COMP9331 ASSIGNMENT

Meng Zhang z5187943

Content

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

Class Sender:

I defined a class called 'sender' which can store all the parameter of the transferring process. The attributes include: all the input attributes (host_ip, port, filename, mss, etc.), some process parameters (timer, LastByteAcked, LastByteSent, etc.), and some statistic parameters (plded num, reordered, corrupted, etc.).

Switch_events: A main function which include all the possible events of a STP Sender.
The conditions include: Waiting, Handshake, Transferring, Wave_goodbye.

Handshake: When self.event is 'Handshake', the Sender sends the 1st syn Segment to the Receiver, and wait for the synack, at last reply the ack Segment back. After that, the connection can be seen established.

Wave_goodbye: After all the 3 threading in transfer done, the self.event would be set to 'Wave_goodbye', and the Sender send the fin Segment to the Receiver, and wait for the ack of this Segment. When the Sender receives the fin signal from the Receiver later, it will also reply the corresponding ack. Then the whole processing is completed.

Transfer

```
/* Assume sender is not constrained by TCP flow or congestion control, that data from above is less
than MSS in size, and that data transfer is in one direction only. */

NextSeqNum=InitialSeqNumber
SendBase=InitialSeqNumber

loop (forever) {
    switch(event)

        event: data received from application above
            create TCP segment with sequence number NextSeqNum
            if (timer currently not running)
                start timer
            pass segment to IP
            NextSeqNum=NextSeqNum+length(data)
            break;

        event: timer timeout
            retransmit not-yet-acknowledged segment with
                smallest sequence number
            start timer
            break;

        event: ACK received, with ACK field value of y
            if (y > SendBase) {
                SendBase=y
                if (there are currently any not-yet-acknowledged segments)
                    start timer
                }
            break;

    } /* end of loop forever */
```

Using the simplified TCP sender structure learned from the reference book, I write the main function of the whole data transferring process, including 3 threading keep doing at mean time independently: sending_thread, receiving_thread, timeout_thread.

Sending_thread: The sending thread is responsible for controlling sending all the segments to the Receiver. The only limit for the sending is to limit the window size under the mws. When the window base moved by the receiving thread, the sending thread would send as many as packets until the window is full. The sending thread also have a PLD module to implement the network fluctuation, and the break condition is: if only all the content of the file has been transferred, the loop break.

Receiving_thread: Then receiving thread is used for managing all the acks received from the Receiver. This thread also controls the fast retransmit when it counts the acks of one unique packet are over 3 , and when the ack_num is larger than the last acked num, it can update the window base, if this ack is because of the receiving of a non-retransmit packet, it will update the sample RTT of the connection, (which used to calculate the interval of the timer, a class called Timer used in the

	timeout thread). When the window is moved, it restarts the timer. The break point is that the Sender confirms that the ack of all the segments has been received.
Timeout_thread	I defined a new class called Timer to implement this part of function. Timer: Attributes: timer, started, estimate_rtt, dev_rtt, gamma, interval Functions: cancel() start() restart() expire() update interval() Every packet (except the retransmit packets) will get a sent time when it is ready to be sent, and when its ack received it can get the sample RTT of the connection which can be used to refresh the timer's interval. When the timer is expired, the Sender need to transfer the segment whose seqnum is the next one to the last acked segment. After the receiving thread stopped, the event would be change to 'Wave goodbye', so the timeout thread will closed, too.
PLD module	PLD module is a module which achieve all the fluctuation during the transferring. Drop means this packet won't be sent, duplicate means the packet will be sent twice directly, corrupt means the payload has been changed during the transferring, and reorder means the packet would enter into an order list, when the number of packet sent afterwards over the max order, it will be sent at once. For the delay function we define a new timer which can achieve the sending after the delay period. If a packet avoids all the process above it will be sent normally.
Other functions	Udp start: establish a socket Get types: get the type of a packet Slice: slice a file into packets in a length of mss
Class Receiver: I defined a class called 'receiver' which can store all the parameter of the transferring process. The attributes include: all the input attributes (port, filename), some process parameters (nextack, nextseq etc.), and some statistic parameters (seg_rev, dup_ack etc.).	
Switch_events	A main function which include all the possible events of a STP Receiver. The conditions include: Waiting, Handshake, Receiving, Wave_goodbye.
Handshake	When the Receiver received the syn signal from the Sender it will send a synack back to the Sender, when it receives the ack of the synack from the Sender, the connection is established.
Wave_goodbye	When the receiver received the fin signal from the sender it will send ack, then send an fin to sender, when it receives the right ack of the fin, the connection closed.
Receiving	When the Receiver in the receiving event, it will check if the md5 of the payload is same as the real checksum to find the corruption packets, detect the duplicate packets, and record all the right packets in the window.
Other functions	Udp start: establish a socket Get types: get the type of a packet Write copy: write all the data received in right order to a file
Other Class:	
Timer	Timer to control the timeout. Has been mentioned in Sender. Timeout thread
Segment	The format of the STP. Will be mentioned in detail in question 2.

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

Header including

Seq_num	Int	Sequence number of the segment
Ack_num	Int	Acknowledge number of the segment
Checksum	String (Md5)	Encrypted information of the payload
Syn	True / False	If syn
Ack	True / False	If ack
Fin	True / False	If fin
Sent_time	Timestamp	The sent time of the segment
If_retransmit	True / False	If the segment is retransmitted
Payload	(Length<Mss)	The content of the segment

3. Discuss any design trade-offs considered and made. Describe possible improvements and extensions to your program and indicate how you could realise them.

Answer:

During the implement of the Sender, I 1st tried in the all while loop approaches, but I found that the while loops cannot make the Sender to receive/send/counting time at the same time, so after a week from I start the project, I change the structure of my code into multi-threads.

Also, in the implement of the Sender, I have been confused in which are the proper packets used to refresh the sample RTT of the timer. After plenty of discussions, I change the position of the refresh part to the position just after the lastbyteacked updated.

The corruption also confused me for a long time, I 1st just add one b'1' into the right payload, then I found that this would cause the window out of mws, so at last, I convert the payload to modify it then convert back to solve the problem.

The wrong log confuses me for a while, too. Because sometimes the recv of one packet is beyond the snd signal. That is really strange. At last I found that the real sendto should be placed after the writing log parts.

The extension of my programming is that I can add a new time at the end of wave goodbye of the Sender, even if the sender cannot receive the acks or the fins, when time out, it should also close the connection.

4. Indicate any segments of code that you have borrowed from the Web or other books.

Computer Networking - A Top-Down Approach Featuring the Internet, J. Kurose and K. Ross, Pearson, 7th Edition, 2017 (Sixth edition will suffice for most parts).

5. Answer the following questions: (include any output as an appendix to the main report.pdf, appendix is not included in the 5-page limit)

Question a. Discuss the resulting packet sequences of both experiments indicating where dropping occurred.

Answer:

Under the 1st condition, the number of dropped packets is 4, the number of timeout retransmits are 2, and the number of fast retransmits are 2.

Under the 1st condition, the number of dropped packets is 24, the number of timeout retransmits are 23, and the number of fast retransmits are 2.

From the statistics above, we can prove that the probability of drop can largely influence the timeout retransmit, but not fast retransmits.

Pdrop = 0.1		
Drop seq_num	Drop/rxt time	Handled by function
201	0.1285/0.317	Fast retransmit after 3 dup acks
2001	1.5681/2.6330	Fast retransmit after 3 dup acks
2701	2.8031/4.8342	Time out retransmit
2801	2.9092/6.7200	Time out retransmit
Pdrop = 0.3		
1	0.0436/ 3.1011	Time out retransmit
401	0.1483/4.6585	Time out retransmit
501	0.1612/6.1813	Time out retransmit
…		Time out retransmit
1201	9.3082/15.4438	Fast retransmit after 3 dup acks
1801	15.4835/21.5809	Fast retransmit after 3 dup acks
…		Time out retransmit

Question b. Show a table that indicates how many STP packets were transmitted in total and how long the overall transfer took. Discuss the results

Answer:

Gamma	Information	Time	Num of packets
2	snd 5934.1725 A 308205 0 2 ========== Size of the file (in Bytes) = 308203 Segments transmitted (including drop & RXT) = 12515 Number of Segments handled by PLD = 12511	5934.17	12515
4	snd 8852.4171 A 308205 0 2 ========== Size of the file (in Bytes) = 308203 Segments transmitted (including drop & RXT) = 12459 Number of Segments handled by PLD = 12455	8852.42	12459
6	snd 13274.1963 A 308205 0 2 ========== Size of the file (in Bytes) = 308203 Segments transmitted (including drop & RXT) = 12451 Number of Segments handled by PLD = 12447	13274.20	12451

Result Discussion:

The gamma's size influences the interval's update with the function as follows:

self.interval = self.estimate_rtt + self.gamma * self.dev_rtt

In the testing cases, the possibility of drop is large and have the delayed conditions too. The only condition where the interval would be updated is the window base changed by normal packet ack receiving, because of the high drop rate, the interval won't be refreshed very frequently. With the different gamma, the ranks of the intervals of different cases should be 6>4>2, which affect the length of transferring time. Then because there are delay cases, if the interval length is smaller, the delayed case would more likely not be waited to be received which can cause the fast retransmission. So the larger the gamma is, the less the packets transmitted.

Question c. Has the file been successfully transferred? How long the overall transfer took? For this experiment, which of the factor (out of pDrop, pDuplicate, pCorrupt and pOrder) is the most critical contributing most in the overall transfer time? How have you determined this?

Answer:

Successfully transferred	Yes.
Time taken	About 2671s

The most critical factor	Drop and Corrupt. To some extent, the two kinds of things can have the same influence on the length of total transferring period. The dropped packets won't be acked, and this can cause the timeout retransmit and fast retransmit. Same as dropped condition, the corrupted packets won't be confirmed too, because of the wrong checksum, and this can also cause the timeout retransmit and fast retransmit. A timeout retransmit means that the transferring time add a total length of the timer's timeout period which may cause a large increase. The other two kinds of network fluctuation, duplicated and reordered won't trigger the timeout retransmit mostly. They can only cause the fast retransmit because of the multi-acked from the receiver in most situation, which won't affect the total time too much, apparently less than the first two conditions. If the max order of reorder function is too large, reordered may cause some timeout retransmit, too. But the possibility of this condition can be really low which may be ignored.

Appendix

Question a. This is the Sender_log.txt which stands for the sequence of transferring.
pDrop=0.1:

rcv	0.0020	S	0	0	0
snd	0.0030	A	0	0	1
rcv	0.0099	A	1	0	1
rcv	0.0674	D	1	100	1
snd	0.0748	A	1	0	101
rcv	0.0868	D	101	100	1
snd	0.0882	A	1	0	201
rcv	0.2539	D	301	100	1
snd/da	0.2554	A	1	0	201
rcv	0.2713	D	401	100	1
snd/da	0.2728	A	1	0	201
rcv	0.2886	D	501	100	1
snd/da	0.2926	A	1	0	201
rcv	0.3298	D	201	100	1
snd	0.3313	A	1	0	601
rcv	0.3417	D	601	100	1
snd	0.3432	A	1	0	701
rcv	0.4528	D	701	100	1
snd	0.4543	A	1	0	801
rcv	0.5094	D	801	100	1
snd	0.5113	A	1	0	901
rcv	0.6076	D	901	100	1
snd	0.6090	A	1	0	1001
rcv	0.6968	D	1001	100	1
snd	0.7013	A	1	0	1101
rcv	0.7494	D	1101	100	1
snd	0.7524	A	1	0	1201
rcv	0.8432	D	1201	100	1
snd	0.8446	A	1	0	1301
rcv	0.9245	D	1301	100	1
snd	0.9260	A	1	0	1401
rcv	0.9756	D	1401	100	1
snd	0.9776	A	1	0	1501
rcv	1.0465	D	1501	100	1
snd	1.0480	A	1	0	1601
rcv	1.1670	D	1601	100	1
snd	1.1705	A	1	0	1701
rcv	1.2300	D	1701	100	1
snd	1.2320	A	1	0	1801
rcv	1.2876	D	1801	100	1
snd	1.2891	A	1	0	1901
rcv	1.3580	D	1901	100	1
snd	1.3595	A	1	0	2001
snd	1.3595	A	1	0	2001
rcv	1.5306	D	2101	100	1
snd/da	1.5321	A	1	0	2001
rcv	1.5976	D	2201	100	1
snd/da	1.5996	A	1	0	2001
rcv	1.6759	D	2301	100	1
snd/da	1.6774	A	1	0	2001
rcv	1.7613	D	2401	100	1
snd/da	1.7627	A	1	0	2001
rcv	1.8332	D	2001	100	1
snd	1.8352	A	1	0	2501
rcv	1.8758	D	2501	100	1
snd	1.8773	A	1	0	2601
rcv	1.9770	D	2601	100	1
snd	1.9805	A	1	0	2701
rcv	2.1233	D	2901	100	1
snd/da	2.1268	A	1	0	2701
rcv	2.1407	D	3001	28	1
snd/da	2.1422	A	1	0	2701
rcv	2.8143	D	2701	100	1
snd	2.8172	A	1	0	2801
rcv	3.5801	D	2801	100	1
snd	3.5821	A	1	0	3029
snd	3.5905	A	1	0	3030
rcv	3.5905	F	1	0	3030
rcv	3.5925	A	3030	0	2

==========

Amount of Data Received (bytes) = 3028
Total segments received = 35
Data segments received = 31
Data Segments with bit errors = 0
Duplicate data segments received = 0
Duplicate Ack sent = 9

==========

pDrop = 0.3:

rcv	0.0025	S	0	0	0	snd	15.4547	A	1	0	1501
snd	0.0045	A	0	0	1	rcv	15.4716	D	1701	100	1
rcv	0.0109	A	1	0	1	snd/da	15.4726	A	1	0	1501
rcv	0.0928	D	101	100	1	rcv	21.5214	D	1501	100	1
snd/da	0.0942	A	1	0	1	snd	21.5224	A	1	0	1801
rcv	0.1156	D	201	100	1	rcv	21.5393	D	2001	100	1
snd/da	0.1215	A	1	0	1	snd/da	21.5403	A	1	0	1801
rcv	0.1374	D	301	100	1	rcv	21.5571	D	2101	100	1
snd/da	0.1394	A	1	0	1	snd/da	21.5576	A	1	0	1801
rcv	3.1393	D	1	100	1	rcv	21.5740	D	2201	100	1
snd	3.1432	A	1	0	401	snd/da	21.5750	A	1	0	1801
rcv	3.2042	D	801	100	1	rcv	21.5968	D	1801	100	1
snd/da	3.2062	A	1	0	401	snd	21.5978	A	1	0	1901
rcv	4.6694	D	401	100	1	rcv	23.1371	D	1901	100	1
snd	4.6704	A	1	0	501	snd	23.1381	A	1	0	2301
rcv	6.2656	D	501	100	1	rcv	23.2174	D	2701	100	1
snd	6.2681	A	1	0	601	snd/da	23.2179	A	1	0	2301
rcv	6.3851	D	1001	100	1	rcv	24.6618	D	2301	100	1
snd/da	6.3866	A	1	0	601	snd	24.6638	A	1	0	2401
rcv	7.7790	D	601	100	1	rcv	26.1795	D	2401	100	1
snd	7.7800	A	1	0	701	snd	26.1825	A	1	0	2501
rcv	7.7964	D	1101	100	1	rcv	27.6968	D	2501	100	1
snd/da	7.7974	A	1	0	701	snd	27.6988	A	1	0	2601
rcv	9.3007	D	701	100	1	rcv	27.7147	D	3001	28	1
snd	9.3017	A	1	0	901	snd/da	27.7157	A	1	0	2601
rcv	9.3250	D	1301	100	1	rcv	30.7041	D	2601	100	1
snd/da	9.3275	A	1	0	901	snd	30.7055	A	1	0	2801
rcv	13.9373	D	901	100	1	rcv	33.7138	D	2801	100	1
snd	13.9383	A	1	0	1201	snd	33.7158	A	1	0	2901
rcv	13.9671	D	1401	100	1	rcv	35.2219	D	2901	100	1
snd/da	13.9686	A	1	0	1201	snd	35.2238	A	1	0	3029
rcv	13.9954	D	1601	100	1	snd	35.2308	A	1	0	3030
snd/da	13.9964	A	1	0	1201	rcv	35.2308	F	1	0	3030
rcv	15.4538	D	1201	100	1	rcv	35.2342	A	3030	0	2
snd	15.4547	A	1	0	1501	==					
rcv	15.4716	D	1701	100	1	Amount of Data Received (bytes) = 3028					
snd/da	15.4726	A	1	0	1501	Total segments received = 35					
rcv	21.5214	D	1501	100	1	Data segments received = 31					
snd	21.5224	A	1	0	1801	Data Segments with bit errors = 0					
rcv	21.5393	D	2001	100	1	Duplicate data segments received = 0					
snd/da	21.5403	A	1	0	1801	Duplicate Ack sent = 16					
rcv	21.5571	D	2101	100	1	==					
snd/da	21.5576	A	1	0	1801						

Question c.

Sender_log.txt

The handshake and 1st 20 entries

snd	0.0005	S	0	0	0
rcv	0.0079	AS	0	0	1
snd	0.0084	A	1	0	1
snd/corr	0.3254	D	1	50	1
snd	0.3695	D	51	50	1
rcv	0.3760	A	1	0	1
snd	0.3913	D	101	50	1
rcv/da	0.3993	A	1	0	1
snd	0.4057	D	151	50	1
rcv/da	0.4122	A	1	0	1
snd	0.4246	D	201	50	1
rcv/da	0.4305	A	1	0	1
snd/rxt	0.4310	D	1	50	1
snd/dup	0.4375	D	1	50	1
snd/dup	0.4439	D	1	50	1
rcv	0.5114	A	1	0	251
snd	0.5114	D	251	50	1
rcv/da	0.5198	A	1	0	251
snd/corr	0.5258	D	301	50	1
rcv	0.5258	A	1	0	301
snd	0.6522	D	351	50	1

The last 20 entries and summary table

rcv/da	2669.3219	A	1	0	1605101
rcv/da	2669.3288	A	1	0	1605101
snd	2669.3407	D	1605501	50	1
rcv/da	2669.3472	A	1	0	1605101
snd/rxt	2669.3472	D	1605101	50	1
snd	2669.3824	D	1605101	50	1
rcv	2669.4251	A	1	0	1605151
snd	2669.4275	D	1605551	35	1
rcv/da	2669.4518	A	1	0	1605151
snd/rxt	2670.3407	D	1605151	50	1
snd/rord	2670.3407	D	651	50	1
snd	2670.3412	D	1605151	50	1
rcv/da	2670.3471	A	1	0	1605151
rcv	2670.3481	A	1	0	1605251
snd/rxt	2671.2617	D	1605251	50	1
snd	2671.2622	D	1605251	50	1
rcv	2671.2687	A	1	0	1605586
snd	2671.2746	F	1605586	0	1
rcv	2671.2771	A	1	0	1605587
rcv	2671.2771	F	1	0	1605587
snd	2671.2776	A	1605587	0	2

rcv/da	0.6582	A	1	0	301
snd	0.6765	D	401	50	1
rcv/da	0.6884	A	1	0	301
snd	0.6944	D	451	50	1
rcv/da	0.7152	A	1	0	301
snd/rxt	0.7152	D	301	50	1
snd	0.7157	D	501	50	1
snd/corr	0.7296	D	301	50	1
rcv/da	0.7693	A	1	0	301
snd	0.7753	D	551	50	1
rcv/da	0.7837	A	1	0	301
snd	0.8065	D	601	50	1

```
================================================
Size of the file (in Bytes) = 1605585
Segments transmitted (including drop & RXT) =  38371
Number of Segments handled by PLD =  38367
Number of Segments Dropped = 4191
Number of Segments Corrupted = 3366
Number of Segments Re-ordered = 1
Number of Segments Duplicated = 3732
Number of Segments Delayed = 0
Number of Retransmissions due to timeout = 953
Number of Fast Retransmissions = 8251
Number of Duplicate Acknowledgements received = 33200
================================================
```

Receiver_log.txt

The handshake and 1st 20 entries

rcv	0.0010	S	0	0	0
snd	0.0020	A	0	0	1
rcv	0.0079	A	1	0	1
rcv/corr	0.3566	D	1	50	1
rcv	0.3680	D	51	50	1
snd/da	0.3700	A	1	0	1
rcv	0.3913	D	101	50	1
snd/da	0.3938	A	1	0	1
rcv	0.4042	D	151	50	1
snd/da	0.4057	A	1	0	1
rcv	0.4231	D	201	50	1
snd/da	0.4246	A	1	0	1
rcv	0.4365	D	1	50	1
snd	0.4380	A	1	0	251
rcv/da	0.4429	D	1	50	1
snd/da	0.4459	A	1	0	251
rcv	0.5119	D	251	50	1
snd	0.5178	A	1	0	301
rcv/corr	0.5248	D	301	50	1
rcv	0.6512	D	351	50	1
snd/da	0.6537	A	1	0	301

The last 20 entries and summary table

rcv	2669.2391	D	1605001	50	1
snd	2669.2395	A	1	0	1605101
rcv/da	2669.2445	D	651	50	1
snd/da	2669.2450	A	1	0	1605101
rcv	2669.2594	D	1605451	50	1
snd/da	2669.2599	A	1	0	1605101
rcv	2669.3412	D	1605501	50	1
snd/da	2669.3422	A	1	0	1605101
rcv	2669.3804	D	1605101	50	1
snd	2669.3809	A	1	0	1605151
rcv	2669.4265	D	1605551	35	1
snd/da	2669.4270	A	1	0	1605151
rcv/da	2670.3392	D	651	50	1
snd/da	2670.3397	A	1	0	1605151
rcv	2670.3402	D	1605151	50	1
snd	2670.3407	A	1	0	1605251
rcv	2671.2607	D	1605251	50	1
snd	2671.2612	A	1	0	1605586
snd	2671.2786	A	1	0	1605587
rcv	2671.2786	F	1	0	1605587
rcv	2671.2796	A	1605587	0	2

snd/da	0.6537	A	1	0	301
rcv	0.6820	D	401	50	1
snd/da	0.6840	A	1	0	301
rcv	0.6989	D	451	50	1
snd/da	0.7113	A	1	0	301
rcv	0.7212	D	501	50	1
snd/da	0.7375	A	1	0	301
rcv/corr	0.7514	D	301	50	1
rcv	0.7738	D	551	50	1
snd/da	0.7757	A	1	0	301
rcv	0.8055	D	601	50	1
snd/da	0.8070	A	1	0	301

```
=====================================:
Amount of Data Received (bytes) = 2571235
Total segments received =  51429
Data segments received =  51425
Data Segments with bit errors = 3336
Duplicate data segments received = 15977
Duplicate Ack sent = 33201
=====================================:
```