COMP3331 Assignment : z5311696

Program design & how it works

Sender.py

A socket is firstly opened. The 3-way handshake is initiated. A SYN is sent to the receiver. Once a SYN-ACK is received, the final ACK of the handshake is sent. If the handshake is successful, 2 threads are started: one thread to send the segments (sendThread) and one thread to receive the ACK segments from the receiver (recvThread). There is a conditional statement stating that the sendThread must be alive before the recvThread is started.

In sendThread, there is an infinite loop. When in this loop, a thread lock must first be acquired. Once the lock is acquired, the sendThread populates the segment window until it reaches its maximum window size (MWS). If the window is initially empty, once it adds one segment to the window and sends it to the receiver, it will start a timer. Once the window is populated to its MWS (window is full), it will release the lock so the recvThread can start receiving ACKs.

In the recvThread, it waits for an ACK to arrive from the receiver. Once an ACK has been received, it waits for the thread lock. Once the lock is acquired, it starts to operate on the ACK. Firstly, it stops the current threading timer. It then takes the ACK number of the received ACK and checks it against the current ACK number. The current ACK number is the ACK number of the previous ACK received (the most recent ACK up until the new ACK received). It then checks if the current ACK number and the received ACK number are the same. If they are, the duplicate ACK counter is incremented. If the duplicate ACK counter reaches 3, a fast retransmit of the oldest segment in the window occurs. In this fast retransmit, a threading timer is started for this retransmit. Else, this means that the new ACK received contains a new ACK number (has not been received before) and so the current ACK number changes to the ACK number of the received ACK. After this, the recvThread loops through the window to remove segments that have a smaller sequence number than the current ACK number. This is because if the sequence number of the segment is smaller than the current ACK number, that means the segment has been acknowledged. After, the recvThread checks if there are any segments in the window. Note, when an ACK was received, the recvThread firstly cancelled the timer. Hence, after updating the window (by removing acknowledged segments), it now needs to start a timer for the oldest segment in the window. After one ACK has been operated on, the recvThread releases the lock so the sendThread can acquire the lock and re-populate the window until it reaches the MWS.

This operation between the two threads ends only when the following occur:

The sendThread will terminate once the file has been read until the end. That is, there is no more data to send to the receiver. At this point, the threading timer and fast retransmission will take care of the remaining segments if they are dropped. That is, if a timeout occurs, the threading timer will start a new threading timer and retransmit the oldest segment in the current window. As for the fast retransmit, this is handled by the recvThread. When 3 consecutive duplicate ACKs are received, the recvThread will call a function that retransmits the oldest segment in the window (i.e. the segment the receiver is waiting to receive).

The recvThread will terminate once the current ACK number is equal to the last expected ACK number (which was calculated prior to starting the thread). Note that the last expected ACK number is for the last segment transmitted with data, not the ACK number of the final ACK from the receiver to close the connection.

Finally, after both threads have finished their jobs, the connection teardown begins. A FIN segment is sent to the receiver. When a FIN-ACK is received, a final ACK is sent to the receiver and then the socket is closed, thus ending the connection between the sender and receiver.

COMP3331 Assignment : z5311696

Receiver.py

A socket is opened, and the receiver waits for a SYN from the sender. When a SYN is received, it sends a SYN-ACK. After this, when an ACK is received (completing the handshake), it enters an infinite loop. In the loop, the receiver waits for a segment. When a segment is received, it checks if the sequence number of the segment is equal to the current ACK number. Note that the current ACK number is the ACK number of the receiver (not the sender) in the previous ACK it sent to the sender. The following occurs:

If the sequence number of the segment is NOT equal to the current ACK number (of the receiver) and the sequence number is bigger than the most recent ACK number (which is the ACK number for the most recent segment added to the receiver buffer), then the segment is added to the receiver buffer. This is because this segment has not been received before (it is new).

On the other hand, if the sequence number of the received segment is equal to the receiver's current ACK number (i.e. the segment was the one it was expecting), the data of the segment is written into the file and the current ACK number will be updated (i.e. the length of data in the segment is added to the current ACK number). Afterwards, the receiver will sort its receiver buffer and loop through the buffer. If there is a segment in the buffer where its sequence number is equal to the current ACK number (i.e. the segment is the next expected segment), its data will be written to the file and the current ACK will be updated. Finally, the segment is removed from the buffer.

Finally, after either of the two (above) events, the receiver sends an ACK back to the sender. The ACK number of this segment is the current ACK number (which was mentioned above).

However, if the segment received has the FIN bit flipped, the receiver will return a FIN-ACK and break from the loop. Outside the loop, the receiver closes the file it was writing to. It then waits for the final ACK from the sender before closing the socket.

Design trade-offs

The receiving and sending threads usually do not run concurrently. This is because I need to lock the sending thread needs to acquire a lock to modify variables also modified in the receiving lock. The main variable is the sender's sliding window. This is because in the sending thread, the sender needs to populate this window with segments it has sent. In the receiving thread, the sender needs to remove packets from the window that have been acknowledged. To avoid concurrent modification, threads must attain a lock before operating. Hence, the two threads do not run simultaneously.

Circumstances where exceptions may occur

Due to the above trade-off, a small timeout value (from experiments and testing: less than 5ms) will result in exceptions due to concurrent modification. That is, in the threading timer, there is a function called "retransmitPacket". This function is required to retransmit the oldest packet in the window and start a new timer for this packet. How the function works is that it will need to sort the window in order to find the oldest packet in the window. However, the same window is also being modified by the sending thread and receiving thread. Since the timing thread does not have a lock, it can call the retransmitPacket function which will then sort the window of packets while the receiving or sending thread may be modifying it. However, even when these exceptions are called, the program still works as expected. That is, the file is still transferred correctly, as tested by the diff command in the linux terminal.

1. **A brief discussion of how you have implemented the PTP protocol. Provide a list of features that you have successfully implemented. In case you have not been able to get certain features of PTP working, you should also mention that in your report.**

Program has been implemented with **python2.**

All features (from the specification) were implemented. The following are the key features that were implemented:

- Three-way handshake
- Connection termination
- PL module
- Cumulative acknowledgement
- Fast retransmission
- Retransmission on timeout
- Sliding window (with size MWS provided in the command line)
- Receiver buffer implemented

2. **A detailed diagram of your PTP header and a quick explanation of all fields (similar to the diagrams that we have used in the lectures to understand TCP/UDP headers).**

Diagram of the PTP header:

SYN	**FIN**	**ACK**	**DATA**
Sequence number	**ACK number**	**Amount of data**	

Actual segment:

```
      {
        'SYNbit': SYNbit,
        'FINbit': FINbit,
        'seq': seq,
        'ACKbit': ACKbit,
        'ack': ack,
        'DATAbit': DATAbit,
        'dataAmount': dataAmount,
        'data': data
        }
```

Description of the fields:
SYNbit = flag indicating whether the segment was a SYN.
FINbit = flag indicating whether the segment was a FIN.
seq = sequence number of the segment.
ACKbit = flag indicating whether the segment was an ACK.
DATAbit = flag indicating whether the segment contains a payload/data.
ack = ACK number of the segment.
dataAmount = the length of the payload.
data = the payload/data contained in the segment.

3. **Answer the following questions:**

(a) Use the following parameter setting: pdrop = 0.1, MWS = 500 bytes, MSS = 50 bytes, seed = 300. Explain how you determine a suitable value for timeout. Note that you may need to experiment with different timeout values to answer this question. Justify your answer.

A suitable value for timeout must ensure the packet can be received by the receiver before a timeout occurs. Firstly, I estimated (via printing the time in the program) it took for the sender to send a segment and the receiver to receive it. That is, the time for a one-way trip. Then, I tested my program with a file of size 1 byte (1 character) and found that a timeout value of 0.0004 milliseconds was the smallest timeout value that will allow the packets to still be transported.

However, this timeout value was then run against a bigger file with a bigger window-size. Intuitively, the bigger the file, the more time that is required in between sending all the segments before acknowledging an ACK from the receiver. Hence, a more suitable timeout value was then estimated to be 10ms. This was tested against multiple large files. This timeout allows all segments (in a reasonably sized window, roughly 500 bytes) to be sent and received before a timeout occurs. Again, for my program, the timeout value varies depending on the window size of the protocol. This is because when I set a timer for a segment sent, I then append it to the window and continue appending segments to the window until the window is full and then hand over control to the receiving thread.

With the timeout value that you have selected, run an experiment with your PTP programs transferring the file test1.txt (available on the assignment webpage). Show the sequence of PTP packets that are observed at the receiver. It is sufficient to just indicate the sequence numbers of the PTP packets that have arrived. You do not have to indicate the payload contained in the PTP packets (i.e., the text). Run an additional experiment with pdrop = 0.3, transferring the same file (test1.txt). In your report, discuss the resulting packet sequences of both experiments indicating where dropping occurred. Also, in the appendix section show the packet sequences of all the experiments.

There were more consecutive drops when pdrop = 0.3 than pdrop = 0.1 Evidently, as the pdrop of the second test was larger than the first test (0.3 in test-2, 0.1 in test-1), this resulted in more packets being lost in the second test. In the second test, 285 packets were dropped whereas in the first test, 66 packets were dropped. As a result of the increase in dropped packets, this meant there was likely more ‘gaps’ in between the segments sent. Increase in ‘gaps’ of segments sent means the receiver will need to send more duplicate ACKs. This is shown in the appendix, where in first test, the number of duplicate ACKs received was 388 whereas in the second test, the number of duplicate ACKs received was 463.

(b) Let Tcurrent represent the timeout value that you have chosen in part (a). Set pdrop = 0.1, MWS = 500 bytes, MSS = 50 bytes, seed = 300 and run three experiments with the following different timeout values:

i. Tcurrent

ii. 4 × Tcurrent

iii. Tcurrent/4

and transfer the file test2.txt (available on the assignment webpage) using PTP. Show a table that indicates how many PTP packets were transmitted (this should include retransmissions) in total and how long the overall transfer took. Discuss the results.

	# of PTP packets transmitted	**Overall time of transfer**
i. Tcurrent	6165	8.93 seconds
ii. 4 x Tcurrent	6137	16 seconds
iii. Tcurrent / 4	6787	16.06 seconds

As estimated from part a, timeout = 10ms is a good estimate for the transfer of the file as seen in the table. That is, it has the shortest overall transfer time. This is because it provides sufficient time for the packets to be sent and then received.

In part ii, where the timeout = 40ms, the overall time of transfer is longer than in part i. This is because the protocol took too long to respond to the dropped packets. That is, it took too long to detect a packet was dropped and therefore retransmission had to happen.

In part iii, where the timeout = 2.5ms, the overall time of transfer is again longer than in part i. This is because packets are being retransmitted without ample time for the sender to receive the ACKs. Also, this is due to how I wrote my program (where the thread is locked when populating the sender's window). When the sender is adding packets to its window and sending them out (as well as starting a timer for the oldest packet in the window), the process of populating the window and sending the segments takes more time than the timeout itself. Hence, unnecessary retransmissions occur. This causes unnecessary ACKs from the receiver which need to be operated on in the sender program. Due to the unnecessary overhead, this increases the transfer time.

<u>Appendix</u>

Sequence of segments at <u>receiver</u> (Q3 a.): pdrop = 0.1

First 20 segments:

rcv	7.81	S	1000	0	0
snd	7.81	SA	2000	0	1001
rcv	7.81	A	1001	0	2001
rcv	7.82	D	1001	50	2001
snd	7.82	A	2001	0	1051
rcv	7.82	D	1051	50	2001
snd	7.82	A	2001	0	1101
rcv	7.82	D	1151	50	2001
snd	7.82	A	2001	0	1101
rcv	7.82	D	1201	50	2001
snd	7.82	A	2001	0	1101
rcv	7.82	D	1251	50	2001
snd	7.82	A	2001	0	1101
rcv	7.82	D	1301	50	2001
snd	7.82	A	2001	0	1101
rcv	7.82	D	1351	50	2001
snd	7.82	A	2001	0	1101
rcv	7.82	D	1401	50	2001
snd	7.82	A	2001	0	1101
rcv	7.82	D	1451	50	2001

COMP3331 Assignment : z5311696

Last 20 segments:

```
rcv     8.43    D       33501   50      2001
snd     8.43    A       2001    0       33401
rcv     8.43    D       33551   50      2001
snd     8.43    A       2001    0       33401
rcv     8.44    D       33601   50      2001
snd     8.44    A       2001    0       33401
rcv     8.44    D       33651   50      2001
snd     8.44    A       2001    0       33401
rcv     8.44    D       33701   50      2001
snd     8.44    A       2001    0       33401
rcv     8.44    D       33751   17      2001
snd     8.44    A       2001    0       33401
rcv     8.44    D       33401   50      2001
snd     8.44    A       2001    0       33451
rcv     8.44    D       33401   50      2001
snd     8.44    A       2001    0       33451
rcv     8.45    D       33451   50      2001
snd     8.45    A       2001    0       33768
rcv     8.45    F       33768   0       2001
snd     8.45    FA      2001    0       33769
rcv     8.51    A       33769   0       2002
```

Summary (from receiver's log):

Amount of original data received (in bytes): 32767

Number of original data segments received: 697

Number of duplicate segments received: 1

COMP3331 Assignment : z5311696

Sequence of segments at <u>sender</u> (Q3 a.): pdrop = 0.1

First 20 segments:

```
snd     0.00    S       1000    0       0
rcv     0.00    SA      2000    0       1001
snd     0.00    A       1001    0       2001
snd     0.01    D       1001    50      2001
snd     0.01    D       1051    50      2001
drop    0.01    D       1101    50      2001
snd     0.01    D       1151    50      2001
snd     0.01    D       1201    50      2001
snd     0.01    D       1251    50      2001
snd     0.01    D       1301    50      2001
snd     0.01    D       1351    50      2001
snd     0.01    D       1401    50      2001
snd     0.01    D       1451    50      2001
rcv     0.01    A       2001    0       1051
snd     0.01    D       1501    50      2001
rcv     0.01    A       2001    0       1101
snd     0.01    D       1551    50      2001
rcv     0.01    A       2001    0       1101
rcv     0.01    A       2001    0       1101
rcv     0.01    A       2001    0       1101
```

Sequence where dropping occurred:

```
drop    2.07    D       87851   50      2001
drop    2.07    D       87901   50      2001
snd     2.07    D       87951   50      2001
snd     2.07    D       88001   50      2001
snd     2.07    D       88051   50      2001
snd     2.07    D       88101   50      2001
rcv     2.08    A       2001    0       87701
rcv     2.08    A       2001    0       87751
```

COMP3331 Assignment : z5311696

Last 20 segments:

```
drop    0.62    D       33451   50      2001
snd     0.62    D       33501   50      2001
snd     0.62    D       33551   50      2001
snd     0.62    D       33601   50      2001
snd     0.62    D       33651   50      2001
snd     0.62    D       33701   50      2001
snd     0.62    D       33751   17      2001
rcv     0.62    A       2001    0       33401
rcv     0.62    A       2001    0       33401
snd     0.62    D       33401   50      2001
rcv     0.62    A       2001    0       33401
rcv     0.62    A       2001    0       33401
rcv     0.62    A       2001    0       33401
snd     0.62    D       33401   50      2001
rcv     0.63    A       2001    0       33401
rcv     0.63    A       2001    0       33451
rcv     0.63    A       2001    0       33451
snd     0.64    D       33451   50      2001
rcv     0.64    A       2001    0       33768
snd     0.64    F       33768   0       2001
rcv     0.64    FA      2001    0       33769
snd     0.64    A       33769   0       2002
```

Summary (from sender's log):

Amount of original data: 32767

Number of data segments (excluding retransmissions): 656

Number of packets dropped: 66

Number of retransmitted packets: 108

Number of duplicate acknowledgements: 388

COMP3331 Assignment : z5311696

Sequence of segments at <u>receiver</u> (Q3 a.): pdrop = 0.3

First 20 segments:

```
rcv    4.71   S      1000   0      0
snd    4.71   SA     2000   0      1001
rcv    4.71   A      1001   0      2001
rcv    4.71   D      1001   50     2001
snd    4.71   A      2001   0      1051
rcv    4.71   D      1051   50     2001
snd    4.71   A      2001   0      1101
rcv    4.71   D      1151   50     2001
snd    4.71   A      2001   0      1101
rcv    4.71   D      1201   50     2001
snd    4.71   A      2001   0      1101
rcv    4.71   D      1251   50     2001
snd    4.71   A      2001   0      1101
rcv    4.71   D      1301   50     2001
snd    4.71   A      2001   0      1101
rcv    4.71   D      1401   50     2001
snd    4.71   A      2001   0      1101
rcv    4.71   D      1451   50     2001
snd    4.71   A      2001   0      1101
rcv    4.72   D      1101   50     2001
```

COMP3331 Assignment : z5311696

Last 20 segments:

```
rcv    10.34   D    33451   50    2001
snd    10.34   A    2001    0     33101
rcv    10.34   D    33501   50    2001
snd    10.34   A    2001    0     33101
rcv    10.35   D    33101   50    2001
snd    10.35   A    2001    0     33351
rcv    10.35   D    33601   50    2001
snd    10.35   A    2001    0     33351
rcv    10.35   D    33701   50    2001
snd    10.35   A    2001    0     33351
rcv    10.35   D    33751   17    2001
snd    10.35   A    2001    0     33351
rcv    10.35   D    33351   50    2001
snd    10.35   A    2001    0     33551
rcv    10.36   D    33551   50    2001
snd    10.36   A    2001    0     33651
rcv    10.37   D    33651   50    2001
snd    10.37   A    2001    0     33768
rcv    10.37   F    33768   0     2001
snd    10.37   FA   2001    0     33769
rcv    10.37   A    33769   0     2002
```

Summary (from receiver's log):

Amount of original data received (in bytes): 32767

Number of original data segments received: 667

Number of duplicate segments received: 2

COMP3331 Assignment : z5311696

Sequence of segments at <u>sender</u> (Q3 a.): pdrop = 0.3

First 20 segments:

```
snd     0.00    S       1000    0       0
rcv     0.00    SA      2000    0       1001
snd     0.00    A       1001    0       2001
snd     0.00    D       1001    50      2001
snd     0.00    D       1051    50      2001
drop    0.01    D       1101    50      2001
snd     0.01    D       1151    50      2001
snd     0.01    D       1201    50      2001
snd     0.01    D       1251    50      2001
snd     0.01    D       1301    50      2001
drop    0.01    D       1351    50      2001
snd     0.01    D       1401    50      2001
snd     0.01    D       1451    50      2001
drop    0.02    D       1001    50      2001
rcv     0.02    A       2001    0       1051
drop    0.02    D       1501    50      2001
rcv     0.02    A       2001    0       1101
drop    0.02    D       1551    50      2001
rcv     0.02    A       2001    0       1101
rcv     0.02    A       2001    0       1101
```

Sequence where dropping occurred:

```
drop    5.31    D       32351   50      2001
drop    5.31    D       32401   50      2001
snd     5.31    D       32451   50      2001
snd     5.31    D       32501   50      2001
rcv     5.31    A       2001    0       32051
rcv     5.31    A       2001    0       32051
```

COMP3331 Assignment : z5311696

Last 20 segments:

```
rcv     5.64    A       2001    0       33101
rcv     5.64    A       2001    0       33101
snd     5.64    D       33101   50      2001
rcv     5.64    A       2001    0       33351
snd     5.64    D       33601   50      2001
drop    5.64    D       33651   50      2001
snd     5.64    D       33701   50      2001
snd     5.64    D       33751   17      2001
rcv     5.64    A       2001    0       33351
rcv     5.64    A       2001    0       33351
rcv     5.64    A       2001    0       33351
snd     5.64    D       33351   50      2001
rcv     5.64    A       2001    0       33551
snd     5.65    D       33551   50      2001
rcv     5.66    A       2001    0       33651
snd     5.67    D       33651   50      2001
rcv     5.67    A       2001    0       33768
snd     5.67    F       33768   0       2001
rcv     5.67    FA      2001    0       33769
snd     5.67    A       33769   0       2002
```

Summary (from sender's log):

Amount of original data: 32767

Number of data segments (excluding retransmissions): 656

Number of packets dropped: 285

Number of retransmitted packets: 298

Number of duplicate acknowledgements: 463