CSE 489/589
Programming Assignment 2 Report
Reliable Transport Protocols

Notes: (IMPORTANT)

- ➔ One of your group members select <File> - <Make a copy> to make a copy of this report for your group, and share that Google Doc copy with your teammates so that they can also edit it.
- ➔ Report your work in each section. Describe the method you used, the obstacles you met, how you solved them, and the results. You can take screenshots at key points. There are NO hard requirements for your description.
- ➔ For a certain test, if you successfully implemented it, **take a screenshot of the result from the grader as required in section 5 (required).** You can just provide the overall result for each test.
- ➔ For a certain test, if you tried but failed to implement it, properly describe your work. We will partially grade it based on the work you did.
- ➔ Do **NOT** claim anything you didn't implement. If you didn't try on a certain protocol or test, leave that section blank. We will run your code, and if it does not match the work you claimed, you and your group won't get any partial grade score for this WHOLE assignment.
- ➔ There will be **15.0** points for this report. These are NOT bonus points and will be given based on the completion of the analysis part (section 6.1).
- ➔ If you decide not to attempt the analysis part (section 6.1) of the assignment, you will still NEED to submit this report with the requirements stated in section 6.
- ➔ Under **NO** circumstances may you rely on the work of your peers, including but not limited to GitHub repositories or code submissions from previous academic terms.
- ➔ All the analysis results in section 6 should come from one of the provided hosts, NOT on your local machine (see section 3.1 in the handout).
- ➔ The maximum score for PA 2: 85 + 15 = 100

1 - Academic Integrity Policy Statement

[Your submission will NOT be graded without this statement.]

I / We have read and understood the course academic integrity policy in the syllabus of the class.

2 - Group and Contributions

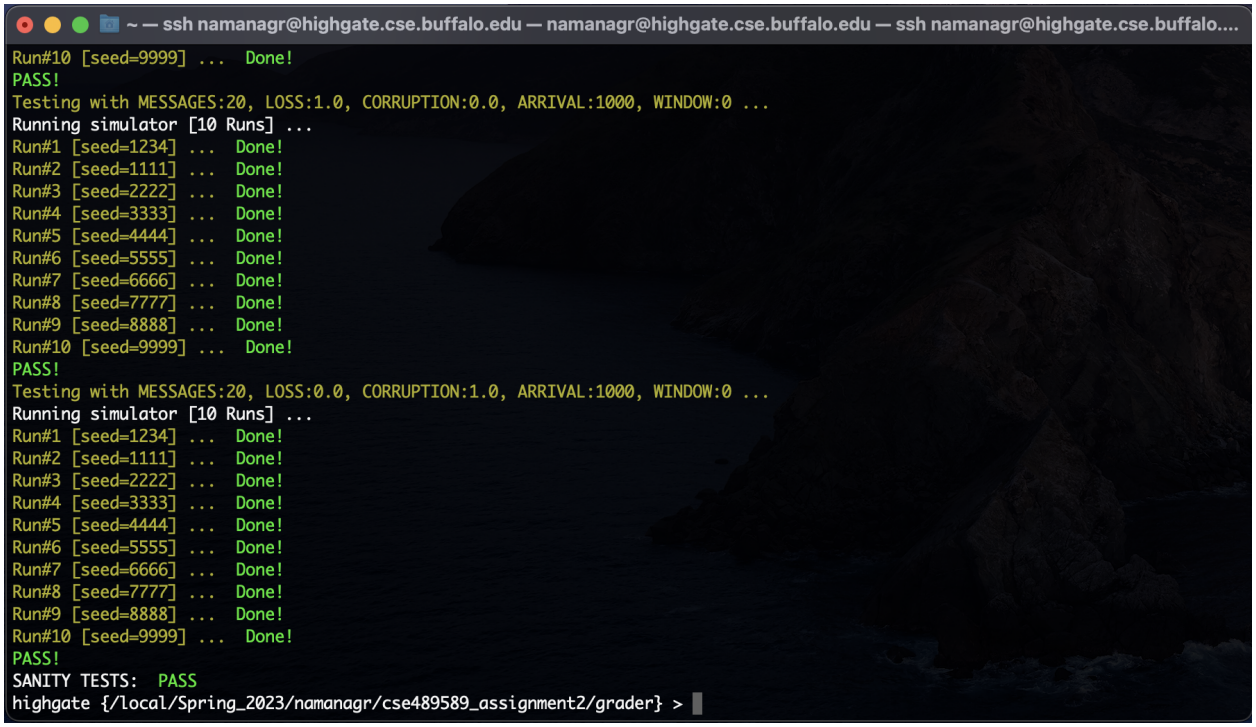
- Name of member 1: **Ayushi Daksh**
  - UBITName: **ayushida**
  - Contributions: GBN, SR, timeout selection experiments

- Name of member 2: **Naman Agrawal**
    - UBITName: **namanagr**
    - Contributions: ABT, SR, experiments, analysis and report

3 - SANITY Tests

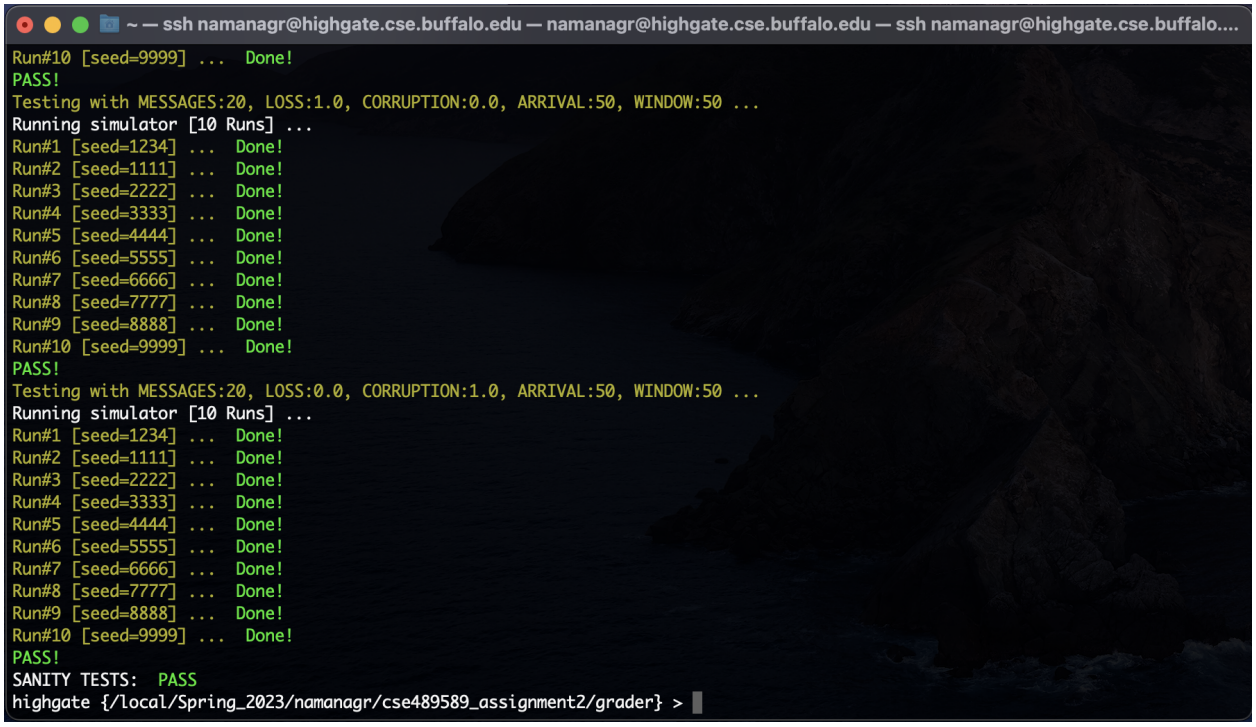
[2.0] ABT

(Put screenshots of the grader here…)



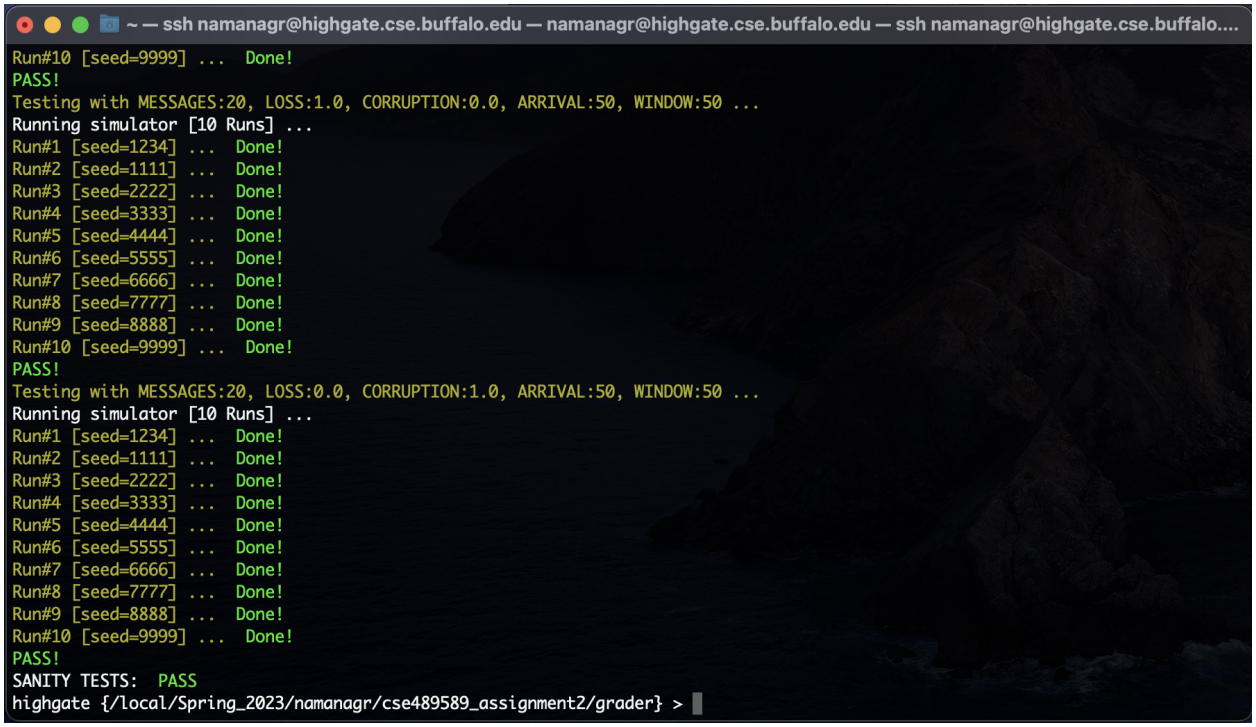
[5.0] GBN

(Put screenshots of the grader here…)



[8.0] SR

(Put screenshots of the grader here…)

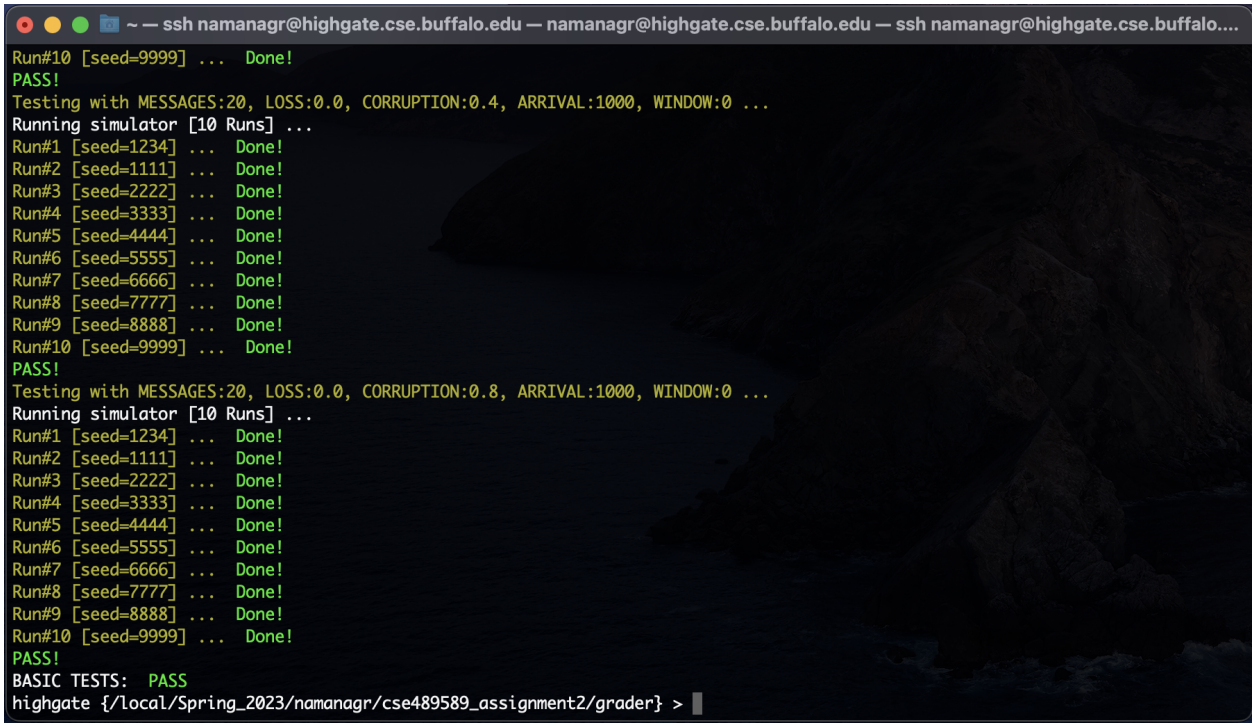


[No further grading for the protocol that fails a SANITY test.]

4 - BASIC Tests

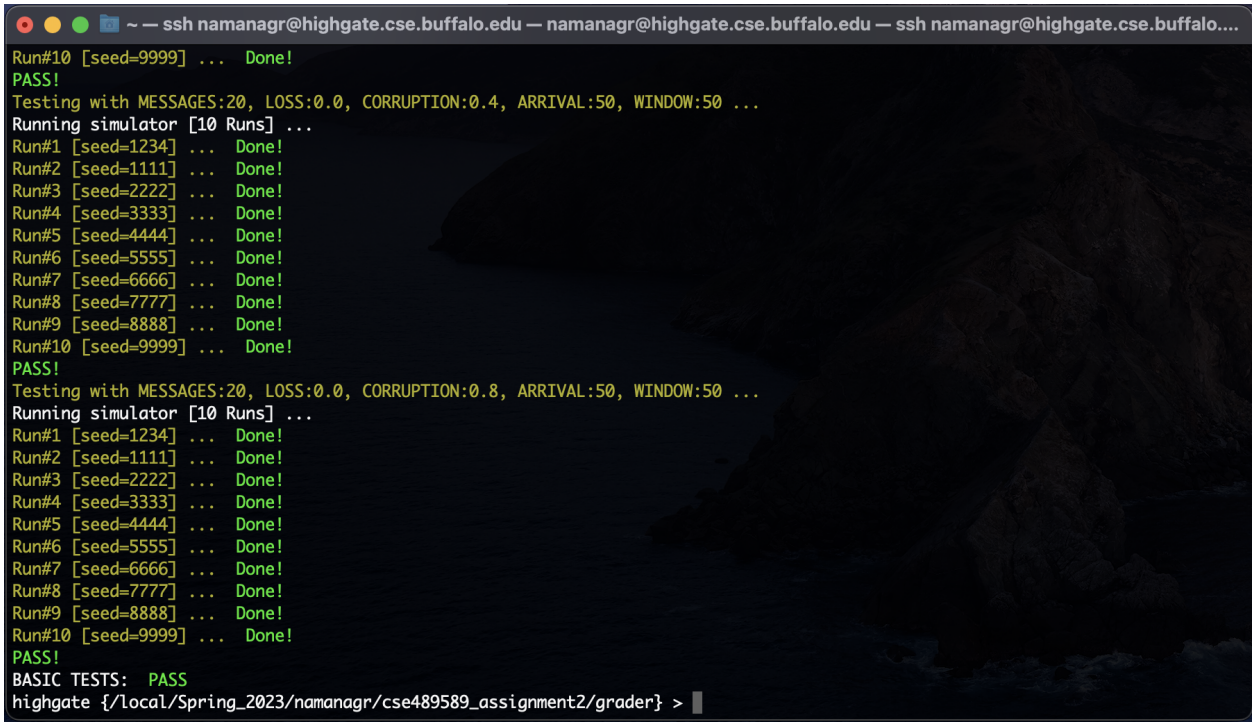
[5.0] ABT

(Put screenshots of the grader here…)



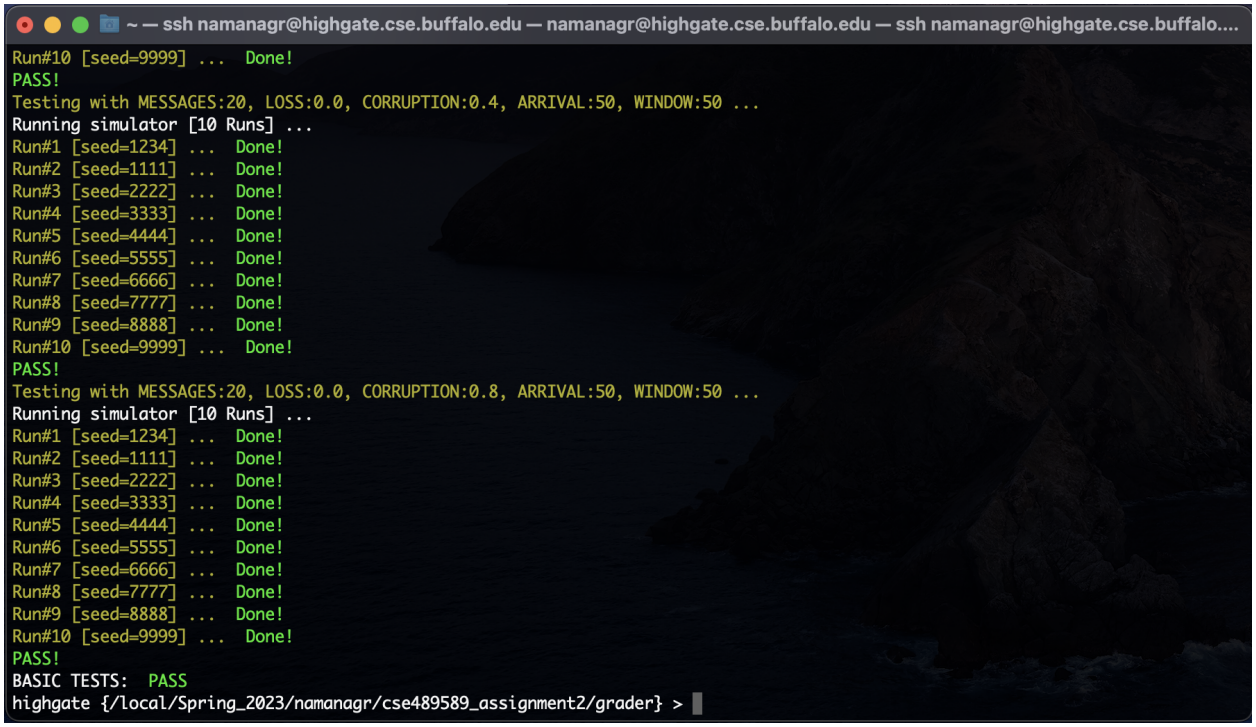
[12.0] GBN

(Put screenshots of the grader here…)



[18.0] SR

(Put screenshots of the grader here…)

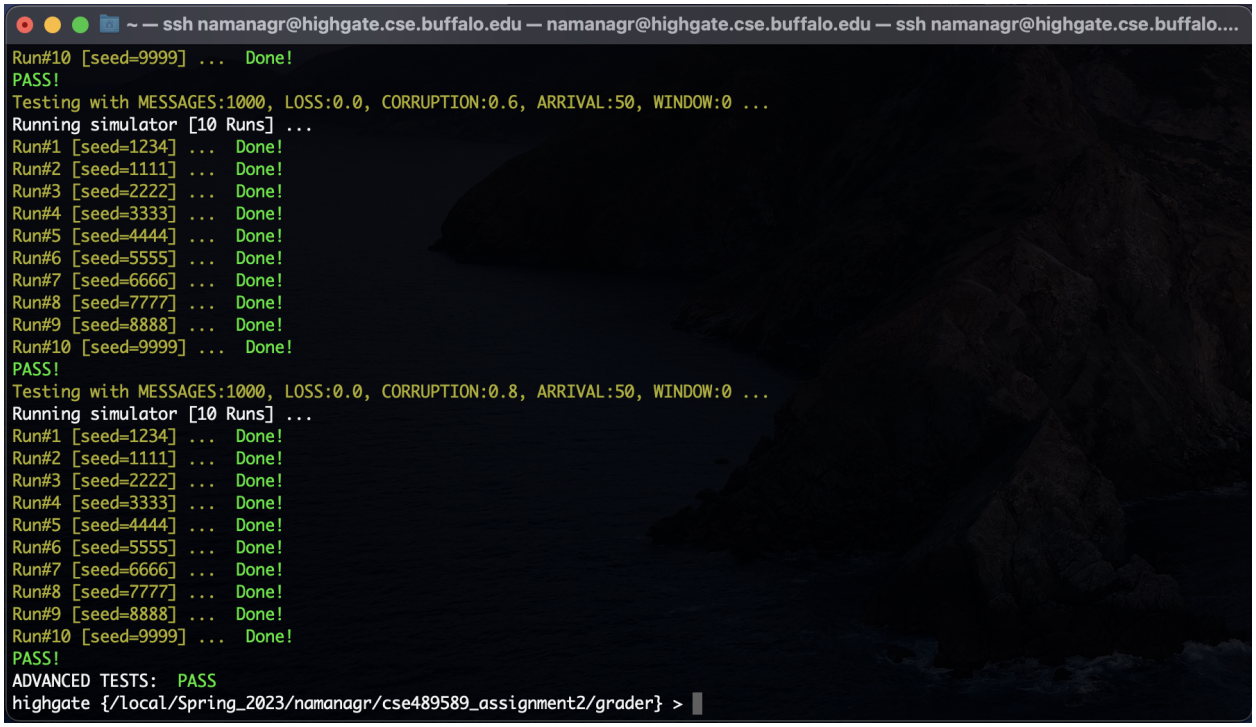


[No further grading for the protocol that fails a BASIC test.]

5 - ADVANCED Tests

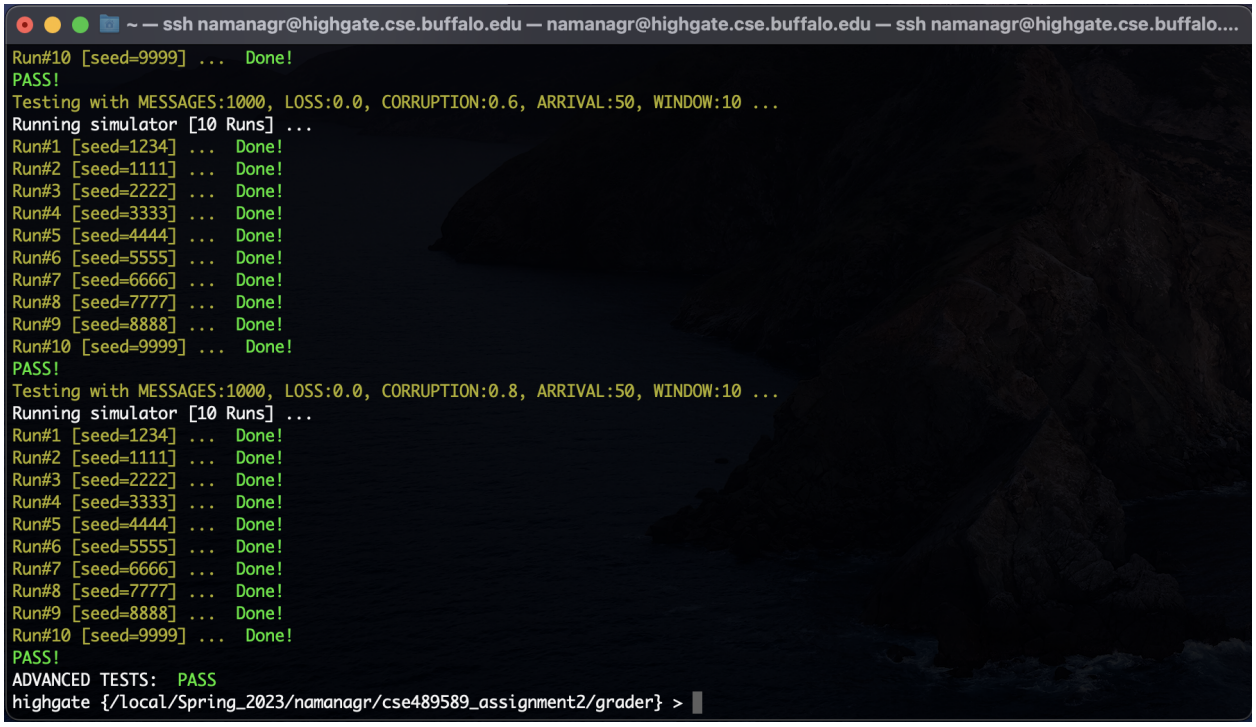
[5.0] ABT

(Put screenshots of the grader here…)



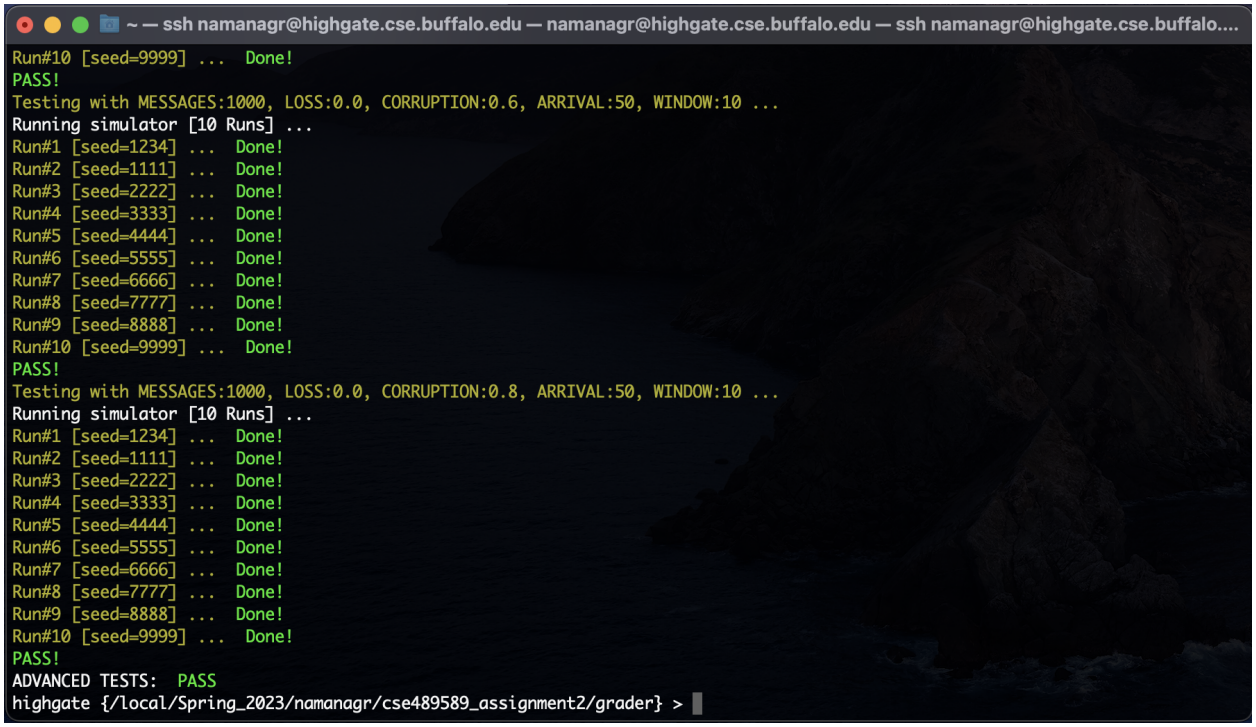
[10.0] GBN

(Put screenshots of the grader here…)



[20.0] SR

(Put screenshots of the grader here…)



6 - 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?)

(Observations, Analysis, Report, Implementation details, start on next page onwards)

Timeout Timeunits Experiment & Analysis

- For my protocol implementation, we selected a timeout scheme based on the requirements and characteristics of each protocol. The timeout duration varies across different protocols, such as Alternative Bit, Go Back N, and Selective Repeat, as their retransmission and buffer management techniques differ. To evaluate the impact of varying loss probabilities on the performance of these protocols, we conducted an analysis of different timeout durations for ABT, GBN, and SR as part of this assignment.

- In the case of Alternative Bit Transfer (ABT), a timeout value of approximately 10 was suggested as a reference in the project handout. Experiments were conducted with timeout as {10, 15, 20, 25}. We found that a timeout value of 15 was more suitable for the implementation. This decision was made to allow for some delays and avoid unnecessary retransmissions caused by a timeout that is too close to the expected round trip time (RTT).

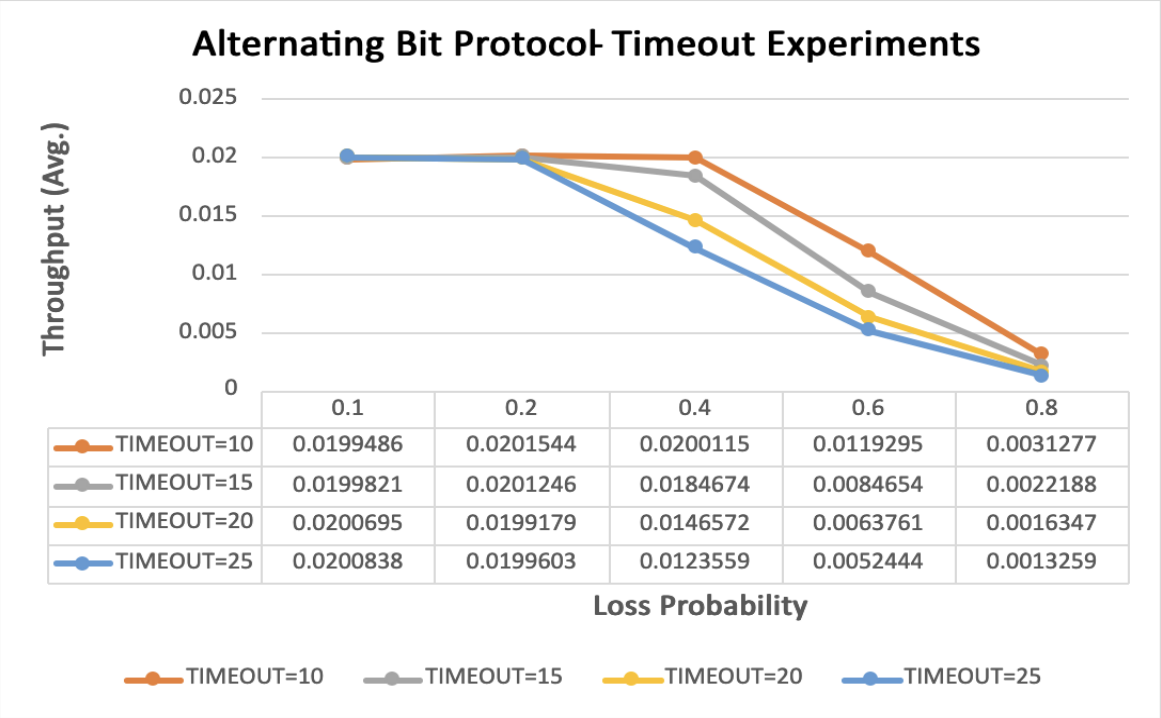
- For Go Back N (GBN), where multiple packets can be in transit at the same time and the round trip time (RTT) can vary, we used a trial-and-error approach to determine the optimal timeout value. Initially, we set the timeout to 10, but this resulted in a high number of retransmissions due to premature timeouts. To strike a good balance between excessive retransmissions and waiting too long, experiments were conducted with timeout as {20, 30, 40, 50, 100}, and we found that 30 timeunits (~3RTTs) gave reasonably good performance.

- For Selective Repeat, similar process was repeated to determine the optimal timeout value with timeout as {15, 20, 25, 30}. SR is optimized over GBN as SR only retransmits the lost/corrupt sequence and no other correctly ack'd sequences and it also buffers data at the receiver end. We observed that a timeout of 15 timeunits gave reasonable good performance.

Note: All Timeout experiments have been carried out using criteria from section 6.1 of performance comparison - Experiment 1. (i.e., a total number of 1000 messages to be sent by entity A, a mean time of 50 between message arrivals (from A's Layer 5), and a corruption probability of 0.2, while varying window size, loss probability and Timeout timeunits).

a) Alternating-Bit Protocol (ABT)

The **average throughput observed** for the **Alternating-Bit** Protocol using different timeouts and loss probabilities is represented in the figure below.

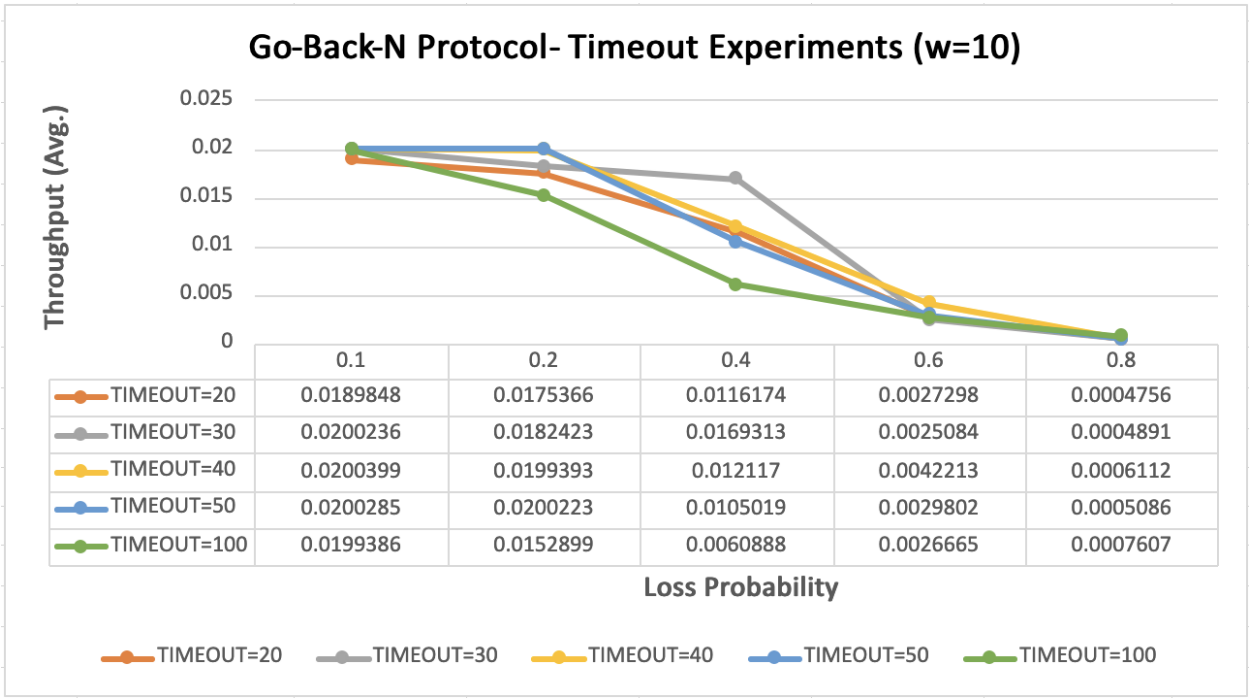


	0.1	0.2	0.4	0.6	0.8
TIMEOUT=10	0.0199486	0.0201544	0.0200115	0.0119295	0.0031277
TIMEOUT=15	0.0199821	0.0201246	0.0184674	0.0084654	0.0022188
TIMEOUT=20	0.0200695	0.0199179	0.0146572	0.0063761	0.0016347
TIMEOUT=25	0.0200838	0.0199603	0.0123559	0.0052444	0.0013259

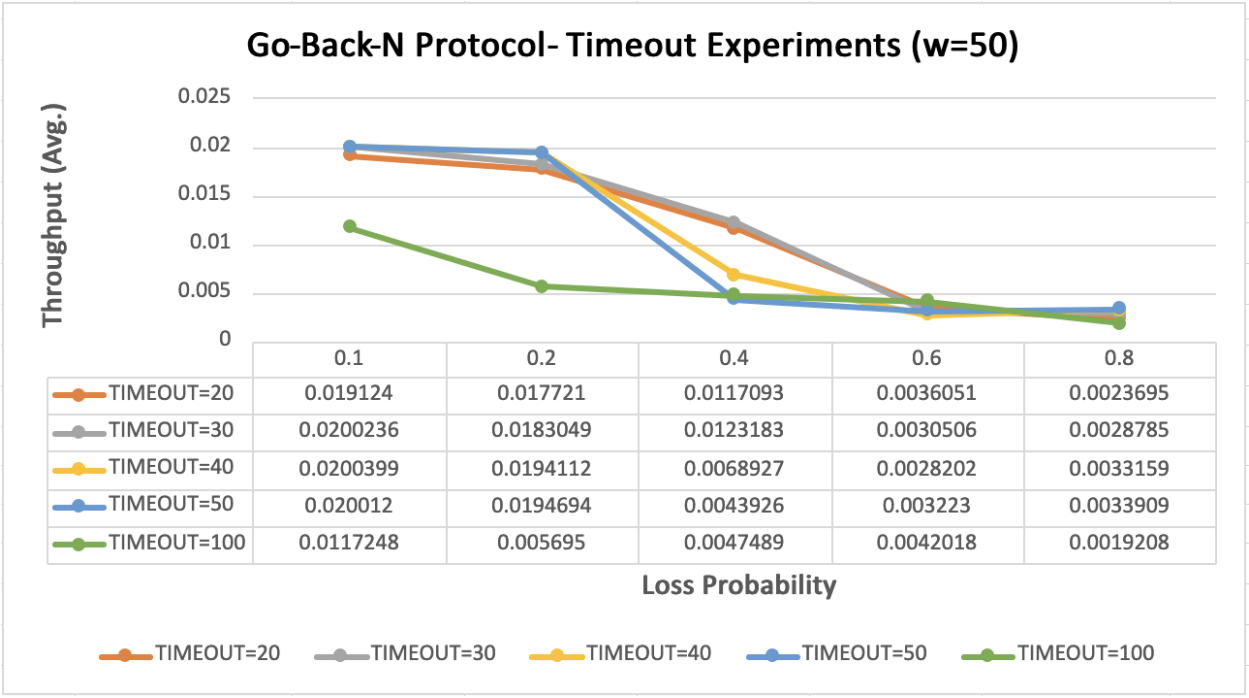
Analysis: We observe, the timeout value 10,15 are that yielded the highest average throughput, as they closer to an average round trip time of 10. Since the Alternative Bit protocol employs a stop-and-wait approach, it is preferable to minimize the amount of time spent waiting after the expected round trip time, although choosing 10 could be insufficient. **Therefore, a timeout value of 15 was selected for generating further observations of Alternating Bit Protocol.**

b) Go-Back-N Protocol (GBN)

The **average throughput observed** for the **Go-Back-N** Protocol using different timeouts and loss probabilities; with **window_size= 10 and 50**; are represented in two the figures below.



	0.1	0.2	0.4	0.6	0.8
TIMEOUT=20	0.0189848	0.0175366	0.0116174	0.0027298	0.0004756
TIMEOUT=30	0.0200236	0.0182423	0.0169313	0.0025084	0.0004891
TIMEOUT=40	0.0200399	0.0199393	0.012117	0.0042213	0.0006112
TIMEOUT=50	0.0200285	0.0200223	0.0105019	0.0029802	0.0005086
TIMEOUT=100	0.0199386	0.0152899	0.0060888	0.0026665	0.0007607

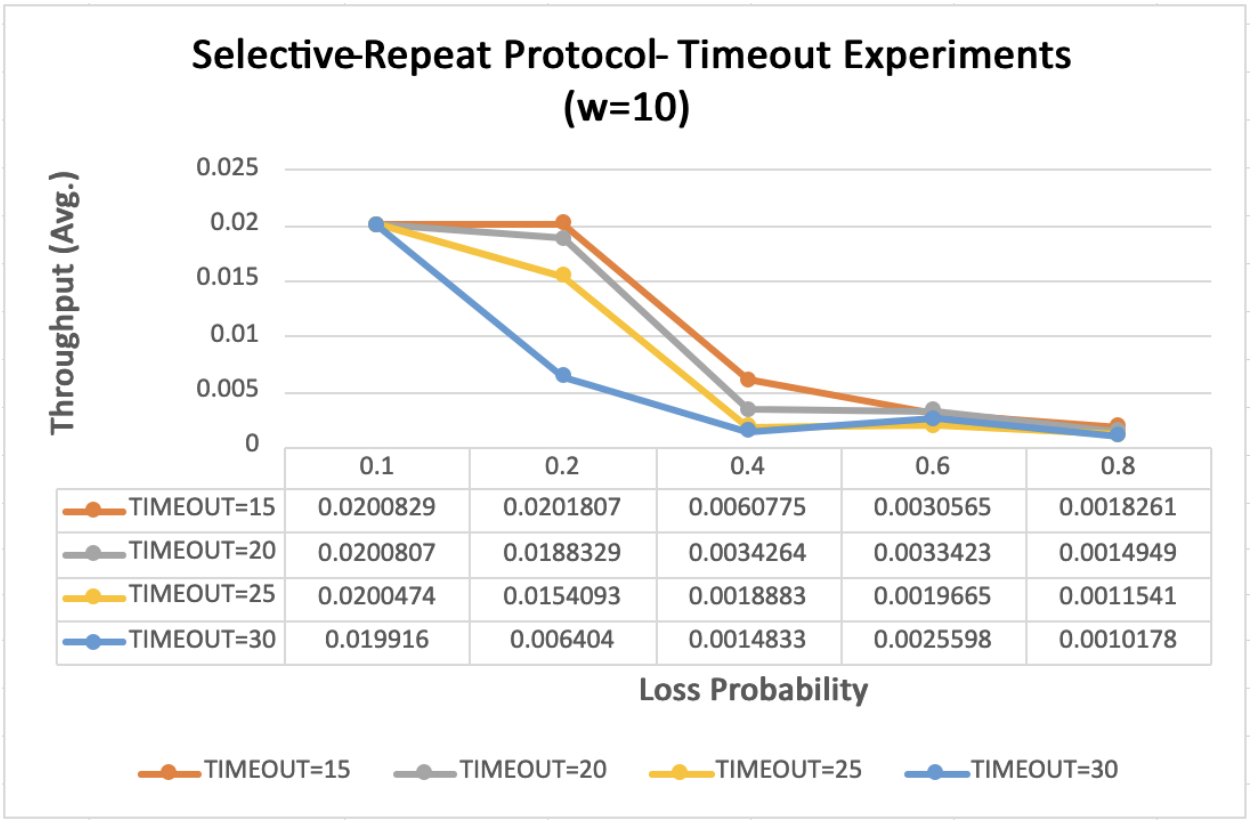


	0.1	0.2	0.4	0.6	0.8
TIMEOUT=20	0.019124	0.017721	0.0117093	0.0036051	0.0023695
TIMEOUT=30	0.0200236	0.0183049	0.0123183	0.0030506	0.0028785
TIMEOUT=40	0.0200399	0.0194112	0.0068927	0.0028202	0.0033159
TIMEOUT=50	0.020012	0.0194694	0.0043926	0.003223	0.0033909
TIMEOUT=100	0.0117248	0.005695	0.0047489	0.0042018	0.0019208

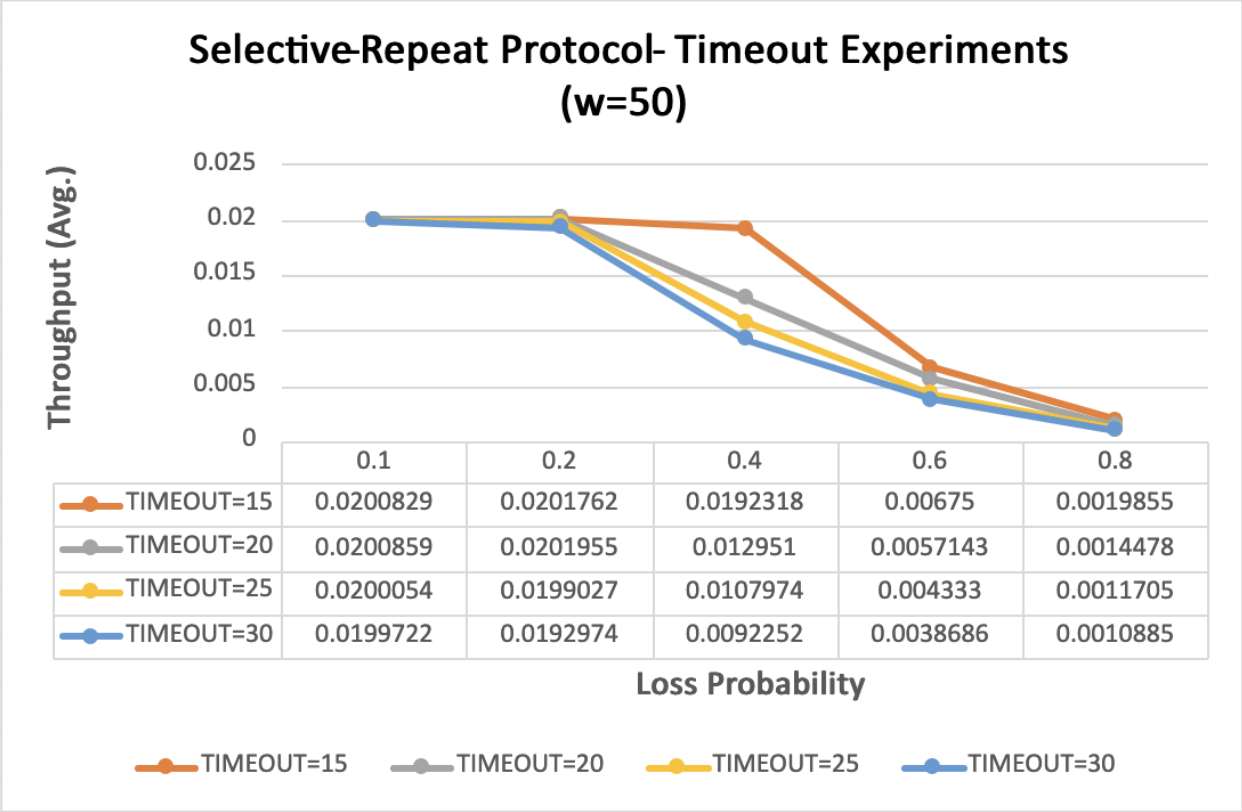
<u>Analysis</u>: Here we observe that **the timeout value that yielded the highest average throughput is closer to 30 timeunits (~ 3 avg. RTTs).** Since the Go-Bak-N protocol returns to the beginning of the window and potentially resends successfully received packets to the receiving entity, it is expected that Go-Back-N will underperform in cases of high loss or corruption. This trend amplifies with larger window sizes. **Therefore, a timeout value of 30 was selected for generating further observations of Go-Back-N Protocol as this provides a relatively best performance as observed in the above figures and also allows delayed packets enough time to be received, potentially saving few unnecessary transmissions.**

c) Selective-Repeat Protocol (SR)

The **average throughput observed** for the **Selective-Repeat Protocol** using different timeouts and loss probabilities; with **window_size= 10 and 50**; are represented in two the figures below.



	0.1	0.2	0.4	0.6	0.8
TIMEOUT=15	0.0200829	0.0201807	0.0060775	0.0030565	0.0018261
TIMEOUT=20	0.0200807	0.0188329	0.0034264	0.0033423	0.0014949
TIMEOUT=25	0.0200474	0.0154093	0.0018883	0.0019665	0.0011541
TIMEOUT=30	0.019916	0.006404	0.0014833	0.0025598	0.0010178



	0.1	0.2	0.4	0.6	0.8
TIMEOUT=15	0.0200829	0.0201762	0.0192318	0.00675	0.0019855
TIMEOUT=20	0.0200859	0.0201955	0.012951	0.0057143	0.0014478
TIMEOUT=25	0.0200054	0.0199027	0.0107974	0.004333	0.0011705
TIMEOUT=30	0.0199722	0.0192974	0.0092252	0.0038686	0.0010885

Analysis: We observe that a timeout value = 15 timeunits performs reasonably well for both window sizes 10 and 50. As Selective Repeat buffers data at the receiver end, it is preferable to select a timeout duration that is closer to the expected round trip time. **Therefore, a timeout value of 15 was selected for generating further observations of Selective-Repeat Protocol.**

Selective-Repeat Implementation (Data Structures):

The variables and data structures used in this protocol file are copied below.

```
/* generic structure for entity (A/B) state */
struct entity
{
   int seq; // sequence number -> alt bit protocol styles (usage values: 0,1)
   int ack; // acknowledgment number
   int windowSize; // window size
   int baseIndex; // base index of window
};
struct entity entity_A; // A
struct entity entity_B; // B

/* custom structure msgs with isAckd flag */
struct buffer_msg
{
   char data[20];
   bool isAckd;
};
/* custom structure for logical timers */
struct seqtimers
{
   int seq; //seq num
   float time; //time of creation
};

// buffer to store incoming msgs from layer 5 for A
vector<struct buffer_msg> messagesBufferA;
// queue for logical seqtimers used by A
queue<struct seqtimers> seqTimersQueueA;
// map for packets in receiving window of B
map<int, struct pkt> receivedPacketsMapB;
```

Selective-Repeat Implementation Details:

- A queue is maintained with elements of type struct seqtimer, i.e, a struct of a integer sequence number and float time of creation (simulation time).

- Incoming data from layer5 for A, is first parsed to type struct buffer_msg with isAckd flag initally set as false, which is set to true later when the corresponding ack is received. This buffer_msg is pushed to vector messagesBufferA and then A performs its checks to see if the next seq lies and in the window and the packet can be sent.

- When a packet is passed to layer3 by A, a custom sequence timer of type struct seqtimers is created with the sequence number and the time as the then current simulation time and is added to queue seqTimersQueueA. If this is only seqtimer in queue, the hardware timer is started.

- When A_input acknowledges ack packet which corresponding to the start of the window, we check recursively if the next seq in window is acknowledged already, and the corresponding seqtimer are popped from the front of queue seqTimersQueueA.

- In case of A's timer interrupt, only the corresponding seqnum is sent for retransmission.

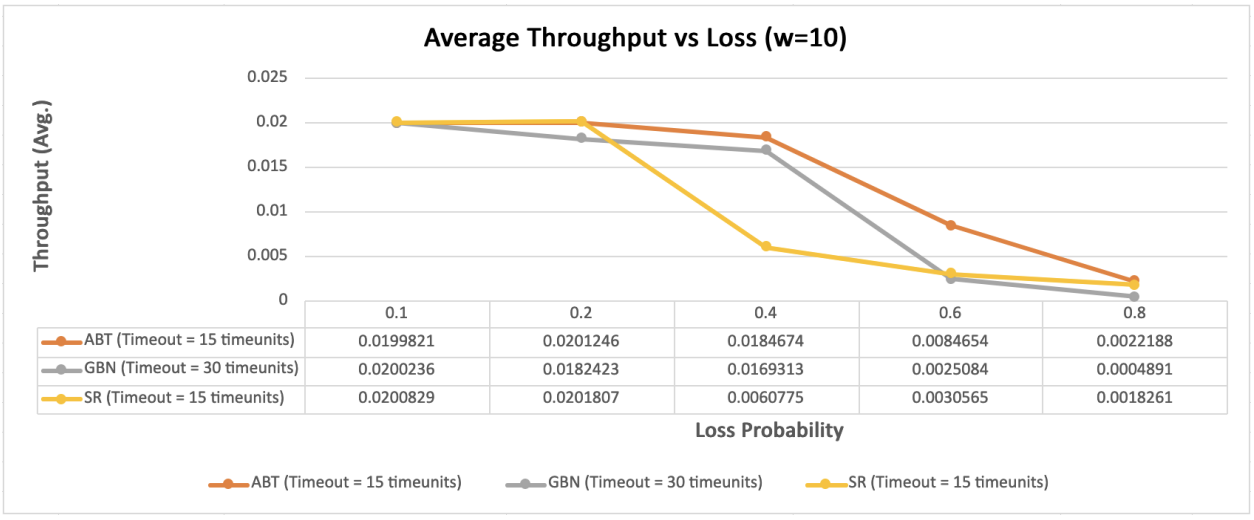
- In case of B_input, an ack is sent back for every valid packet received. If the packet's seqnum is the expected base index of receiving window, then it's payload immediately passed to layer5 or else it is added to a map receivedPacketsMapB with seqnum as key. Similar to as mentioned before, these packets are recursively cleared from the map for all packet seqnums in order and waiting, and then the payload is passed to layer5 in expected order.

6.1 Performance Comparison

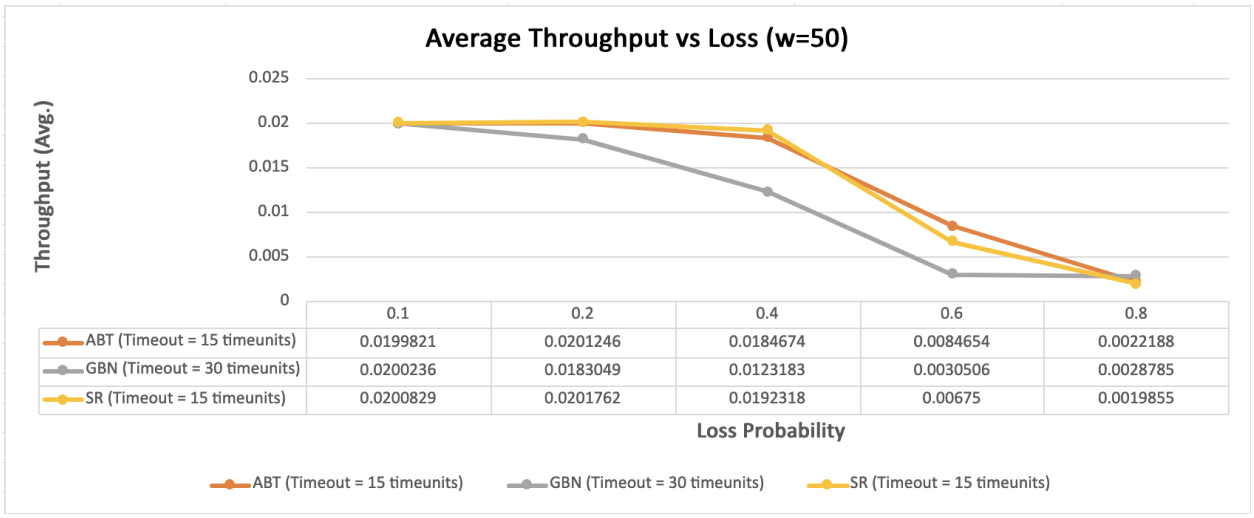
Experiment - 1:

Experiment 1 is conducted on constant values of 1000 messages to be sent by entity A, a mean time of 50 between message arrivals (from A's layer5) and a corruption probability of 0.2.

The experiments are done on the basis of different loss probabilities of 0.1, 0.2, 0.4, 0.6 and 0.8 and window size of 10 and 50 (for GBN and SR), comparing the 3 protocols (ABT, GBN and SR).



	0.1	0.2	0.4	0.6	0.8
ABT (Timeout = 15 timeunits)	0.0199821	0.0201246	0.0184674	0.0084654	0.0022188
GBN (Timeout = 30 timeunits)	0.0200236	0.0182423	0.0169313	0.0025084	0.0004891
SR (Timeout = 15 timeunits)	0.0200829	0.0201807	0.0060775	0.0030565	0.0018261



	0.1	0.2	0.4	0.6	0.8
ABT (Timeout = 15 timeunits)	0.0199821	0.0201246	0.0184674	0.0084654	0.0022188
GBN (Timeout = 30 timeunits)	0.0200236	0.0183049	0.0123183	0.0030506	0.0028785
SR (Timeout = 15 timeunits)	0.0200829	0.0201762	0.0192318	0.00675	0.0019855

Analysis & Observation:

- As the probability of packet loss increases, the number of packets that fail to reach the receiver also increases, leading to a decrease in throughput across all protocols.

- Notably, a significant drop in throughput can be observed after a loss probability of 0.4. As the loss probability continues to increase, we observe that SR starts to perform closer to ABT, which can be attributed to its practice of only retransmitting unacknowledged packets.

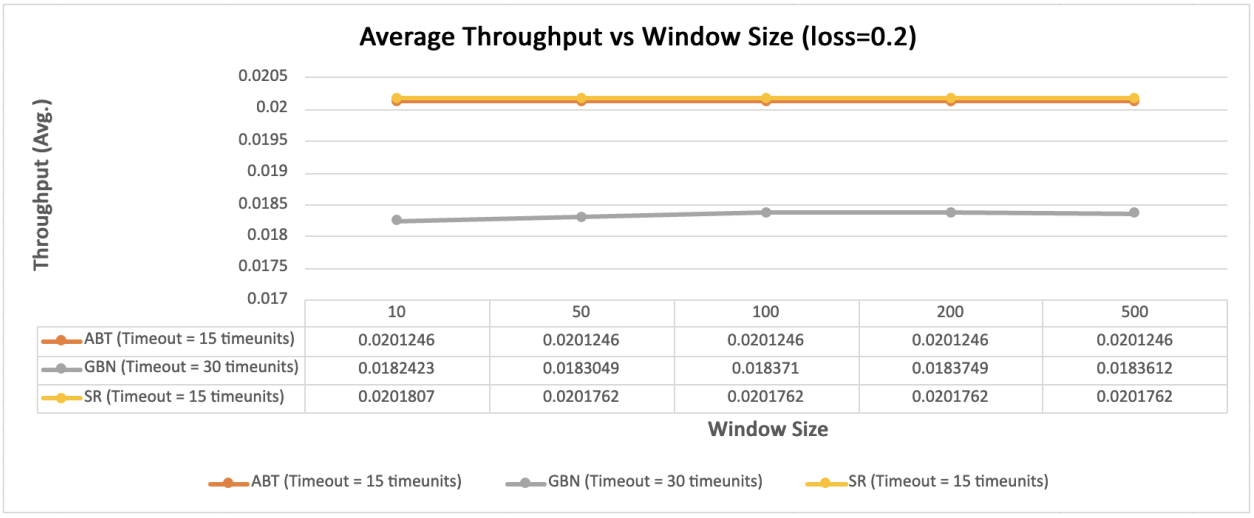
- For window size 10, ABT is observed to perform with the best average throughput.

- As the window size is increased to 50, GBN observes significant drops in throughput as the protocol requires retransmission of packets from the base sequence of the window. SR seems to be the clear winner over GBT with large window sizes. (ABT is independent of window size)

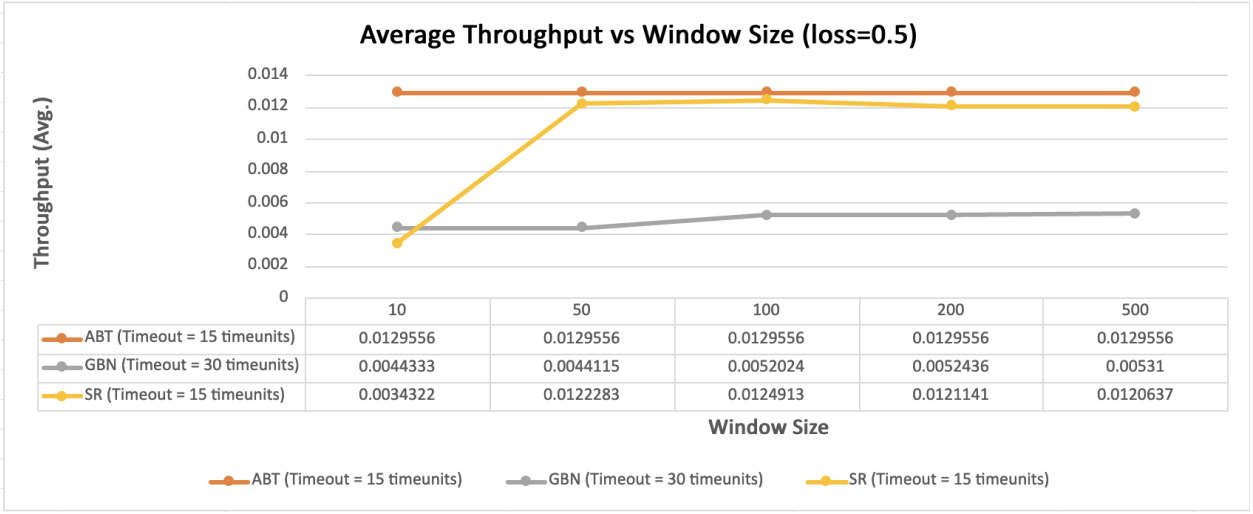
Experiment - 2:

Experiment 2 is conducted on constant values of 1000 messages to be sent by entity A, a mean time of 50 between message arrivals (from A's layer5) and a corruption probability of 0.2.

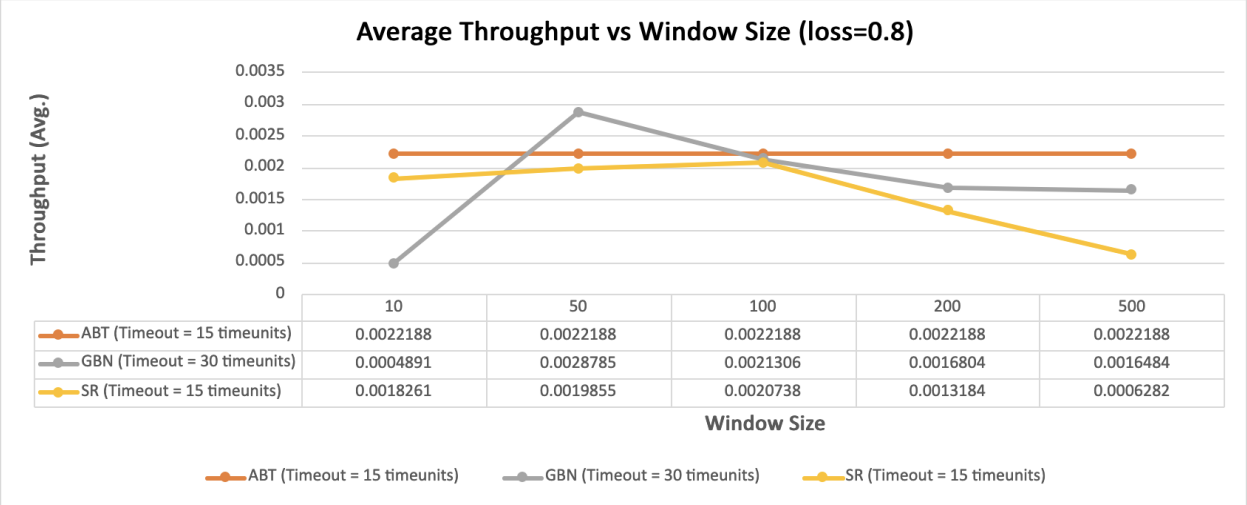
The experiments are done on the basis of different loss probabilities of 0.2, 0.5, 0.8 and window size of 10, 50, 100, 200, 500 (for GBN and SR), comparing the 3 protocols (ABT, GBN and SR).



	10	50	100	200	500
ABT (Timeout = 15 timeunits)	0.0201246	0.0201246	0.0201246	0.0201246	0.0201246
GBN (Timeout = 30 timeunits)	0.0182423	0.0183049	0.018371	0.0183749	0.0183612
SR (Timeout = 15 timeunits)	0.0201807	0.0201762	0.0201762	0.0201762	0.0201762



	10	50	100	200	500
ABT (Timeout = 15 timeunits)	0.0129556	0.0129556	0.0129556	0.0129556	0.0129556
GBN (Timeout = 30 timeunits)	0.0044333	0.0044115	0.0052024	0.0052436	0.00531
SR (Timeout = 15 timeunits)	0.0034322	0.0122283	0.0124913	0.0121141	0.0120637



Analysis & Observation:

- At loss probability = 0.2, the varying of window size does not have any significant impact on the avg. throughput of both GBN (~0.018) and SR (~0.02). ABT (~0.02) is independent of window size. This is expected as the loss probability here is relatively low.
- As we increase the loss probability to 0.5, the heaviest impact is on GBN's performance (~0.004) as it probably suffering from retransmissions of large windows. Apart from an exception at w=10, the performance of SR (~0.012) is much better than GBN's as SR is optimized to be able to only retransmit lost/corrupt packets instead of entire windows. ABT (~0.012) is independent of window size and performs reasonably well in our experiments.
- Finally as the loss probability is increased to 0.8, we observe that the performance of all three protocols ABT (~0.002), GBN (~0.0018), SR (~0.0019) is heavily impacted. This is expected as all protocols suffer from retransmitting large amount of lost packets. GBN seems to outperform SR at w=50, which could be an exception or due to the the specific implementation and choice of timeouts. Otherwise, SR performs better than GBN, which is expected as it is only retransmitting the unack'd packets.

You have reached the end of the report. Thank You!
