COMP 9331 Assignment for session 2, 2018
Report

Created by: Rongbing Shan (z5186051)

Question 1:

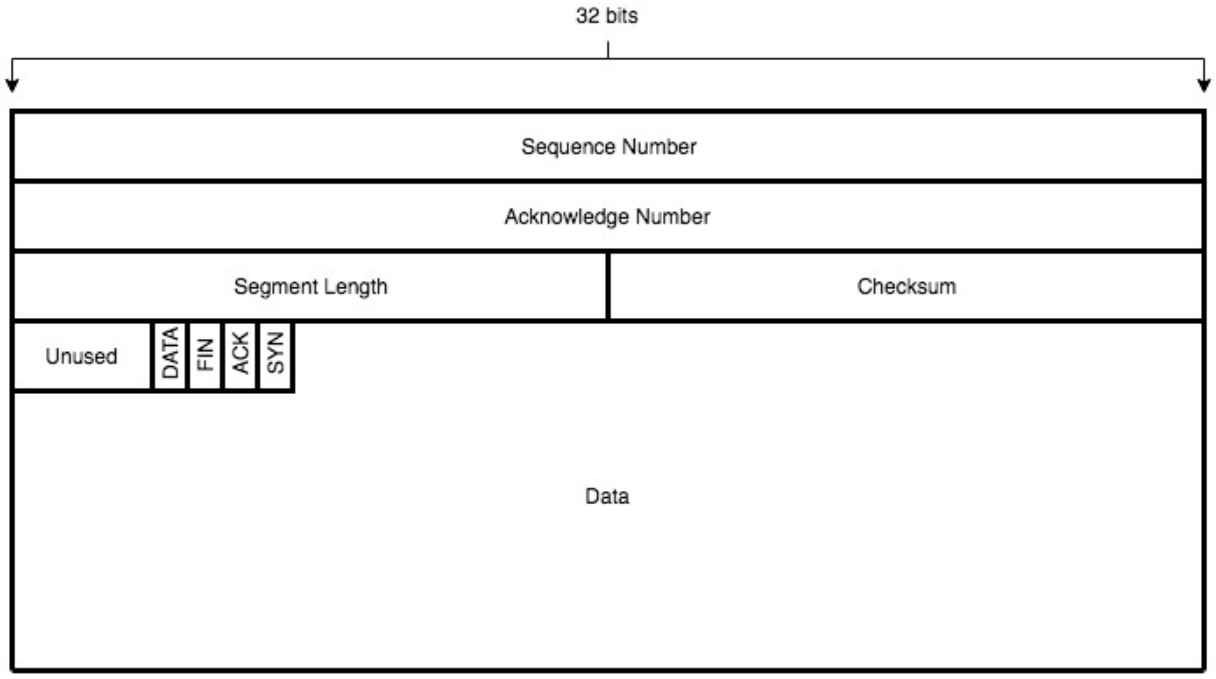
1. First of all, I build a skeleton for simple UDP client and server, which can deliver message from client to server and from server to client. In fact, we have done similar tasks during the weekly Lab, I just rewrite them in the C language.
2. Then the segment header is designed and implemented in **packet** ADT. The necessary fields for reliable data transfer is added to header, which are sequence number, acknowledge number and checksum. And a segment length field is also added to the header in order to implement for MSS, which is a changeable value with user input. FIN, ACK, SYN flags are used for connection and DATA flag is to indicate if a segment contains valid data.
3. Next step, I develop the **buffer** ADT for buffering the packets sent or received. A struct type named **Block** is created as linked list based on **packet**. **BufferWrite()** will always arrange the packets in ascending order of sequence number, therefor the receiver is able to handle out-of-order packets, and sender window will be able to send from the last acknowledged packet.
4. All other functions related to STP are implemented in **stp**. Here all the UDP socket functions like send and receive are replaced by STP functions. Also, write log functions are implemented during this time.
5. Next step, I write the three way handshake and FINACK teardown in sender and receiver separately. As the connection setup and teardown will not go through PLD module, it is simply implemented as sending desired packets(eg, SYN) to the other side and reply with the desired packet(eg, ACK). I make sure that **3-way handshake** and **FIN-ACK FIN-ACK teardown** both worked correctly here.
6. The receiver will receive DATA packets and add them to **recbuffer**. And the receiver will check **recbuff** and determine the proper acknowledge number, and generate the proper ACK packet by **receiver_ack_data_packet()**, and send it back to server. Now the receiver side is fully implemented.
7. PLD module is developed as **cal_PLD()** and added to sender side. Based on the possibilities input by user, a result is returned from the function as either nomal transmit, drop packets, duplicate packets, corrupt packets, re-order packets or delay packets and the packet will be handled accordingly in **sender.c**.
8. To implement the maximum window size, a field **last_ack_sequence** is kept, the sender will only send packets with sequence numbers between last_ack_sequence and last_ack_sequence + MWS.
9. A single timer is added for time out and the timeout interval is calculated every time a non-retransmit packet is successfully acknowledged by receiver.
10. Also, to implement **Fast retransmit**, a field **dupack_rec_times** is added to keep track of the times a same sequence is acknowledged. Whenever dupack_rec_times is larger than 3, the sender will re-transmit the current buffer window.
11. The sender and receiver are tested through a series of scenarios, and log files are examined to guarantee STP works properly.

STP features implemented are all implemented as request.

Both	Three-way handshake (SYN SYNACK ACK) for connection setup
Both	Four-segment(FIN ACK FIN ACK) connection termination
Both	Reliable data transfer, sequence and acknowledge number, checksum
Sender	RTT estimation and RTO estimation single timer time out.
Sender	Fast retransmit
Sender	PLD module
Sender	MWS and MSS
Receiver	Immediate acknowledge

Question 2:

The STP header diagram is:



Sequence Number:	32-bit number used for reliable data transfer.
Acknowledge Number:	32-bit number used for reliable data transfer.
Segment Length:	The length of the segment, including both header and payload.
Checksum:	16-bit checksum for checking bit errors.
DATA:	1-bit flag to indicate if segment contains data.
FIN:	1-bit flag used for connection teardown.
ACK:	1-bit flag used for acknowledgement.
SYN:	1-bit flag used for connection setup.

Question 3:

Tradeoffs:

1. To simplify the implementation, the initial sequence number is made to be always 0. And hence the starting sequence for data packets is always 1.
2. On sender side, the socket receive timeout is set to be 50ms. In this way, the timeout re-transmit will sometimes not be triggered immediately, but a small delay less than 50ms. It can be improved by using select() to monitor the socket description files or use multi thread to interrupt.
3. The maximum MSS is set to be 1024, in fact, I was developing a version with full functional MSS, but it only works on my mac, it doesn't work well on CSE machines.
4. The file to be sent is first encapsulated and save to memory in my design. It takes a lot of memory if the file size is really large. It can be improved.

Future improvements:

1. Multi-thread could be used to make the program more efficient.
2. Point 2 in tradeoff can be improved by encapsulate only the maximum window size allowed of data, and once a packet is acknowledged, next MSS size of data is encapsulated and previous packet in memory should be released. In this way, the program will be more memory efficient.
3. The program will terminate after one file is sent to server, it can be modified to send multiple files. The receiver will be changed to keep listening once a file is received. It can also receive multiple files at same time by add filename to the header and create separate buffer for them.

Question 4:

Function code check_sum() in packet.c is borrowed from Web to calculate the checksum for header. It has been modified specified for my STP implementation.

Question 5:

(a)

With pDrop = 0.1, there are 3 packet drop, and its sequence number is 101, 2601, 2901. After packet 1, the receiver receivers 201, which means 101 is missed. After packet 501, the sender receives 3 duplicate ACKs and it retransmit 101. Same happen for 2601. However with 2901, since there is not enough packets left to trigger fast retransmit, it is retransmit by timeout.

With pDrop = 0.3, there are 12 packet drops, and the sequence numbers are: 101, 301, 601, 1101, 1401, 1701, 1801, 2101, 2301, 2401, 2601, 2801. The overall behavior is same as drop rate of 0.1

With drop rate higher, there are more packet lost during the transmission, and it takes more time than the lower drop rate since some retransmission needs to be trigger by timeout instead of fast retransmit. In other ways, we can say fast re-transmit saves the transmission time over timeout.

(b)

gamma	No. of STP packets	Time Taken(s)
2	9475	700.35
4	9460	717.01
6	9441	731.35

As the DevRTT represends the variation of the status in the channel, the higher we set the value gamma, the larger our time out will be affected by the variation, which means the channel RTT is not close to constant, hence we will have longer timeout interval, leading to longer transmitting time.

(c)

The File has been successfully transferred.
The overall transfer took 253.72 seconds .
pDrop and pCorrupt is the factor that determines more on the overall transfer time. Since pDrop and pCorrupt can have the chance to cause the timeout, where in our case, as the re-order size is 4, re-order will more like to cause fast retransmit. Duplicate packets do not have much impact on the transfer time.

Appendix

(a) Sequence arrived for test0.pdf with pDrop = 0.1/0.3, MWS = 500 bytes, MSS = 100 bytes, seed = 100, gamma = 4, and pDuplicate, pCorrupt, pOrder, MaxOrder, pDelay, MaxDelay = 0. Sequence at receiver:

pDrop = 0.1	pDrop = 0.3
sequence	sequence
0	0
1	1
1	1
201	201
301	401
401	501
501	101
101	701
601	301
701	801
801	901
901	1001
1001	601
1101	1201
1201	1301
1301	1501
1401	1101
1501	1601
1601	1401
1701	1901
1801	2001
1901	1701
2001	2201
2101	1801
2201	2501
2301	2101
2401	2701
2501	2301
2701	2401
2801	2901
3001	3001
2601	2601
2901	2801
3029	3029
1	1

(b) Screen caps for question c

Sender log:

First 20 lines:

snd	0.00	S	0	0	0
rcv	0.00	SA	0	0	1
snd	0.00	A	1	0	1
snd	12.08	D	1	50	1
snd	12.08	D	51	50	1
snd/dup	12.08	D	101	50	1
snd/dup	12.08	D	101	50	1
snd	12.08	D	151	50	1
snd/corr	12.08	D	200	50	1
snd	12.08	D	251	50	1
snd	12.08	D	301	50	1
snd	12.09	D	401	50	1
snd	12.09	D	451	50	1
rcv	12.09	A	1	0	51
snd/corr	12.09	D	500	50	1
rcv	12.09	A	1	0	101
snd	12.09	D	551	50	1
rcv	12.09	A	1	0	151
rcv/DA	12.09	A	1	0	151
rcv	12.09	A	1	0	201

Last 20 lines and summary:

snd	253.56	D	1605501	50	1
snd/RXT	253.57	D	1605051	50	1
rcv/DA	253.57	A	1	0	1605051
rcv/DA	253.57	A	1	0	1605051
rcv/DA	253.57	A	1	0	1605051
snd/RXT	253.57	D	1605051	50	1
rcv/DA	253.57	A	1	0	1605051
rcv/DA	253.57	A	1	0	1605051
rcv	253.58	A	1	0	1605101
snd/dup	253.58	D	1605551	35	0
snd/dup	253.58	D	1605551	35	0
rcv/DA	253.58	A	1	0	1605101
rcv/DA	253.59	A	1	0	1605101
rcv/DA	253.59	A	1	0	1605101
snd/RXT	253.59	D	1605101	50	1
rcv	253.60	A	1	0	1605301
snd/RXT	253.65	D	1605301	50	1
rcv	253.65	A	1	0	1605401
snd/RXT	253.71	D	1605401	50	1
rcv	253.72	A	1	0	1605586
snd	253.72	F	1605586	0	0
rcv	253.72	A	1	0	1605587
rcv	253.72	F	1	0	1
snd	253.72	A	1	0	2

```
============================================================
Size of the file (in Bytes)                          1605585
Segments transmitted (including drop & RXT)            48122
Number of Segments handled by PLD                      32112
Number of Segments Dropped                              3286
Number of Segments Corrupted                            2519
Number of Segments Re-ordered                            456
Number of Segments Duplicated                           2893
Number of Segments Delayed                                 0
Number of Retransmissions due to timeout                5813
Number of Fast Retransmissions                          6844
Number of Duplicate Acknowledgements received          30430
============================================================
```

Receiver log:

First 20 lines:

rcv	2.14	S	0	0	0
snd	2.14	SA	0	0	1
rcv	2.14	A	1	0	1
rcv	14.22	D	1	50	1
snd	14.22	A	1	0	51
rcv	14.22	D	51	50	1
snd	14.22	A	1	0	101
rcv	14.22	D	101	50	1
snd	14.22	A	1	0	151
rcv	14.22	D	101	50	1
snd	14.22	A	1	0	151
rcv	14.22	D	151	50	1
snd	14.22	A	1	0	201
rcv	14.22	D	200	50	1
rcv/corr	14.22	D	200	50	1
rcv	14.22	D	251	50	1
snd/DA	14.22	A	1	0	201
rcv	14.23	D	301	50	1
snd/DA	14.23	A	1	0	201
rcv	14.23	D	401	50	1
snd/DA	14.23	A	1	0	201

Last 20 lines with summary:

rcv	255.71	D	1605501	50	1
snd/DA	255.71	A	1	0	1605051
rcv	255.71	D	1605051	50	1
snd	255.71	A	1	0	1605101
rcv	255.72	D	1605051	50	1
snd	255.72	A	1	0	1605101
rcv	255.72	D	1605551	35	0
snd/DA	255.72	A	1	0	1605101
rcv	255.72	D	1605551	35	0
snd/DA	255.73	A	1	0	1605101
rcv	255.73	D	1605101	50	1
snd/DA	255.73	A	1	0	1605301
rcv	255.79	D	1605301	50	1
snd/DA	255.79	A	1	0	1605401
rcv	255.85	D	1605401	50	1
snd/DA	255.85	A	1	0	1605586
rcv	255.86	F	1605586	0	0
snd	255.86	A	1	0	1605587
snd	255.86	F	1	0	1
rcv	255.86	A	1	0	2

```
================================================
Amount of data received (bytes)       2823580
Total Segments Received                 42785
Data segments received                  40262
Data segments with Bit Errors            2519
Duplicate data segments received         8150
Duplicate ACKs sent                     34919
================================================
```