CSE 489/589
Programming Assignment 2
Reliable Transport Protocols

1 - SANITY Tests

[2.0] ABT

ABT Description:
In this protocol, on the A side we maintain a vector a_buf of packets and also track the current sequence number in a boolean as we only need two sequence numbers here - 0 and 1. In A_init() we initialize required values. In A_output() we construct a packet and buffer it. If A is not waiting for an acknowledgement, we take the first packet from the buffer, send it and start the timer. In A_input() if the checksum is valid, we remove the first element from the buffer and stop the timer. If there are still packets in the buffer, we send the first packet and start the timer, or else we stop the timer. If wrong acknowledgement was received, we retransmit the first packet. In A_timerinterrupt(), we send the first packet and start the timer if the buffer is not empty.
On the B side, in B_init() we initialize the required values. In B_input(), we validate the checksum and if valid we send an acknowledgement packet back to A and also deliver the message to layer 5 of B.
Also we have two user defined functions construct_packet() to construct a packet and calculate_checksum() to calculate the checksum. These are the same across all our 3 protocols.

(Put screenshots of the grader here…)
Note: Since the test outputs are very long, we are including only the overall result in the screenshot and typing the command we ran in the next line.

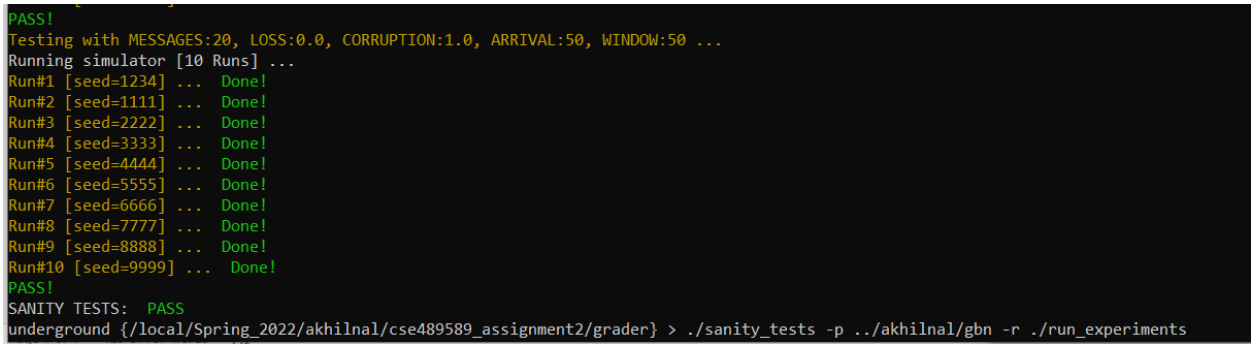
```
Testing with MESSAGES:20, LOSS:0.0, CORRUPTION:1.0, ARRIVAL:1000, WINDOW:0 ...
Running simulator [10 Runs] ...
Run#1 [seed=1234] ...  Done!
Run#2 [seed=1111] ...  Done!
Run#3 [seed=2222] ...  Done!
Run#4 [seed=3333] ...  Done!
Run#5 [seed=4444] ...  Done!
Run#6 [seed=5555] ...  Done!
Run#7 [seed=6666] ...  Done!
Run#8 [seed=7777] ...  Done!
Run#9 [seed=8888] ...  Done!
Run#10 [seed=9999] ...  Done!
PASS!
SANITY TESTS:  PASS
underground {/local/Spring_2022/akhilnal/cse489589_assignment2/grader} > ./sanity_tests -p ../akhilnal/abt -r ./run_experiments
```

[5.0] GBN

GBN Description:
In this protocol, on the A side, we maintain 2 buffers - a vector of msgs in a_buf and a vector of packets that might need to be retransmitted in a_retrans. We also maintain starting and ending points of a_buf and the starting point (base) of the sender window. In A_init() we reserve space

for the number of items in each buffer and initialize the required variables. In A_output() if a_seq_no is out of the window we buffer it in a_buf. We construct a packet from the message and send it be and also buffer it in a_retrans in case it needs to be retransmitted. We start the timer only if we are at the start of our window. In A_input(), we validate the checksum and move the window forward if valid. Then if there are any packets in the a_buf that haven't been sent yet, we send them and also buffer them in a_retrans. If there are no packets left in a_buf we stop the timer or else we restart the timer. In A_timeinterrupt(), we send all the packets in our window since we haven't got any acknowledgements for them. We also start the timer.
On the B side in B_init(), we initialize the expected sequence number to 0 in the variable b_seqno. In B_input(), we validate the checksum. If checksum is valid, we check if we got the expected packet and if yes we deliver the packet and also send an ack packet; if not we send an acknowledgement packet with the previously received acknowledgement number.

(Put screenshots of the grader here…)



[8.0] SR

SR Description:
In the SR protocol on the A side we maintain two buffers - a vector a_buf that holds messages(a also variables to maintain its starting and ending points) and another vector a_retrans that holds objects of the struct resend_pkt i.e. packets that might need to be retransmitted. The struct resend_pkt stores the packet, a boolean to check if its been acknowledged and a timeout value for that particular packet since each packet has a separate timeout time. Below is the declaration for the resend_pkt struct.

```
struct resend_pkt
{
  pkt p;
  bool ack;
  float timeout;
};
```

In A_init() we reserve the space for the buffers and also initialize other required variables. In A_output(), we buffer the message in a_buf if a_seq_no is out of our current window. We create a packet from the message and send it to B via layer 3. We also calculate the timeout for this packet current time + TIMER_VALUE. We create a resend_pkt object and store it in the a_retrans buffer. We start the timer only if the current seq number (a_seq_no) is at the base of the window. In A_input(), we validate the checksum and if the acknowledgement number falls in the window, we mark it as acknowledged. If the received acknowledgement is the smallest unacknowledged packet, we advance the window base until the next unacknowledged sequence number. Then if there are any packets in the a_buf that haven't been sent yet, we send them and also buffer them in a_retrans after updating their corresponding timeouts. We stop the timer if a_seq_no is at the window base. In A_timerinterrupt we check the timeout of each packet in the window using the timeout value of that packet in a_retrans and if the value is equal to current time and it still hasn't been acknowledged, we retransmit it and update its timeout value. We also start the timer with the timeout value of the unacknowledged packet that has the smallest relative time i.e., packet which is supposed to timeout next. (This is calculated using another function we defined, get_min_time()).
On the B side, we maintain a buffer b_buf which is a vector of structs b_buf_item which has a packet and a boolean field that maintains whether it's valid or not. Below is the declaration of that struct.

```
struct b_buf_item
{
  pkt p;
  bool valid;
};
```

In B_init() we reserve the space for the buffer and initialize b_seq_no to 0 which is the expected seq number. In B_input(), we validate the checksum and also check if it falls in B's window and proceed forward if the two conditions are met. We send an acknowledgement packet for the received packet. If the received packet is out of order we buffer it in b_buf. If it's in order, we advance B's window. We also deliver all the packets to the upper layer if there are any in the buffer that fall in order with the reception of this packet.

(Put screenshots of the grader here…)

```
PASS!
Testing with MESSAGES:20, LOSS:0.0, CORRUPTION:1.0, ARRIVAL:50, WINDOW:50 ...
Running simulator [10 Runs] ...
Run#1 [seed=1234] ...  Done!
Run#2 [seed=1111] ...  Done!
Run#3 [seed=2222] ...  Done!
Run#4 [seed=3333] ...  Done!
Run#5 [seed=4444] ...  Done!
Run#6 [seed=5555] ...  Done!
Run#7 [seed=6666] ...  Done!
Run#8 [seed=7777] ...  Done!
Run#9 [seed=8888] ...  Done!
Run#10 [seed=9999] ...  Done!
PASS!
SANITY TESTS:  PASS
underground {/local/Spring_2022/akhilnal/cse489589_assignment2/grader} > ./sanity_tests -p ../akhilnal/sr -r ./run_experiments
```

[No further grading for the protocol that fails a SANITY test.]

2 - BASIC Tests

[5.0] ABT

(Put screenshots of the grader here…)

```
PASS!
Testing with MESSAGES:20, LOSS:0.0, CORRUPTION:0.8, ARRIVAL:1000, WINDOW:0 ...
Running simulator [10 Runs] ...
Run#1 [seed=1234] ...  Done!
Run#2 [seed=1111] ...  Done!
Run#3 [seed=2222] ...  Done!
Run#4 [seed=3333] ...  Done!
Run#5 [seed=4444] ...  Done!
Run#6 [seed=5555] ...  Done!
Run#7 [seed=6666] ...  Done!
Run#8 [seed=7777] ...  Done!
Run#9 [seed=8888] ...  Done!
Run#10 [seed=9999] ...  Done!
PASS!
BASIC TESTS:  PASS
underground {/local/Spring_2022/akhilnal/cse489589_assignment2/grader} > ./basic_tests -p ../akhilnal/abt -r ./run_experiments
```

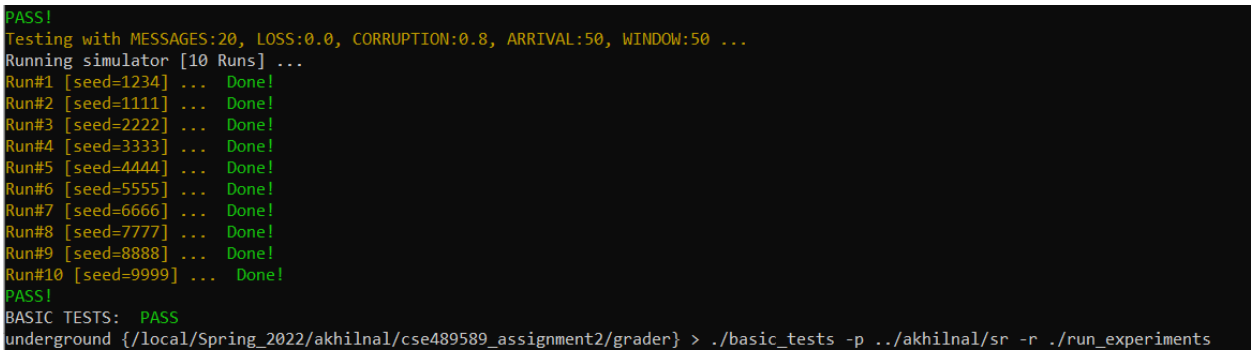
[12.0] GBN

(Put screenshots of the grader here…)

```
PASS!
Testing with MESSAGES:20, LOSS:0.0, CORRUPTION:0.8, ARRIVAL:50, WINDOW:50 ...
Running simulator [10 Runs] ...
Run#1 [seed=1234] ...  Done!
Run#2 [seed=1111] ...  Done!
Run#3 [seed=2222] ...  Done!
Run#4 [seed=3333] ...  Done!
Run#5 [seed=4444] ...  Done!
Run#6 [seed=5555] ...  Done!
Run#7 [seed=6666] ...  Done!
Run#8 [seed=7777] ...  Done!
Run#9 [seed=8888] ...  Done!
Run#10 [seed=9999] ...  Done!
PASS!
BASIC TESTS:  PASS
underground {/local/Spring_2022/akhilnal/cse489589_assignment2/grader} > ./basic_tests -p ../akhilnal/gbn -r ./run_experiments
```

[18.0] SR

(Put screenshots of the grader here…)

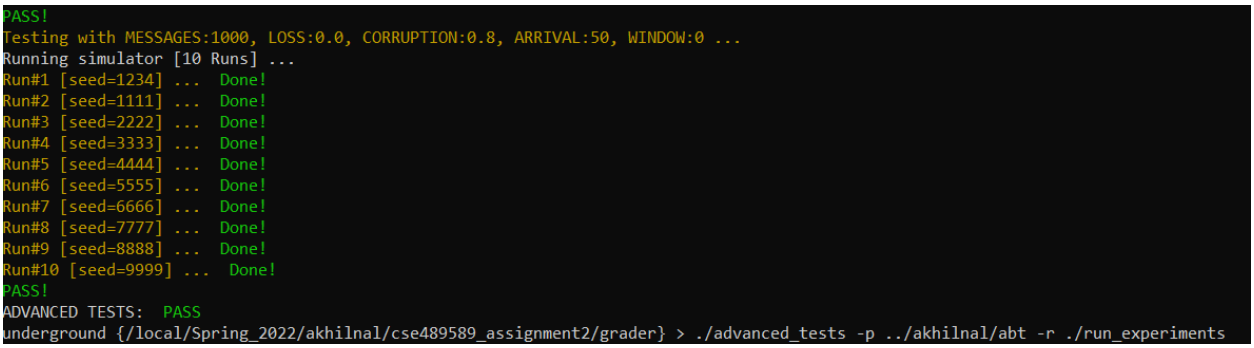


[No further grading for the protocol that fails a BASIC test.]

3 - ADVANCED Tests

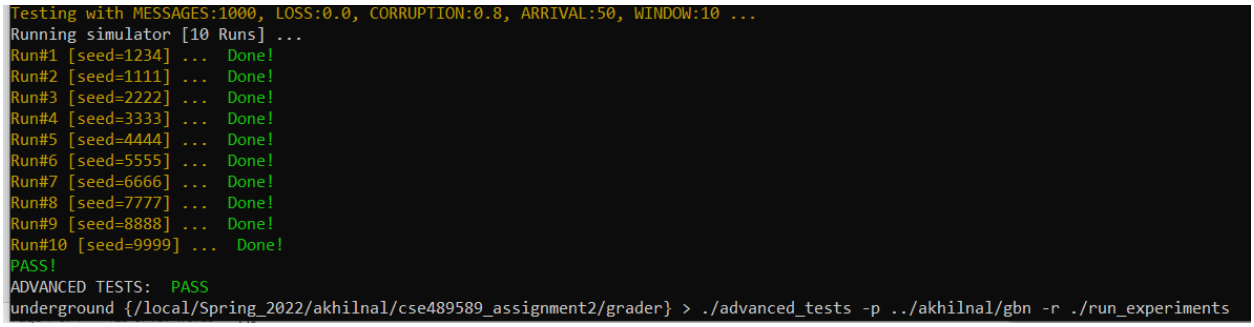
[5.0] ABT

(Put screenshots of the grader here…)



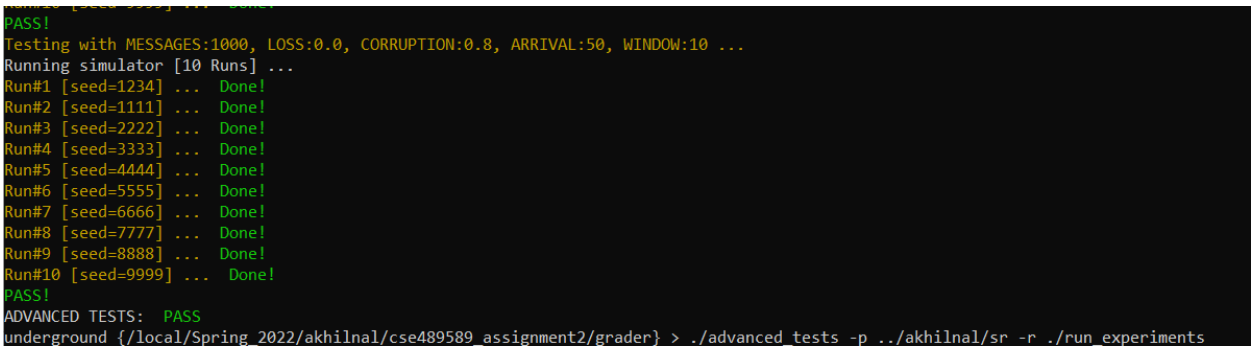
[10.0] GBN

(Put screenshots of the grader here…)



[20.0] SR

(Put screenshots of the grader here…)



4 - ANALYSIS & REPORT [15.0]

(We expect you to use graphs to show your results for each of the experiments in 6.1 and then write down your observations. Further, your report, at the very least, should answer questions like: What variations did you expect for throughput by changing those parameters and why? Do you agree with your measurements; if not then why?)

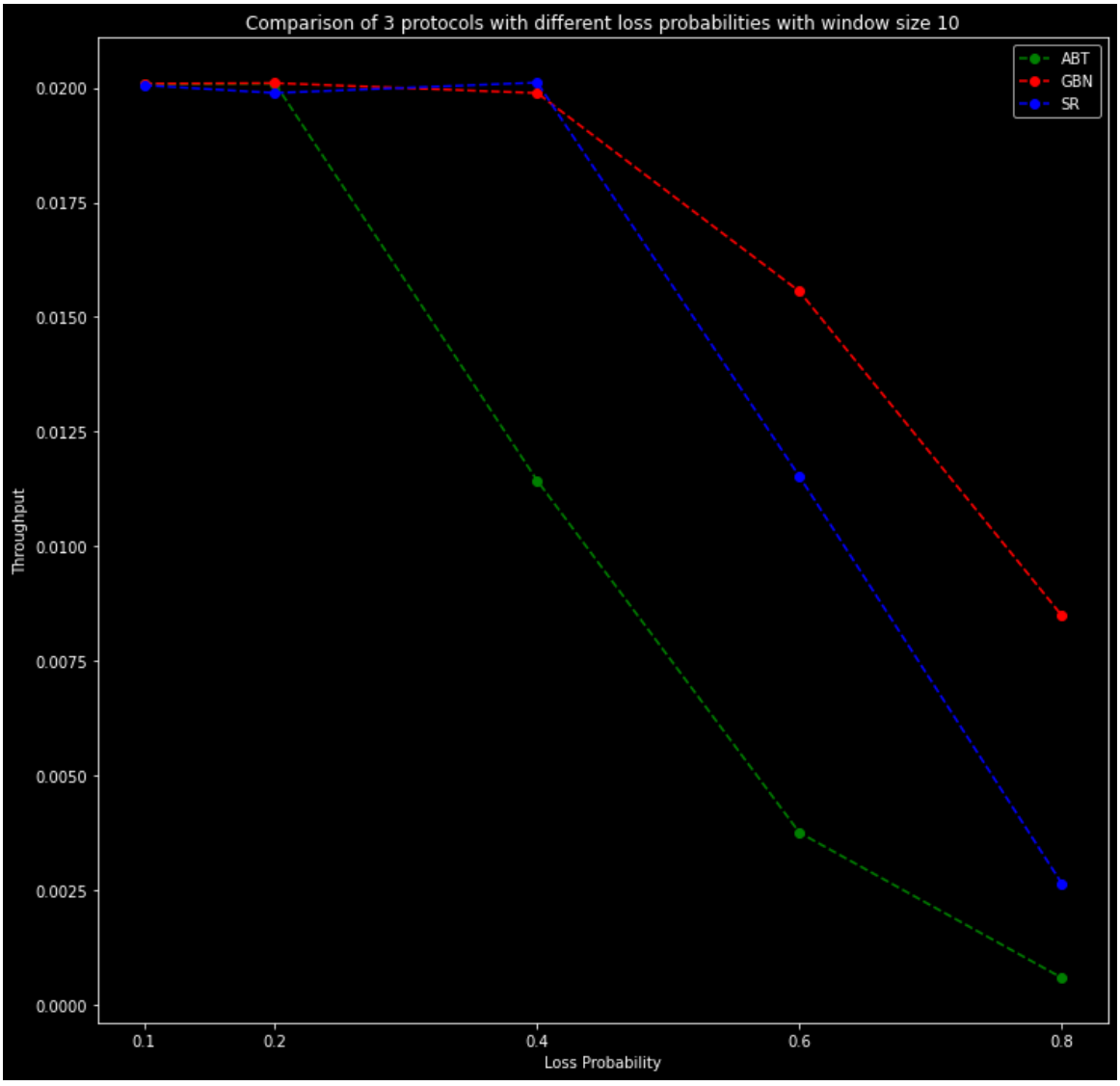
Experiment1:

Part1: With **loss probabilities: {0.1, 0.2, 0.4, 0.6, 0.8}** and **window size 10,** we have the below throughput values (average of all 10 runs) for all the three protocols.

Window size = 10	**Loss probability**				
Protocol	**0.1**	**0.2**	**0.4**	**0.6**	**0.8**

ABT	0.0200777	0.0200968	0.0114326	0.0037549	0.000589
GBN	0.0200899	0.0201031	0.0198841	0.0155588	0.0084925
SR	0.0200541	0.0198865	0.0201138	.0115217	0.0026515

Below is a graph comparing all the three protocols:



Key Observations:

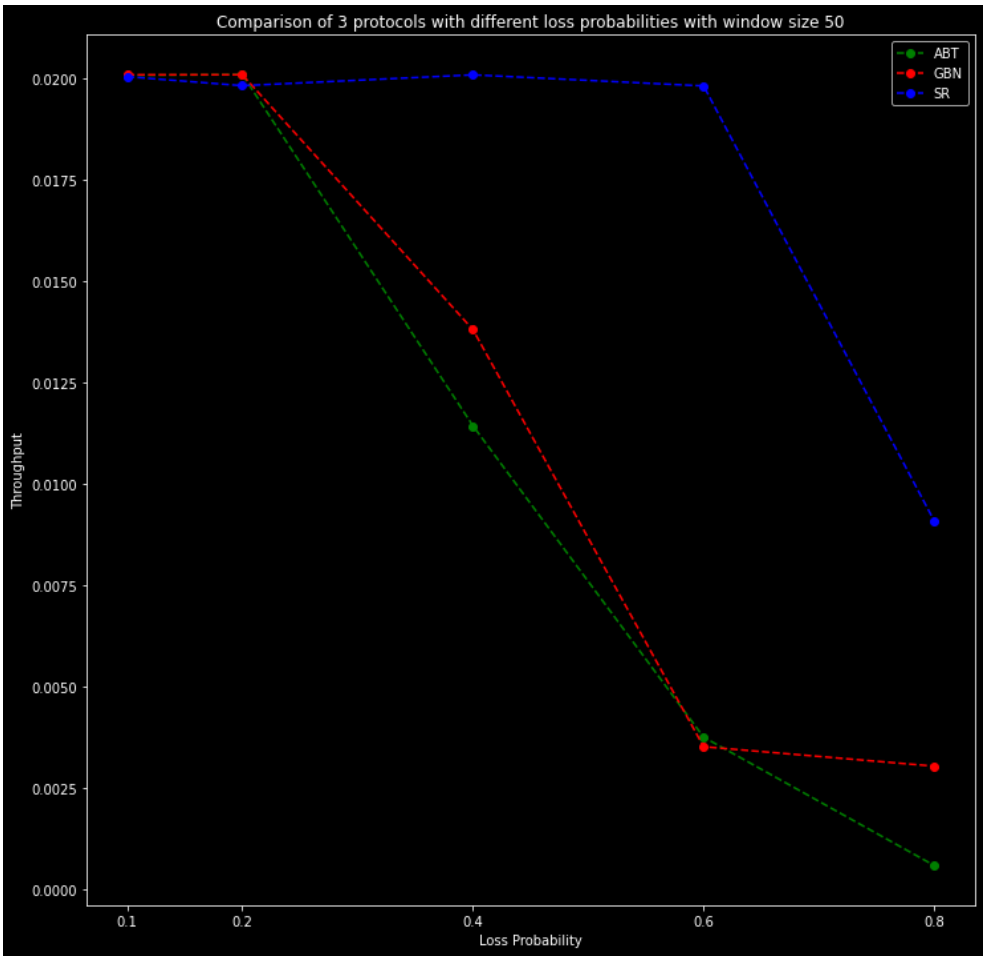
- As seen in the above graph, all the 3 protocols share similar throughput till the loss probability is 0.2.

- However, when the loss probability increases, the throughput of ABT protocol declines more quickly than the other 2. This is because, with rise in loss probability, the number of messages to be retransmitted using ABT protocol also increases, thus lowering its throughput.
- On the other hand, the SR and GBN protocols have almost same throughput till 0.4 loss probability and with further rise in the probability, the former protocol sees a slightly sharp dip in throughput when compared to the latter protocol.
- Thus, for a window of size 10, we can conclude that the GBN protocol has better throughput compared to the other 2 and the SR protocol is better than ABT.

Part2: With **loss probabilities: {0.1, 0.2, 0.4, 0.6, 0.8}** and **window size 50,** we have the below throughput values (average of all 10 runs) for all the three protocols.

Window size = 50	**Loss probability**				
Protocol	**0.1**	**0.2**	**0.4**	**0.6**	**0.8**
ABT	0.0200777	0.0200968	0.0114326	0.0037549	0.000589
GBN	0.0200909	0.0200909	0.0138045	0.0035188	0.0030391
SR	0.0200437	0.0198865	0.020089	0.0198162	0.009087

Below is a graph comparing all the three protocols:



Key Observations:

- As observed earlier, all the 3 protocols share similar throughput till the loss probability is 0.2.
- However, when the loss probability increases, the throughput of both ABT and GBN protocols decline rapidly with ABT having slightly better throughput than GBN. This could be because, with rise in loss probability and for a window of size 50, the number of unacknowledged messages to retransmit also increases compared to the one with window size 10. thus lowering the throughput.
- On the other hand, since the SR protocol has a buffer to store the received packets, the throughput of SR protocol is much better compared to the other 2 protocols.
- Thus, for a window of size 50, we conclude that the SR protocol has better throughput compared to the other 2.

Conclusion:

On the whole, we can infer from experiment 1 that:

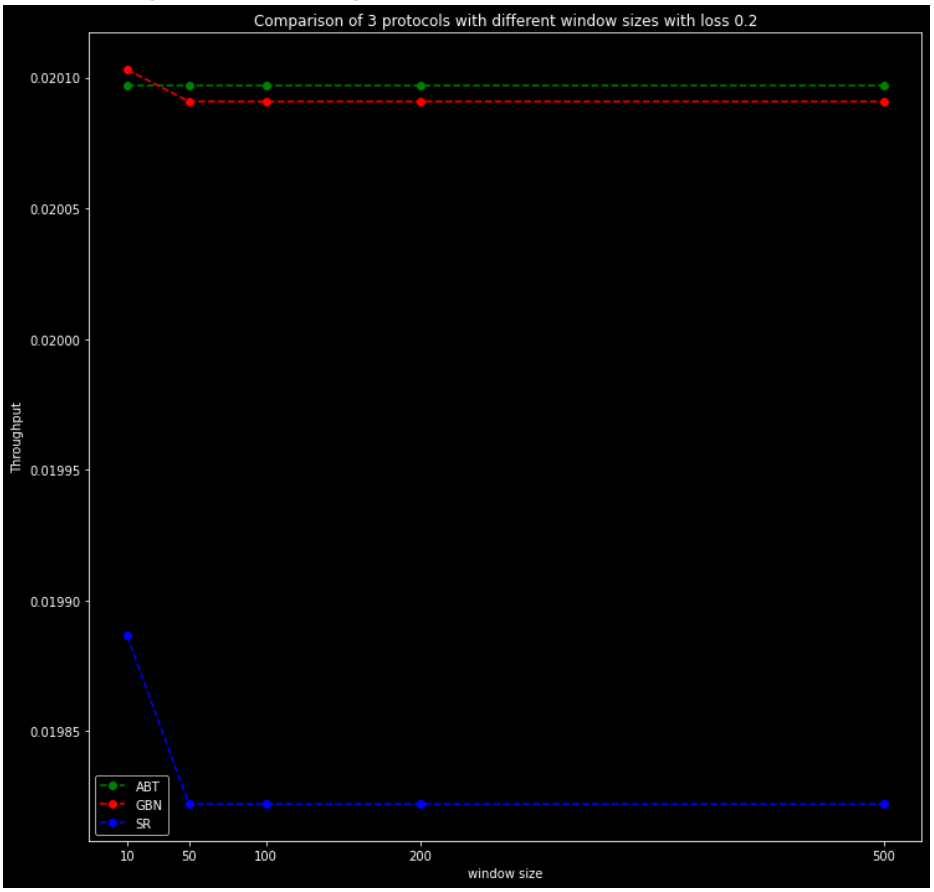
- SR protocol could be seen maintaining better throughput over different loss probabilities.
- GBN protocol has good efficiency for relatively small losses when compared with ABT protocol.

Experiment2:

Part1: With window sizes: {10, 50, 100, 200, 500} and **loss probabilities 0.2** we have the below throughput values (average of all 10 runs) for all the three protocols.

loss = 0.2	**Window Size**				
Protocol	**10**	**50**	**100**	**200**	**500**
ABT	0.0200968	0.0200968	0.0200968	0.0200968	0.0200968
GBN	0.0201031	0.0200909	0.0200909	0.0200909	0.0200909
SR	0.0198865	0.0198222	0.0198222	0.0198222	0.0198222

Below is a graph comparing all the three protocols:



Key Observations:

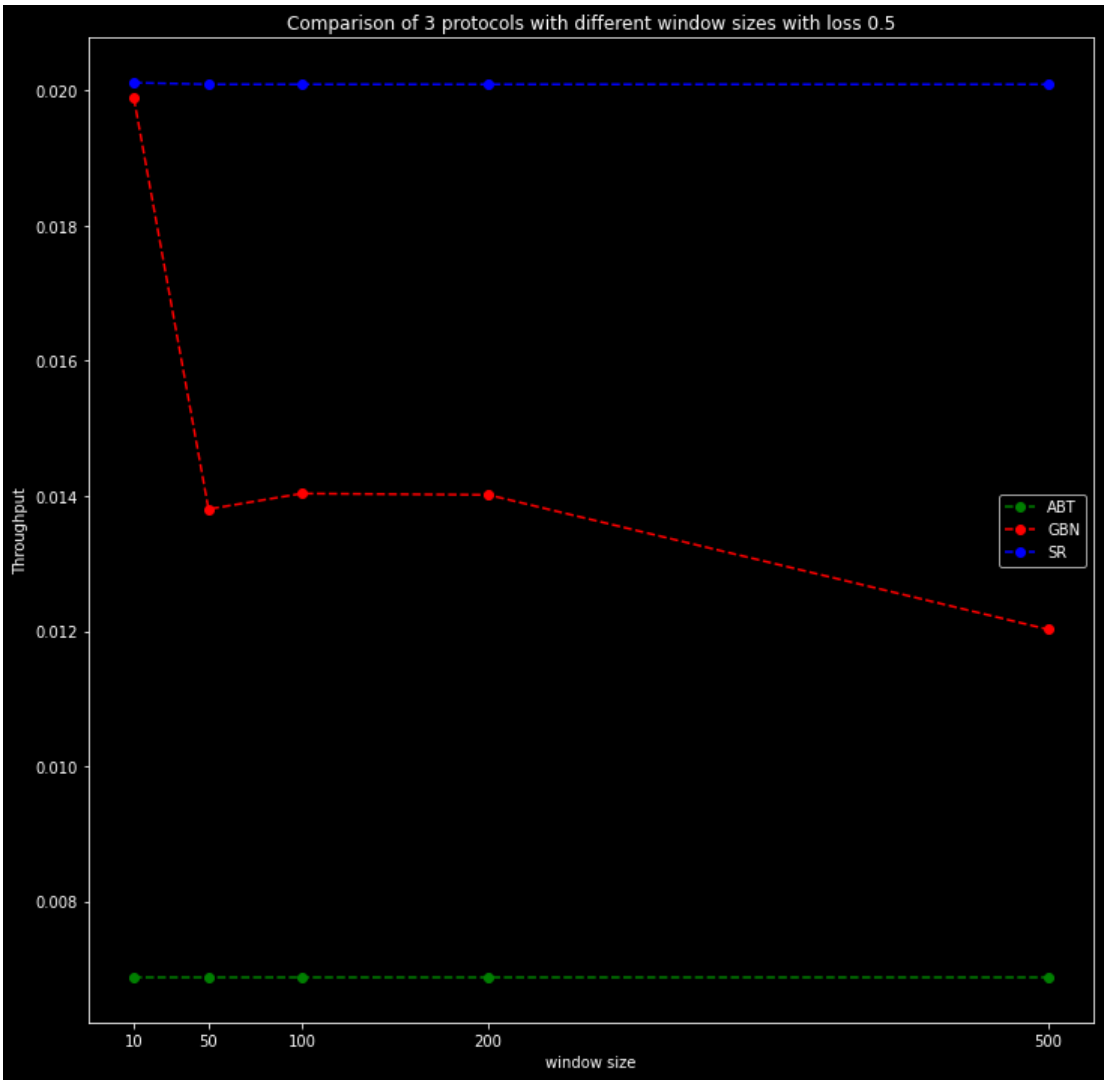
- As seen in the above graph, the ABT and GBN protocols share similar throughput for a loss probability of 0.2 across different window sizes.
- However, when the window size increases, the efficiency of SR protocol declines.

- Thus, for a loss probability of value 0.2, we conclude that ABT and GBN protocols have better throughput over different window sizes.

Part2: With window sizes: {10, 50, 100, 200, 500} and **loss probabilities 0.5** we have the below throughput values (average of all 10 runs) for all the three protocols.

loss = 0.5	**Window Size**				
Protocol	**10**	**50**	**100**	**200**	**500**
ABT	0.0068736	0.0068736	0.0068736	0.0068736	0.0068736
GBN	0.0198841	0.0138045	0.0140353	0.0140154	0.0120243
SR	0.0201138	0.020089	0.020089	0.020089	0.020089

Below is a graph comparing all the three protocols:



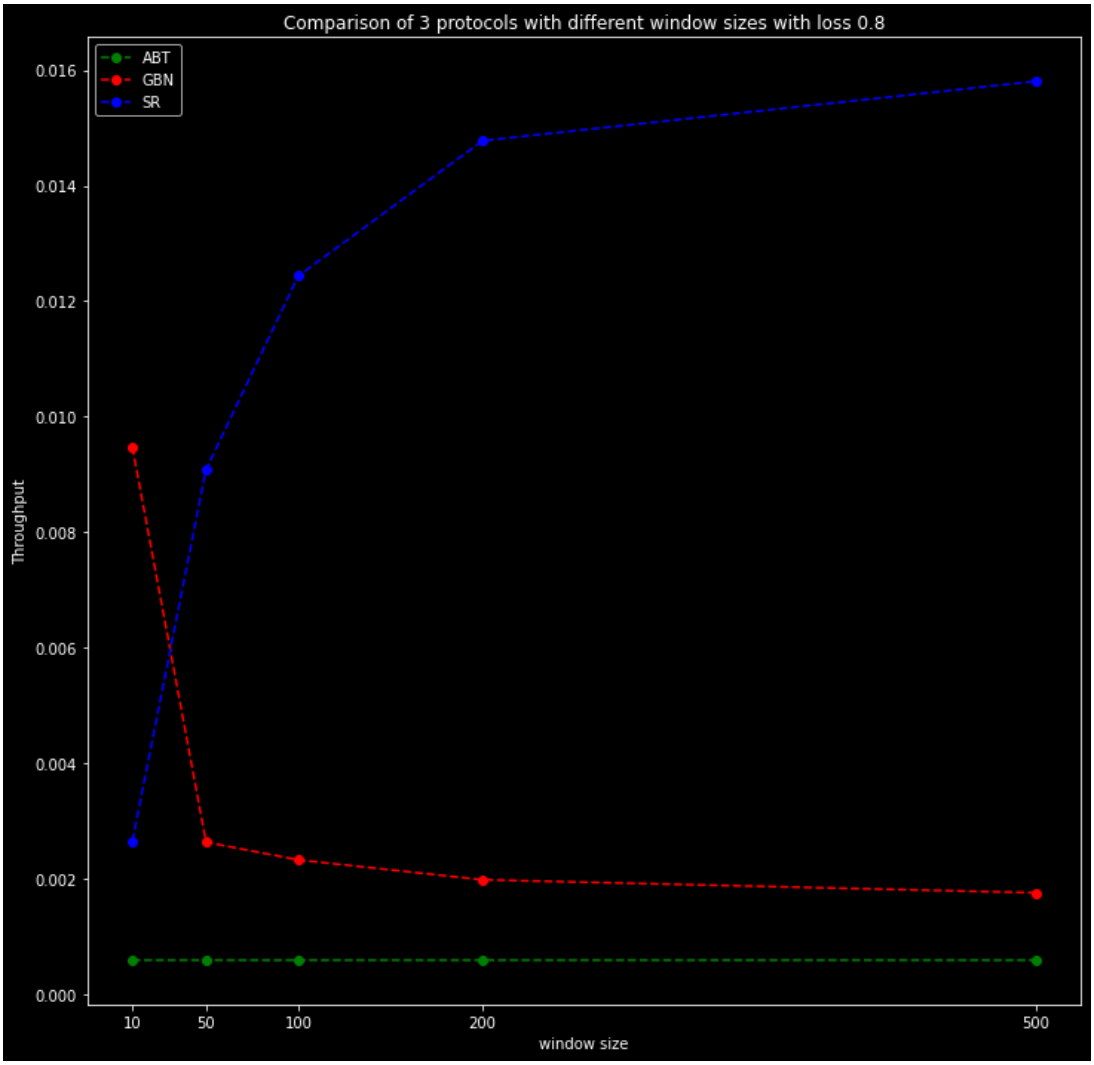
Key Observations:

- As seen in the above graph, when the loss probability is 0.5 and with increase in window size, the SR protocol has much better throughput.
- The efficiency of GBN protocol was initially high when the window size was 10. But with rise in window size, the throughput of GBN protocol gradually drops but is still better when compared to ABT protocol which has the least probability when the loss is 0.5.
- Thus, for a loss probability of value 0.5, we conclude that SR protocol has good throughput over different window sizes.

Part3: With window sizes: {10, 50, 100, 200, 500} and **loss probabilities 0.8** we have the below throughput values (average of all 10 runs) for all the three protocols.

loss = 0.8	Window Size				
Protocol	**10**	**50**	**100**	**200**	**500**
ABT	0.000589	0.000589	0.000589	0.000589	0.000589
GBN	0.0094779	0.0026356	0.0023288	0.0019844	0.0017595
SR	0.0026515	0.009087	0.0124367	0.0147788	0.0158064

Below is a graph comparing all the three protocols:



Key Observations:

- As seen in the above graph, when the loss probability is 0.8, the SR protocol starts with less throughput but with increase in window size, the efficiency also increases and achieves the best possible efficiency over the 3 protocols for larger window sizes.
- On the other hand, the efficiency of GBN protocol was initially high when the window size was 10. But with the rise in window size, the throughput of GBN protocol gradually drops but is still better when compared to ABT protocol which has the least probability for any window size.
- Thus, for a loss probability of value 0.8, we conclude that SR protocol has good throughput over different window sizes.

Conclusion:

On the whole, we can infer from experiment 2 that:

- When compared over different window sizes, SR protocol stands out with better throughput protocol among the three .
- GBN protocol has good efficiency for relatively small losses when compared with ABT protocol.
- ABT protocol could be seen performing better when the loss probability (0.2) is very small.