Implement of STP protocol

Successfully implemented features:
simplified sender, fast retransmit, resend after timeout
concept used:
sequence numbers, cumulative acknowledgements, timer, buffer, window
Logic in Sender and Server:

Sender:

3 threads: timer, sender_thread, and receiver_thread.

Global variable: GIOBAL_STATE 0: hand-shake 1: transmission 2: close connection

Sender_thread:

The starting state of base is equal to nextseq.

When the state is transmission (1), if the next packet's sequence number nextseq is less than the file length, then continue.

If nextseq + MSS is less than the length of the file, then package an MSS-sized package, or package it to the end of the file.

If nextseq is less than base + window size, then continue to package to send, otherwise wait until window has space.

After packing, if the drop is determined, print the drop log, nextseq plus MSS.

If determine send, then print sdn log, nextseq plus MSS and send.

In both cases, if base is equal to nextseq, timer need to be restarted.

Receiver_thread:

The initial state is an empty array rcvACKs, whenever received a confirmation, put into the array, then base move to the confirmation of the ack, if receive four consecutive ack (1 original 3 repeat) pop out the last 3 acks, to prevent the sender from send repeat packets in short time, and then perform the resend function.

Resend function:

If determine the drop, then print the drop log, nextseq plus MSS.

If determine send, then print sdn, nextseq plus MSS and send.

Timer_thread:

The first timer start when the base = 0 send / drop. Restart timer conditions are: when the base equal to nextseq when sending packets; or due to timeout.

Receiver:

Maintain a Seqlist to save ordered ACKs and a dictionary to save all received data.

Receiver receives packet, if the length of packet is more than length of header, which means the packet contains data, unpack it.

If the seq of data is the same as the ACK sent, which means this packet is ordered otherwise send the last ACK again to tell sender, it's not the packet expected. Save the unorder packet and until the right one arrives, keep them in right order and write all of them to target file.

STP header

32 bit (4 byte)
Acknowledgement number
Sequence number
FIN flag
SYN flag
DATA

The STP header has 4 parts, all of them are int. Acknowledgement number is the sequence number of the packet expected to receive, and sequence number is for the send packet. When SYN flag is one, there will be no data and sender and server build a connection. When SYN flag is one they are closing connection.

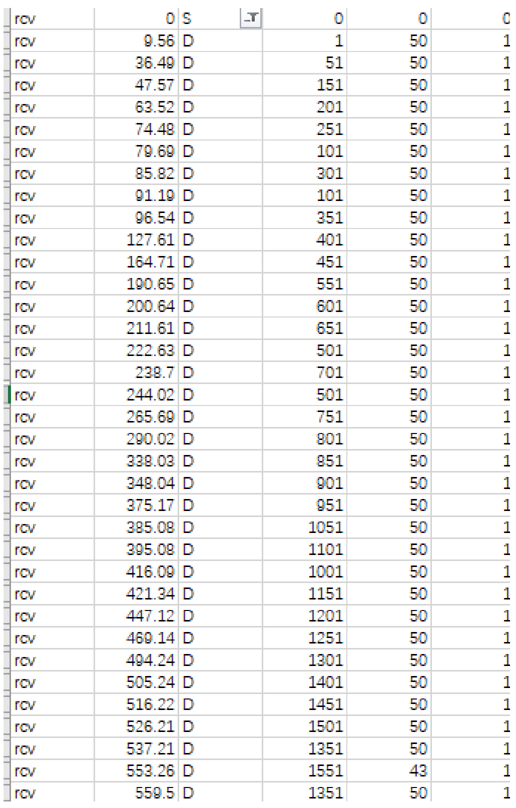
Question answer

a) As text book recommended I choose 1s which is 1000ms as a start. The program did well and drop rate is round 0.1. Then I tried 100ms. No much difference on retransmission rate but . I tried it with no drop and find the rrt is about 12ms. So then I tried 20ms which cause really high retransmission rate. At last I choose 50ms as timeout value.

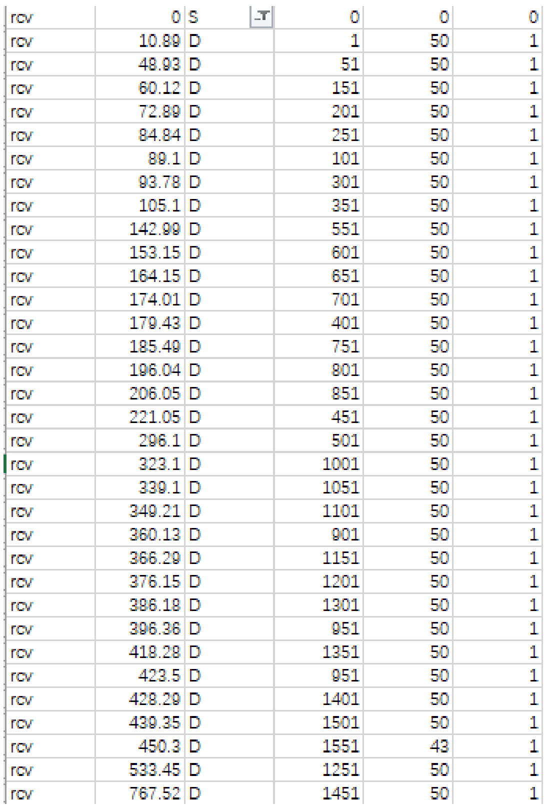
   When pdrop increases, the retransmission occurs more often so the whole transmission need more time. The resulting sequence is more unorder when more drop happens. Also there is higher possibility to receive more duplicated packets caused by fast transmission and timeout retransmission.

Left picture: filename = "test1.txt", mws = 500, mss = 50, pdrop= 0.1, seed = 300
Right picture: filename = "test1.txt", mws = 500, mss = 50, pdrop= 0.3, seed = 300

	Pdrop=0.1	Pdrop=0.3
Droped seq#	101 501 1001 1351	101*2 401 451 501*2 751 901 1201 1401 1451*2



rcv	0	S	0	0	0
rcv	9.56	D	1	50	1
rcv	36.49	D	51	50	1
rcv	47.57	D	151	50	1
rcv	63.52	D	201	50	1
rcv	74.48	D	251	50	1
rcv	79.69	D	101	50	1
rcv	85.82	D	301	50	1
rcv	91.19	D	101	50	1
rcv	96.54	D	351	50	1
rcv	127.61	D	401	50	1
rcv	164.71	D	451	50	1
rcv	190.65	D	551	50	1
rcv	200.64	D	601	50	1
rcv	211.61	D	651	50	1
rcv	222.63	D	501	50	1
rcv	238.7	D	701	50	1
rcv	244.02	D	501	50	1
rcv	265.69	D	751	50	1
rcv	290.02	D	801	50	1
rcv	338.03	D	851	50	1
rcv	348.04	D	901	50	1
rcv	375.17	D	951	50	1
rcv	385.08	D	1051	50	1
rcv	395.08	D	1101	50	1
rcv	416.09	D	1001	50	1
rcv	421.34	D	1151	50	1
rcv	447.12	D	1201	50	1
rcv	469.14	D	1251	50	1
rcv	494.24	D	1301	50	1
rcv	505.24	D	1401	50	1
rcv	516.22	D	1451	50	1
rcv	526.21	D	1501	50	1
rcv	537.21	D	1351	50	1
rcv	553.26	D	1551	43	1
rcv	559.5	D	1351	50	1



rcv	0	S	0	0	0
rcv	10.89	D	1	50	1
rcv	48.93	D	51	50	1
rcv	60.12	D	151	50	1
rcv	72.89	D	201	50	1
rcv	84.84	D	251	50	1
rcv	89.1	D	101	50	1
rcv	93.78	D	301	50	1
rcv	105.1	D	351	50	1
rcv	142.99	D	551	50	1
rcv	153.15	D	601	50	1
rcv	164.15	D	651	50	1
rcv	174.01	D	701	50	1
rcv	179.43	D	401	50	1
rcv	185.49	D	751	50	1
rcv	196.04	D	801	50	1
rcv	206.05	D	851	50	1
rcv	221.05	D	451	50	1
rcv	296.1	D	501	50	1
rcv	323.1	D	1001	50	1
rcv	339.1	D	1051	50	1
rcv	349.21	D	1101	50	1
rcv	360.13	D	901	50	1
rcv	366.29	D	1151	50	1
rcv	376.15	D	1201	50	1
rcv	386.18	D	1301	50	1
rcv	396.36	D	951	50	1
rcv	418.28	D	1351	50	1
rcv	423.5	D	951	50	1
rcv	428.29	D	1401	50	1
rcv	439.35	D	1501	50	1
rcv	450.3	D	1551	43	1
rcv	533.45	D	1251	50	1
rcv	767.52	D	1451	50	1

b)

	Packets sent	Packets droped	Time(ms)
T	42	4	990
4T	42	4	1002.06
0.25T	198	21	5751.69

In this case, T costs least time since 50ms is suitable for my program, it happens the same retransmission as 4T, which means it uses fast retransmission as in 4T case and thanks for short timeout, the transmit because of timeout happens faster than in 4T case.

In 0.25T case, retransmission happens too often, the total time becomes really long because of unnecessary retransmission.
Possible cases:
Total_time(Tcurrent)> total_time(4*Tcurrent):
In 4T case has more ack come back before timeout retransmit, fast re-transmit happens more which makes transmission faster.

Appendix

See in folder.