rcv 2.084239  700 100 0
17 snd 2.084239 A 0 0 800
18 rcv 2.084245  800 100 0
19 snd 2.084245 A 0 0 900
20 rcv 2.084251  900 100 0
21 snd 2.084250 A 0 0 1000
22 rcv 2.084257  1000 100 0
23 snd 2.084256 A 0 0 1100
24 rcv 2.084262  1100 100 0
25 snd 2.084262 A 0 0 1200
26 rcv 2.084268  1200 100 0
27 snd 2.084268 A 0 0 1300
28 rcv 2.084274  1300 100 0
29 snd 2.084273 A 0 0 1400
30 rcv 2.084279  1400 100 0
31 snd 2.084279 A 0 0 1500
32 rcv 2.084285  1500 100 0
33 snd 2.084285 A 0 0 1600
34 rcv 2.084294  1600 100 0
35 snd 2.084294 A 0 0 1700
36 rcv 2.084300  1700 100 0
37 snd 2.084299 A 0 0 1800
38 rcv 2.084305  1800 100 0
39 snd 2.084305 A 0 0 1900
40 rcv 2.084311  1900 100 0
41 snd 2.084311 A 0 0 2000
42 rcv 2.084317  2000 100 0
43 snd 2.084317 A 0 0 2100
44 rcv 2.084325  2100 100 0
45 snd 2.084324 A 0 0 2200
46 rcv 2.084332  2200 100 0
47 snd 2.084332 A 0 0 2300
48 rcv 2.084339  2300 100 0
49 snd 2.084339 A 0 0 2400
50 rcv 2.084346  2400 100 0
51 snd 2.084346 A 0 0 2500
52 rcv 2.084354  2500 100 0
53 snd 2.084353 A 0 0 2600
54 rcv 2.084366  2700 100 0
55 snd 2.084365 A 0 0 2800
56 rcv 2.084374  2800 100 0
```

```
snd 2.084374 A 0 0 2900
rcv 2.084385  3000 28 0
snd 2.084384 A 0 0 3028
rcv 2.844785  100 100 0
snd 2.844784 A 0 0 200
rcv 2.844794  2600 100 0
snd 2.844794 A 0 0 2700
rcv 2.844800  2900 100 0
snd 2.844799 A 0 0 3000
rcv 2.844810 C 3029 0 0
snd 2.844814 A 0 0 3029
rcv 2.844825 F 0 0 0
snd 2.844829 A 0 0 0
snd 3.427662 F 0 0 0
rcv 3.427683 A 0 0 0
==============================
Size of the file (in Bytes) 3028
Segments received 36
Segments bit errors 0
Segments duplicated 0
Duplicate ACKs 0
==============================
```

6.2 Test0 i sender

```
snd 0.000097 S 0 0 0
snd 1.500089 S 0 0 0
snd 3.000161 S 0 0 0
snd 4.500092 S 0 0 0
rcv 4.500153 SA 0 0 0
snd 4.500172 A 0 0 0
snd 5.250287  0 100 0
snd/drop 5.250288  100 100 0
snd 5.250291  200 100 0
snd 5.250295  300 100 0
snd 5.250297  400 100 0
snd 5.250309  500 100 0
rcv 5.250309 A 0 0 100
snd 5.250316  600 100 0
rcv 5.250317 A 0 0 300
snd 5.250324  700 100 0
rcv 5.250324 A 0 0 400
snd 5.250337  800 100 0
rcv 5.250337 A 0 0 500
snd 5.250344  900 100 0
rcv 5.250345 A 0 0 600
snd 5.250351  1000 100 0
rcv 5.250352 A 0 0 700
snd 5.250358  1100 100 0
rcv 5.250359 A 0 0 800
snd 5.250366  1200 100 0
rcv 5.250366 A 0 0 900
snd 5.250373  1300 100 0
rcv 5.250373 A 0 0 1000
snd 5.250380  1400 100 0
rcv 5.250381 A 0 0 1100
snd 5.250387  1500 100 0
rcv 5.250388 A 0 0 1200
snd 5.250395  1600 100 0
```

```
rcv 5.250395 A 0 0 1300
snd 5.250401  1700 100 0
rcv 5.250402 A 0 0 1400
snd 5.250409  1800 100 0
rcv 5.250409 A 0 0 1500
snd 5.250416  1900 100 0
rcv 5.250417 A 0 0 1600
snd 5.250423  2000 100 0
rcv 5.250424 A 0 0 1700
snd 5.250431  2100 100 0
rcv 5.250431 A 0 0 1800
snd 5.250438  2200 100 0
rcv 5.250439 A 0 0 1900
snd 5.250445  2300 100 0
rcv 5.250446 A 0 0 2000
snd 5.250452  2400 100 0
rcv 5.250453 A 0 0 2100
snd 5.250459  2500 100 0
rcv 5.250460 A 0 0 2200
snd/drop 5.250464  2600 100 0
rcv 5.250465 A 0 0 2300
snd 5.250471  2700 100 0
rcv 5.250472 A 0 0 2400
snd 5.250478  2800 100 0
rcv 5.250479 A 0 0 2500
snd/drop 5.250483  2900 100 0
rcv 5.250484 A 0 0 2600
snd 5.250490  3000 28 0
rcv 5.250491 A 0 0 2800
rcv 5.250495 A 0 0 2900
rcv 5.250499 A 0 0 3028
snd 6.000583  100 100 0
snd 6.000588  2600 100 0
snd 6.000591  2900 100 0
rcv 6.000603 A 0 0 200
rcv 6.000613 A 0 0 2700
rcv 6.000618 A 0 0 3000
snd 6.000622 C 3029 0 0
rcv 6.000633 A 0 0 3029
snd 6.000638 F 0 0 0
rcv 6.000648 A 0 0 0
rcv 6.578822 F 0 0 0
snd 6.578833 A 0 0 0
==============================
Size of the file (in Bytes) 3028
Segments sent 39
Segments dropped 3
Segments corrupted 0
Segments re-ordered 0
Segments duplicated 0
Segments delayed 0
Duplicate ACKs 0
==============================
```

6.3 Test0 ii receiver

rcv 2.693072 S 0 0 0
    snd 2.693114 SA 0 0 0
    rcv 2.693145 A 0 0 0

rcv 3.451257 0 100 0
snd 3.451256 A 0 0 100
rcv 3.451266 200 100 0
snd 3.451265 A 0 0 300
rcv 3.451277 400 100 0
snd 3.451276 A 0 0 500
rcv 3.451290 500 100 0
snd 3.451290 A 0 0 600
rcv 3.451301 700 100 0
snd 3.451300 A 0 0 800
rcv 3.451310 800 100 0
snd 3.451309 A 0 0 900
rcv 3.451320 900 100 0
snd 3.451320 A 0 0 1000
rcv 3.451330 1000 100 0
snd 3.451329 A 0 0 1100
rcv 3.451347 1200 100 0
snd/DA 8.567959 A 0 0 307900
rcv 8.567984 307600 100 0
snd 8.567984 A 0 0 307700
rcv 8.567992 307700 100 0
snd 8.567991 A 0 0 307800
rcv 8.568002 308100 100 0
snd 8.568001 A 0 0 308200
rcv 8.568025 C 308204 0 0
snd 8.568028 A 0 0 308204
rcv 8.568072 F 0 0 0
snd 8.568075 A 0 0 0
snd 9.150911 F 0 0 0
rcv 9.150932 A 0 0 0
==========
Size of the file (in Bytes) 308203
Segments received 3282
Segments bit errors 0
Segments duplicated 194
Duplicate ACKs 194
==========

6.4 Test0 ii sender

snd 0.002097 S 0 0 0
rcv 0.002159 SA 0 0 0
snd 0.002174 A 0 0 0
snd 0.753097 0 100 0
snd/drop 0.753102 100 100 0
snd 0.753112 200 100 0
snd/drop 0.753116 300 100 0
snd 0.753121 400 100 0
snd 0.753138 500 100 0
rcv 0.753139 A 0 0 100
snd/drop 0.753146 600 100 0
rcv 0.753147 A 0 0 300
snd 0.753156 700 100 0
rcv 0.753156 A 0 0 500

snd 0.753165 800 100 0
rcv 0.753166 A 0 0 600
snd 0.753175 900 100 0
rcv 0.753176 A 0 0 800
snd 0.753185 1000 100 0
rcv 0.753186 A 0 0 900
snd 5.863510 308100 100 0
rcv 5.863526 A 0 0 307700
rcv 5.863530 A 0 0 307800
rcv 5.863535 A 0 0 308200
snd 5.863541 C 308204 0 0
rcv 5.863551 A 0 0 308204
snd 5.863585 F 0 0 0
rcv 5.863597 A 0 0 0
rcv 6.445698 F 0 0 0
snd 6.445708 A 0 0 0
==========
Size of the file (in Bytes) 308203
Segments sent 3282
Segments dropped 1382
Segments corrupted 0
Segments re-ordered 0
Segments duplicated 0
Segments delayed 0
Duplicate ACKs 194
==========

6.5 Test1 i receiver

rcv 0.504072 S 0 0 0
snd 0.504129 SA 0 0 0
rcv 0.504188 A 0 0 0
rcv 1.102285 150 50 0
snd 1.102284 A 0 0 200
rcv 1.102298 400 50 0
snd 1.102297 A 0 0 450
rcv 1.102307 450 50 0
snd 1.102306 A 0 0 500
rcv 1.696451 300 50 0
snd 1.696450 A 0 0 350
rcv 2.089807 0 50 0
snd 2.089806 A 0 0 50
rcv 2.089821 350 50 0
snd 2.089820 A 0 0 400
rcv 2.089828 500 50 0
snd 2.089828 A 0 0 550
rcv 2.098564 600 50 0
snd 2.098563 A 0 0 650
rcv 2.098606 700 50 0
snd/DA 214.875442 A 0 0 307750
rcv 214.875450 307750 50 0
snd 214.875449 A 0 0 307800
rcv 214.875464 308100 50 0
snd/DA 214.875463 A 0 0 308150

rcv 214.875471 308200 3 0
snd/DA 214.875470 A 0 0 308203
rcv 214.875506 C 308204 0 0
snd 214.875511 A 0 0 308204
rcv 214.875557 F 0 0 0
snd 214.875563 A 0 0 0
snd 215.458399 F 0 0 0
rcv 215.458420 A 0 0 0
==========
Size of the file (in Bytes) 308203
Segments received 8316
Segments bit errors 0
Segments duplicated 2146
Duplicate ACKs 2146
==========

6.6 Test1 i sender

snd 0.001795 S 0 0 0
snd 1.501793 S 0 0 0
snd 3.001800 S 0 0 0
rcv 3.001878 SA 0 0 0
snd 3.001900 A 0 0 0
snd/drop 3.596873 0 50 0
snd/drop 3.596878 50 50 0
snd/drop 3.596881 100 50 0
snd 3.596903 150 50 0
snd/drop 3.596908 200 50 0
snd/drop 3.596913 250 50 0
snd/drop 3.596917 300 50 0
snd/drop 3.596920 350 50 0
snd 3.596928 400 50 0
snd 3.596935 450 50 0
snd/drop 3.596959 500 50 0
rcv 3.596960 A 0 0 200
snd/drop 3.596971 550 50 0
rcv 3.596972 A 0 0 450
snd/drop 3.596982 600 50 0
rcv/DA 217.382419 A 0 0 307750
rcv 217.382423 A 0 0 307800
snd 217.382427 C 308204 0 0
rcv 217.382432 A 0 0 308150
rcv 217.382436 A 0 0 308203
rcv 217.382440 A 0 0 308204
snd 217.382477 F 0 0 0
rcv 217.382491 A 0 0 0
rcv 217.964804 F 0 0 0
snd 217.964815 A 0 0 0
==========
Size of the file (in Bytes) 308203
Segments sent 8318
Segments dropped 8308
Segments corrupted 0
Segments re-ordered 0

Segments duplicated 0
Segments delayed 1640
Duplicate ACKs 2144
==========

6.7 Test1 ii receiver

rcv 2.554042 S 0 0 0
snd 2.554110 SA 0 0 0
rcv 2.554135 A 0 0 0
rcv 3.328215 150 50 0
snd 3.328214 A 0 0 200
rcv 3.328239 400 50 0
snd 3.328237 A 0 0 450
rcv 3.328268 450 50 0
snd 3.328267 A 0 0 500
rcv 4.074393 300 50 0
snd 4.074391 A 0 0 350
rcv 4.558826 0 50 0
snd 4.558825 A 0 0 50
rcv 4.558843 350 50 0
snd 4.558842 A 0 0 400
rcv 4.558850 500 50 0
snd 4.558849 A 0 0 550
rcv 4.567573 600 50 0
snd 4.567572 A 0 0 650
rcv 4.567611 700 50 0
snd/DA 241.010419 A 0 0 308000
rcv 241.010427 308050 50 0
snd/DA 241.010426 A 0 0 308100
rcv 241.010436 308200 3 0
snd 241.010435 A 0 0 308203
rcv 241.775224 307600 50 0
snd 241.775223 A 0 0 307650
rcv 241.775247 C 308204 0 0
snd 241.775254 A 0 0 308204
rcv 241.775294 F 0 0 0
snd 241.775301 A 0 0 0
snd 242.358133 F 0 0 0
rcv 242.358153 A 0 0 0
==========
Size of the file (in Bytes) 308203
Segments received 7446
Segments bit errors 0
Segments duplicated 1276
Duplicate ACKs 1276
==========

6.8 Test1 ii sender

snd 0.002074 S 0 0 0
rcv 0.002159 SA 0 0 0
snd 0.002172 A 0 0 0
snd/drop 0.753768 0 50 0

snd/drop 0.753773 50 50 0
snd/drop 0.753778 100 50 0
snd 0.753795 150 50 0
snd/drop 0.753799 200 50 0
snd/drop 0.753805 250 50 0
snd/drop 0.753809 300 50 0
snd/drop 0.753814 350 50 0
snd 0.753821 400 50 0
snd 0.753827 450 50 0
snd/drop 0.753893 500 50 0
rcv 0.753894 A 0 0 200
snd/drop 0.753910 550 50 0
rcv 0.753911 A 0 0 450
snd/drop 0.753925 600 50 0
rcv 0.753925 A 0 0 500
snd/drop 1.504038 0 50 0
snd/drop 238.856401 307600 50 0
snd/drop 239.065323 307600 50 0
snd 239.274262 307600 50 0
rcv 239.274283 A 0 0 307650
snd 239.274299 C 308204 0 0
rcv 239.274313 A 0 0 308204
snd 239.274344 F 0 0 0
rcv 239.274359 A 0 0 0
rcv 239.857001 F 0 0 0
snd 239.857010 A 0 0 0
==========
Size of the file (in Bytes) 308203
Segments sent 7446
Segments dropped 7387
Segments corrupted 0
Segments re-ordered 0
Segments duplicated 0
Segments delayed 1481
Duplicate ACKs 1276
==========

6.9 Test1 iii receiver

rcv 1.741127 S 0 0 0
snd 1.741243 SA 0 0 0
rcv 1.741343 A 0 0 0
rcv 2.646399 150 50 0
snd 2.646398 A 0 0 200
rcv 2.646409 400 50 0
snd 2.646408 A 0 0 450
rcv 2.646420 450 50 0
snd 2.646419 A 0 0 500
rcv 3.548642 300 50 0
snd 3.548640 A 0 0 350
rcv 4.132733 0 50 0
snd 4.132732 A 0 0 50
rcv 4.132751 350 50 0
snd 4.132750 A 0 0 400

rcv 4.132758 500 50 0
snd 4.132758 A 0 0 550
rcv 4.141493 600 50 0
snd 4.141493 A 0 0 650
rcv 4.141533 700 50 0
snd 265.912572 A 0 0 308203
rcv 266.340481 308150 50 0
snd 266.340480 A 0 0 308200
rcv 266.361934 308000 50 0
snd 266.361933 A 0 0 308050
rcv 266.678473 307400 50 0
snd 266.678472 A 0 0 307450
rcv 266.678484 C 308204 0 0
snd 266.678488 A 0 0 308204
rcv 266.678533 F 0 0 0
snd 266.678537 A 0 0 0
snd 267.261382 F 0 0 0
rcv 267.261404 A 0 0 0
==========
Size of the file (in Bytes) 308203
Segments received 7067
Segments bit errors 0
Segments duplicated 897
Duplicate ACKs 897
==========

6.10 Test1 iii sender

snd 0.001554 S 0 0 0
rcv 0.001723 SA 0 0 0
snd 0.001780 A 0 0 0
snd/drop 0.909486 0 50 0
snd/drop 0.909490 50 50 0
snd/drop 0.909494 100 50 0
snd 0.909518 150 50 0
snd/drop 0.909521 200 50 0
snd/drop 0.909524 250 50 0
snd/drop 0.909527 300 50 0
snd/drop 0.909530 350 50 0
snd 0.909538 400 50 0
snd 0.909544 450 50 0
snd/drop 0.909562 500 50 0
rcv 0.909563 A 0 0 200
snd/drop 0.909574 550 50 0
rcv 0.909574 A 0 0 450
snd/drop 0.909584 600 50 0
rcv 0.909585 A 0 0 500
snd/drop 1.816123 0 50 0
snd 264.950916 308000 50 0
rcv 264.950930 A 0 0 308050
snd 265.267745 307400 50 0
rcv 265.267767 A 0 0 307450
snd 265.267773 C 308204 0 0
rcv 265.267783 A 0 0 308204

snd 265.267819 F 0 0 0
rcv 265.267830 A 0 0 0
rcv 265.850869 F 0 0 0
snd 265.850879 A 0 0 0
==========
Size of the file (in Bytes) 308203
Segments sent 7067
Segments dropped 6999
Segments corrupted 0
Segments re-ordered 0
Segments duplicated 0
Segments delayed 1412
Duplicate ACKs 897
==========

6.11 Test2 receiver

rcv 2.757929 S 0 0 0
snd 2.757972 SA 0 0 0
rcv 2.758453 A 0 0 0
rcv 3.514189 0 50 0
snd 3.514187 A 0 0 50
rcv 3.514200 50 50 0
snd 3.514199 A 0 0 100
rcv 3.514211 100 50 0
snd 3.514210 A 0 0 150
rcv 3.514227 100 50 0
snd/DA 3.514226 A 0 0 150
rcv 3.514234 150 50 0
snd 3.514234 A 0 0 200
rcv/corr 3.514238 0 0 0
rcv 3.514270 250 50 0
snd 3.514267 A 0 0 300
rcv 3.514279 300 50 0
snd 3.514279 A 0 0 350
rcv 3.514285 400 50 0
snd 3.514285 A 0 0 450
snd 6.840970 A 0 0 1604550
rcv 6.840977 1605200 50 0
snd 6.840977 A 0 0 1605250
rcv 6.840983 1605450 50 0
snd/DA 6.840983 A 0 0 1605500
rcv 6.840989 1605200 50 0
snd/DA 6.840988 A 0 0 1605250
rcv 6.841005 C 1605586 0 0
snd 6.841008 A 0 0 1605586
rcv 6.841099 F 0 0 0
snd 6.841102 A 0 0 0
snd 7.423991 F 0 0 0
rcv 7.424019 A 0 0 0
==========
Size of the file (in Bytes) 1605585
Segments received 41592
Segments bit errors 2541

Segments duplicated 6934
Duplicate ACKs 6934
==========

6.12 Test2 sender

snd 0.008604 S 0 0 0
rcv 0.008747 SA 0 0 0
snd 0.008763 A 0 0 0
snd 0.764691 0 50 0
snd 0.764702 50 50 0
snd/dup 0.764710 100 50 0
snd/dup 0.764715 100 50 0
snd 0.764720 150 50 0
snd/corr 0.764726 200 50 0
snd 0.764733 250 50 0
snd 0.764739 300 50 0
snd 0.764747 400 50 0
snd 0.764752 450 50 0
snd 0.764916 500 50 0
rcv 0.764918 A 0 0 50
snd 0.764989 550 50 0
snd/reord 0.764992 350 50 0
rcv 0.764993 A 0 0 100
snd 0.765038 600 50 0
rcv 0.765039 A 0 0 150
rcv 4.072340 A 0 0 1604550
rcv 4.072345 A 0 0 1605250
snd 4.072349 C 1605586 0 0
rcv 4.072353 A 0 0 1605500
rcv 4.072357 A 0 0 1605250
rcv 4.072361 A 0 0 1605586
snd 4.072442 F 0 0 0
rcv 4.072452 A 0 0 0
rcv 4.654921 F 0 0 0
snd 4.654939 A 0 0 0
==========
Size of the file (in Bytes) 1605585
Segments sent 41592
Segments dropped 3327
Segments corrupted 2541
Segments re-ordered 2385
Segments duplicated 2924
Segments delayed 0
Duplicate ACKs 6932
==========