PTP REPORT: Z5311917

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.

The implementation of the PTP protocol was relatively straightforward and logical. I followed the assignments tips on getting started and worked through the requirements gradually. I began with the simple stop and wait protocol with no PL module. Here, the receiver's job was only to receive incoming packets and the sender sent the packets one by one. I researched ways to implement the PTP header and decided to use the structs module as it allowed me to easily package the header fields neatly (for the most part). Thus, in the first part, the following features of PTP were implemented.

Features of my PTP up until this point

1. Connection setup
2. Connection teardown
3. MSS
4. Stop and wait transfer of file
    a. File is transferred packet by packet in order
5. Creation of transferred file

On top of this, the log was created at this stage to help with debugging and future development.

In the second stage of implementation, I implemented the PL module and adjusted my existing stop and wait to ensure it reliably sent the file still. Here, I hardcoded this so that if the PL module returned false, the packet wouldn't be sent. A debugging print statement would say "failed to send" and then the packet would be resent immediately after. At this stage, there is no timeout or 'real' packet loss management.

Features of my PTP up until this point

1. Connection setup
2. Connection teardown
3. MSS
4. MWS
5. PL module
6. Stop and wait transfer of file
    a. File is transferred packet by packet in order
    b. Hardcoded to adjust for packet loss

Following on from this, the next step was to implement pipelining and speed everything up. To do this, threading was needed. Before coding, I spent days researching python threads to understand how they work. Once I had a sufficient understanding of how they worked, I set off to work. Up until this point, the entire transfer of the file existed within one while loop so it was in charge of both sending and receiving packets. I removed all the elements of the loop that had to do with receiving ACKS and moved it into a receive thread which would run concurrently with the main sending

thread. I also implemented the MWS feature. Here, no changes were required for the receiver as it was still receiving packets in order (Assumption that packets guaranteed to deliver). Running the code and viewing the log, there was a small issue where the Ack from receiver would sometimes arrive before the initial few window packets were all sent out. Presumably this is because they are running on the same machine and therefore the delay is very small.

Features of my PTP up until this point

1. Connection setup
2. Connection teardown
3. MSS
4. MWS
5. Stop and wait transfer of file
    a. File is transferred packet by packet in order

   Pipelining
    b. File is transferred packet by packet in order
6. Creation of transferred file

Next step after was to get the timeout feature working. I spent probably the most time stuck in the part as I couldn't figure out how to get Timer threads to restart in the middle of them running. Eventually I figured it out though. I used threading.Timer as the timer. This thread timer would run in the background and call the timeout function when it ran out of time. The timeout function would then call a new threading.timer at the end of its execution, restarting the timer. After this, I also had the refactor my receiver to account for the receiving of out of order packets. I did this by having a buffer. My PTP was finally coming along and can now deal with packet loss. A challenge I faced in this section was within my receiver. When a new packet was received, I had to check the existing buffer to determine what the appropriate new ack was. For example, if packets 1,2,3 were dropped and packet 4 sent, when 1,2,3 do get sent, the next ACK should be for 5.

Features of my PTP up until this point

1. Connection setup
2. Connection teardown
3. MSS
4. MWS
5. Stop and wait transfer of file
    a. File is transferred packet by packet in order

   Pipelining
    b. Timer and timeout
6. Receiver buffering out of order segments

PTP is working fine for the most part now but it's quite slow, if a packet is ever lost, I have to wait for the entire duration of the timeout before continuing, reducing its speed. The next step was to implement the fast retransmit. This was relatively simple to do, I put a duplicate ACK counter inside the receiver thread and if it ever reached 3, it would retransmit the oldest packet. Thus in the end, I was able to implement all the required features of PTP and get it working.

Features of PTP final(taken from assignment spec)

Sender

1. Connection setup

2. Data Transmission (repeat until end of file)

    a. Read file

    b. Create PTP segment

    c. Start Timer if required (retransmit oldest unacknowledged segment on expiry)

    d. Send PTP segment to PL module

    e. If PTP segment is not dropped, transmit to Receiver

    f. Process ACK if received including fast retransmit

3. Connection teardown

Receiver

1. Connection setup

2. Data Transmission (repeat until end of file)

    a. Receive PTP segment

    b. Send ACK segment

    c. Buffer data or write data into file

3. Connection teardown

ISSUES

There is a major issue with the timer and timeout when selecting certain values for timeout result in the programming locking up and the following error. My guess it has to do with the receiver being unable to cancel the timer before it times out. I have not been able to resolve this issue. For large size files, program works in stop and wait but sometimes not for pipelining

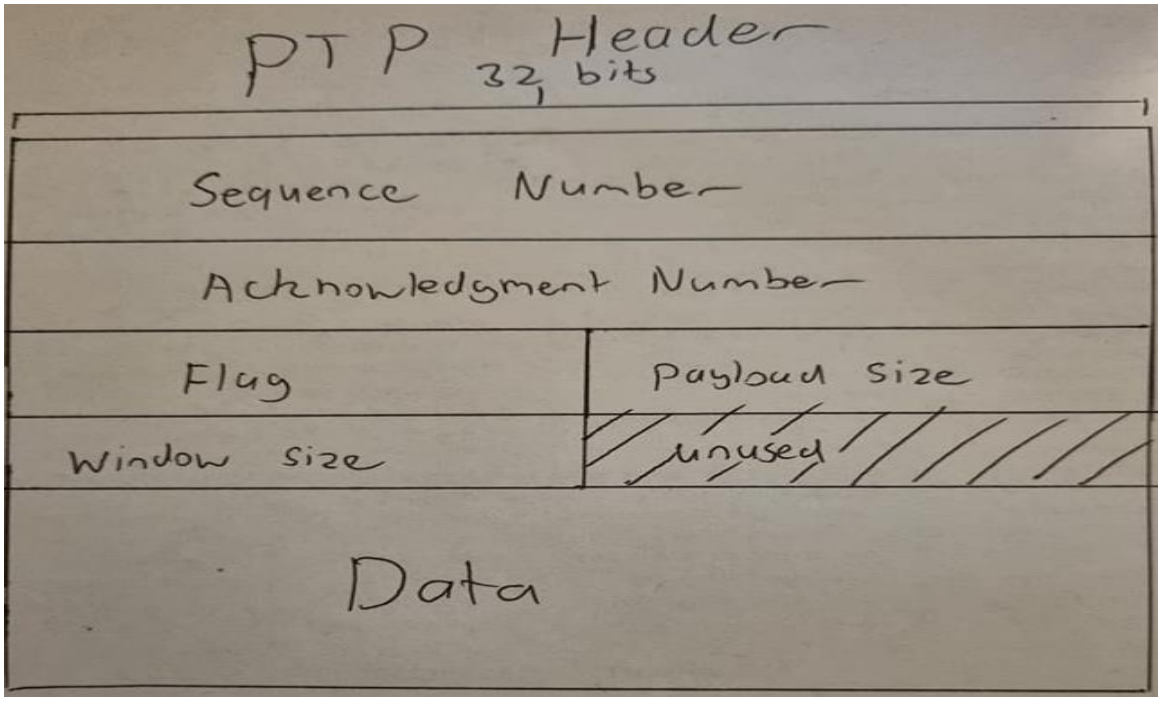
```
z5311917@vx5:/tmp_amd/ravel/export/ravel/1/z5311917/3331/ass$ python3 sender.py 127.0.0.1
 5001 32KB.txt 500 50 50 0.1 300
Exception in thread Thread-2:
Traceback (most recent call last):
  File "/usr/lib/python3.7/threading.py", line 917, in _bootstrap_inner
    self.run()
  File "/usr/lib/python3.7/threading.py", line 865, in run
    self._target(*self._args, **self._kwargs)
  File "sender.py", line 170, in Recevier
    tmr.start()
  File "/usr/lib/python3.7/threading.py", line 847, in start
    _start_new_thread(self._bootstrap, ())
RuntimeError: can't start new thread

^CTraceback (most recent call last):
  File "sender.py", line 220, in <module>
    time.sleep(0.01)
KeyboardInterrupt
```

Sometimes ACKS come back before the window has been sent out. For example, here the MWS is equivalent to 4 packets. 3 Packets are sent out but before the fourth one can be sent out, an ack for the first one has already arrived. I don't know if this is a major issue or if it's just part of having sender and receiving both running on the local machine.

snd	52.2	D	201	10	1
snd	52.3	D	211	10	1
snd	52.36	D	221	10	1
rcv	52.44	A	1	0	211
snd	52.56	D	231	10	1

One other thing is that my receiver buffers the entire file before writing it out in the end. I liken this to transferring a file on my computer, if the connection is abruptly cancelled the file isn't left there half transferred. The spec does say that packets should be buffered or written though so this may be an issue. The end result is the same though.

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)



Sequence Number: Same as with TCP (32bits)

Acknowledgment Number: Same as with TCP (32bits)

Flag: 16 bit number corresponding to flag
0 = Noflag, data
1 = FIN
2 = SYN
4 = ACK
6 = SYNACK
5 = FINACK

Payload size: Size in bytes of data portion of packet, main reason for including this was to make it easier to generate log report. So technically not needed in transmission.

Window Size: This is 'optional' and only added to the header in the first SYN Packet. Every header after does not use it. Reason for adding it was that I realised the receiver needed to know the

senders MWS/MSS to check what ACK to reply with. For example, if first packet is dropped but next 8 packets are sent, hypothetically if there was no fast transmission, when the dropped packet is resent, the receiver loops through the buffer MWS/MSS times to determine the ACK number for the 10^{th} packet.

Thus size of header is 12/14 bytes (depending on if window size is included).

Experiment

a.

We want a timeout that allows for fastest transfer time of file as this demonstrates faster throughput which is what we want.

I started at 100ms and found that as the timeout was lower, the time taken to transmit the file was lower but also the number of retransmissions in the log increased. When I tried 20ms as a timeout value, my code locked up and the timer kept retransmitting the file and. This may be due to the fact that I selected a value that is lower than the RTT. Thus, I have selected a timeout value of 21ms as that is the lowest value that transmits the file in the fastest time while not crashing the program.

The file I used to determine the initial value was 16kb. When testing Test1.txt on 21ms timeout, the program locks up and crashes again. I had to adjust the value to 70ms to avoid this issue.

Upping the pdrop to 0.3, a 70ms timeout cause the program to lockup again… I adjusted it to 80ms to avoid this issue. I redid the above transfer as well from 70 to 80ms to keep the experiment consistent. Appendix A is an extract of the last few ptp packets received from pdrop = 0.1 and Appendix B is an extract of the last few ptp packets received from pdrop = 0.3

In Appendix A, it can be seen that packets with seq 101, 501 and 1351 are dropped, hence they are retransmitted again later.

In appendix B, it can be seen that packets with seq 351, 501, 651, 851, 951, 1201, 1451, 1601, 1751 are dropped and retransmitted later, this lines up with pdrop as there around 3x more drops occurring.

P.S (I realise I misspelled rcv as recv, changed it just then 😊).

b.

Using timeout value of 80.

	Tcurrent	Tcurrent * 4	Tcurrent/4
PTP packets			
Time taken			

Unfortunately, I was unable to get the program to run with this large file size due to issues with the timer. The issue arises at around file sizes of 100kbs. If you find the reason while marking, I'm curious to know.

Thanks

Appendix A

1	rcv	275.21	S	0	0	0
2	recv	275.37	A	1	0	1
3	recv	275.68	D	1	50	1
4	recv	275.96	D	51	50	1
5	recv	275.98	D	151	50	1
6	recv	276.0	D	201	50	1
7	recv	276.02	D	251	50	1
8	recv	276.04	D	301	50	1
9	recv	276.05	D	351	50	1
10	recv	276.07	D	401	50	1
11	recv	276.09	D	451	50	1
12	recv	277.06	D	101	50	1
13	recv	277.23	D	101	50	1
14	recv	286.23	D	551	50	1
15	recv	286.27	D	601	50	1
16	recv	286.3	D	651	50	1
17	recv	286.32	D	701	50	1
18	recv	286.35	D	751	50	1
19	recv	286.37	D	801	50	1
20	recv	286.39	D	851	50	1
21	recv	286.42	D	901	50	1
22	recv	286.44	D	951	50	1
23	recv	286.58	D	501	50	1
24	recv	286.79	D	501	50	1
25	recv	296.62	D	1001	50	1
26	recv	296.66	D	1051	50	1
27	recv	296.69	D	1101	50	1
28	recv	296.72	D	1151	50	1
29	recv	296.74	D	1201	50	1
30	recv	296.76	D	1251	50	1
31	recv	296.79	D	1301	50	1
32	recv	296.81	D	1401	50	1
33	recv	296.85	D	1451	50	1
34	recv	306.91	D	1501	50	1
35	recv	306.98	D	1551	50	1
36	recv	307.02	D	1601	50	1
37	recv	307.06	D	1651	50	1
38	recv	307.1	D	1701	50	1
39	recv	307.14	D	1751	50	1
40	recv	307.18	D	1801	50	1
41	recv	307.36	D	1351	50	1
42	recv	307.64	D	1351	50	1
43	recv	307.98	D	1351	50	1
44	recv	317.49	D	1851	50	1
45	recv	317.58	D	1901	50	1
46	recv	317.63	D	1951	50	1
47	recv	317.68	D	2001	50	1
48	recv	317.72	D	2051	50	1
49	recv	317.77	D	2101	50	1

Appendix B

1	rcv	286.77	S	0	0	0
2	recv	286.85	A	1	0	1
3	recv	287.06	D	1	50	1
4	recv	287.27	D	51	50	1
5	recv	287.3	D	151	50	1
6	recv	287.31	D	201	50	1
7	recv	287.33	D	251	50	1
8	recv	287.34	D	301	50	1
9	recv	287.36	D	401	50	1
10	recv	287.37	D	451	50	1
11	recv	297.48	D	551	50	1
12	recv	368.25	D	101	50	1
13	recv	368.65	D	351	50	1
14	recv	368.79	D	351	50	1
15	recv	370.56	D	601	50	1
16	recv	370.63	D	701	50	1
17	recv	370.68	D	751	50	1
18	recv	370.72	D	801	50	1
19	recv	370.76	D	901	50	1
20	recv	370.97	D	501	50	1
21	recv	371.26	D	501	50	1
22	recv	380.81	D	1001	50	1
23	recv	380.88	D	1051	50	1
24	recv	380.92	D	1101	50	1
25	recv	381.12	D	651	50	1
26	recv	391.21	D	1151	50	1
27	recv	391.29	D	1251	50	1
28	recv	391.34	D	1301	50	1
29	recv	391.61	D	851	50	1
30	recv	401.52	D	1351	50	1
31	recv	401.59	D	1401	50	1
32	recv	451.28	D	951	50	1
33	recv	451.87	D	1201	50	1
34	recv	451.95	D	1201	50	1
35	recv	453.12	D	1501	50	1
36	recv	453.19	D	1551	50	1
37	recv	453.24	D	1651	50	1
38	recv	453.29	D	1701	50	1
39	recv	453.33	D	1801	50	1
40	recv	453.38	D	1851	50	1
41	recv	453.42	D	1901	50	1
42	recv	453.64	D	1451	50	1
43	recv	453.95	D	1451	50	1
44	recv	463.37	D	1951	50	1
45	recv	463.44	D	2051	50	1
46	recv	530.69	D	1601	50	1
47	recv	532.4	D	1751	50	1
48	recv	532.64	D	2001	50	1
49	recv	534.06	D	2101	50	1