EECE 7374 Programming Assignment 2
Reliable Transport Protocols

Name: Germain Mucyo
NUID: 002301781
Email: mucyo.g@northeastern.edu

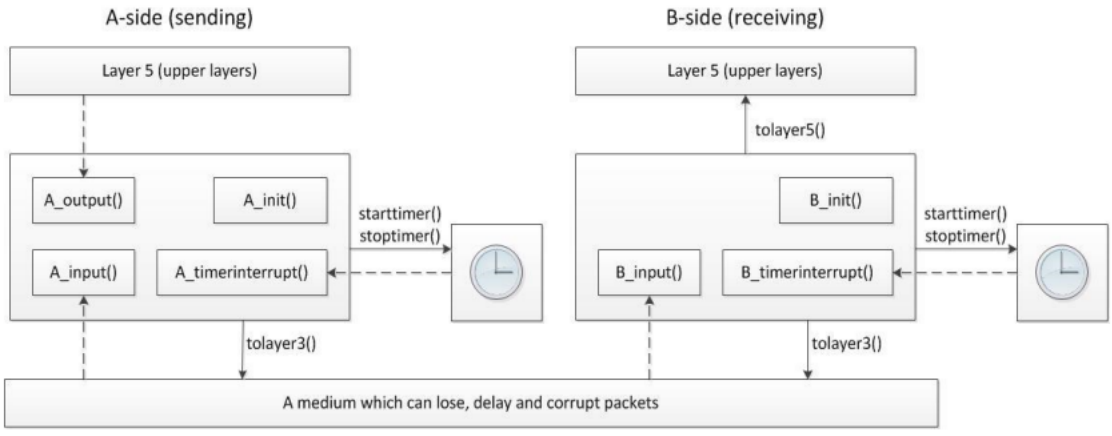
Overview

In this programming assignment, I used C++ to write the codes that send and receive packets to implement a simple reliable data transfer protocol. Each stage will highlight my experience challenges I faced and how I overcame some challenges to make my implementation.

I used office hours, lecture slides, readings, research papers, and discussions with classmates on Piazza and in-person and more personal efforts to evaluate my skills to understand and achieve the target of this programming assignment fully.

Variables and data structures of my codes, before proceeding in building the prototype functions, I refreshed my mind on some programming skills through geeksforgeeks and julialang.org[i] as well as the book used throughout this course Computer Networking: A top-down approach[ii]. After that, on each protocol, I added utility functions to create data and acknowledgment packets, calculate checksums (checksumming()), alter the sequence number (alter_seqnum()), etc. Also, the timeout schemes used, which will be explained in this report.

The implementation routines; I didn't change, as the description highlighted. Below is the flowchart of the structure of the environment.



Testing: In this lab, I have run the script for testing to analyze the behavior of my protocols as per the lab description. I have successfully run the three test experiment; ./sanity_tests -h $ ./basic_tests -h and $ ./advanced_tests -h with the consideration of /run_experiments -h to all tests, successfully all have passed the test.

Experiment 1

Timeout Scheme Description, ABT

I implemented a single timer to handle the acknowledgment of packets. The simplicity of this protocol lies in its stop-and-wait approach, where one packet is sent at a time, and the sender waits for an acknowledgment before sending the next packet.

Throughput analysis, ABT

Below I will share the findings I have found while analyzing the throughput according to the setting of the environment. Seed as a random number I used 3.

Using -s 3; -m 20; -t 50, -v 2 below I present the test cases according to loss and corruption variation.

Test Cases

Test Case	Loss (-l)	Corruption (-c)	Throughput packets/time units
1	0.1	0.0	0.001672
2	0.2	0.0	0.001741
3	0.4	0.0	0.000805
4	0.6	0.0	0.001005
5	0.8	0.0	0.001013
6	0.0	0.1	0.002819
7	0.0	0.2	0.002551
8	0.0	0.4	0.002658
9	0.0	0.6	0.002004
10	0.0	0.8	0.000996

Observation: These findings demonstrate the inefficiency of the ABT protocol, which was anticipated. The throughput decreases as packet loss increases, and the level of corruption remains relatively stable. I attempted to adjust the timeout value, but the results were nearly identical (refer to the code). Overall, the nature of the ABT protocol leads to packet loss, causing subsequent transmission delays and resulting in low throughput.

Timeout Scheme Description. GBN

In the Go-Back-N (GBN) protocol, I build up this protocol from abt, in this case, I used single timer for the unacknowledged packet; if it expires, all unacknowledged packets in the window are retransmitted. I used lecture slides to understand the logic, all tests passed and after that, I ran the experiment; *-s 3; -m 20; -w10 -t 50*, -v 2 as experiment setup. Below are different test case results.

GBN with Window 10

Test Case	Loss (-l)	Corruption (-c)	Throughput
1	0.1	0.0	0.015269
2	0.2	0.0	0.013800
3	0.4	0.0	0.003920
4	0.6	0.0	0.000901
5	0.8	0.0	0.000813

Timeout Scheme Description. SR

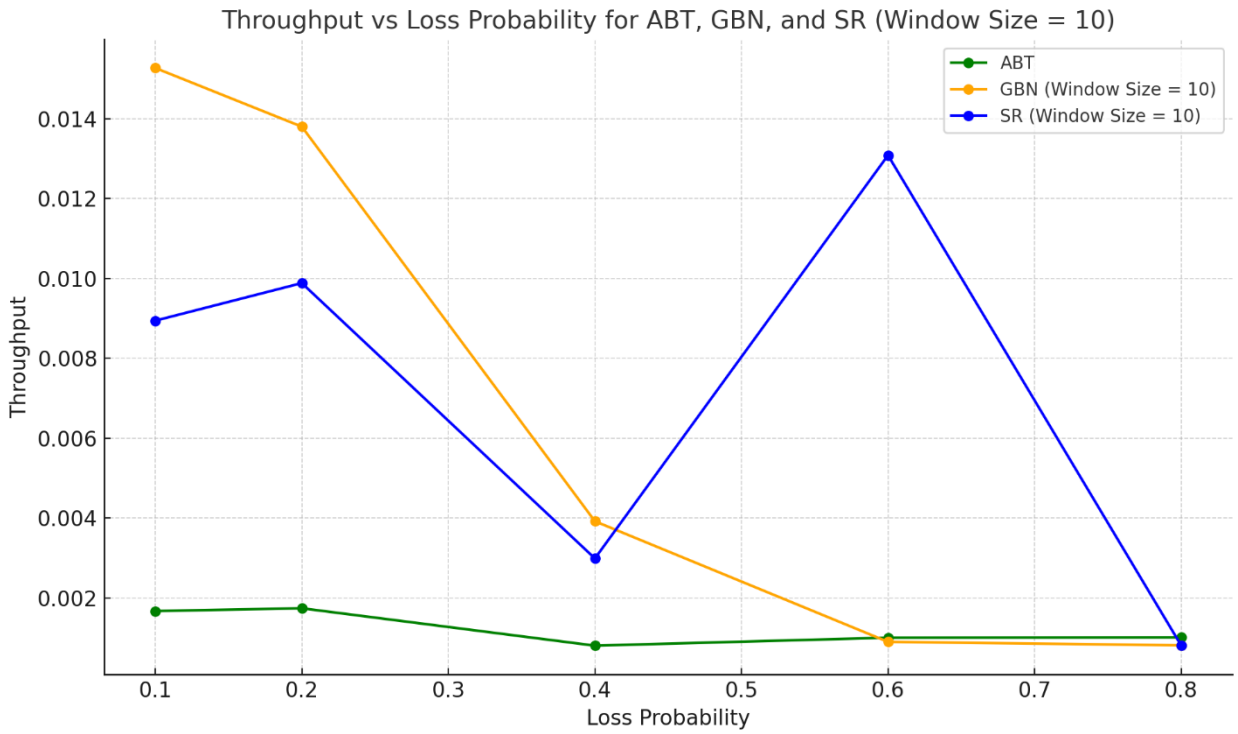
I used the sender_timers[] array to build several logical timers. The get_sim_time() function provided the current simulation time for comparison, and my array recorded the expiration timings of every packet in the sender window. I iterated across the window, verifying each packet's logical timer to see if it had expired when the timer interrupt (A_timerinterrupt()) was triggered. Individual packets that needed to be retransmitted were sent again, and their clocks were reset. This method preserves SR's need for selected retransmissions while avoiding the inefficiencies of GBN as explained earlier.

To be able to achieve this experiment I referred to RFC 793 in sections 3.7 to 4.2[iii] which explains the logic timers and other concepts related, class slides and readings, with my programming skills, plus more practice that could work and fail until I was able to pass all tests and run the experiment on different test cases as presented below.

To complete the 1st experiment, I ran the experiment; *-s 3; -m 20; -w10 -t 50*, -v 2 as experiment setup, on different loss probability, and corruption kept on 0.0. Below are different test case results.

sr with Window 10

Test Case	Loss (-l)	Corruption (-c)	Throughput
1	0.1	0.0	0.008942
2	0.2	0.0	0.009883
3	0.4	0.0	0.002989
4	0.6	0.0	0.013083
5	0.8	0.0	0.000813



Observation; The experiment demonstrates the performance of the three methods under various loss probabilities. Although ABT is straight forward (details explained above), its stop-and-wait design results in constantly low throughput, making it inappropriate for high-loss situations. Due to retransmissions, GBN suffers at larger loss rates but performs well in low-loss settings(). By retransmitting only dropped packets, SR effectively manages loss and maintains a greater throughput.

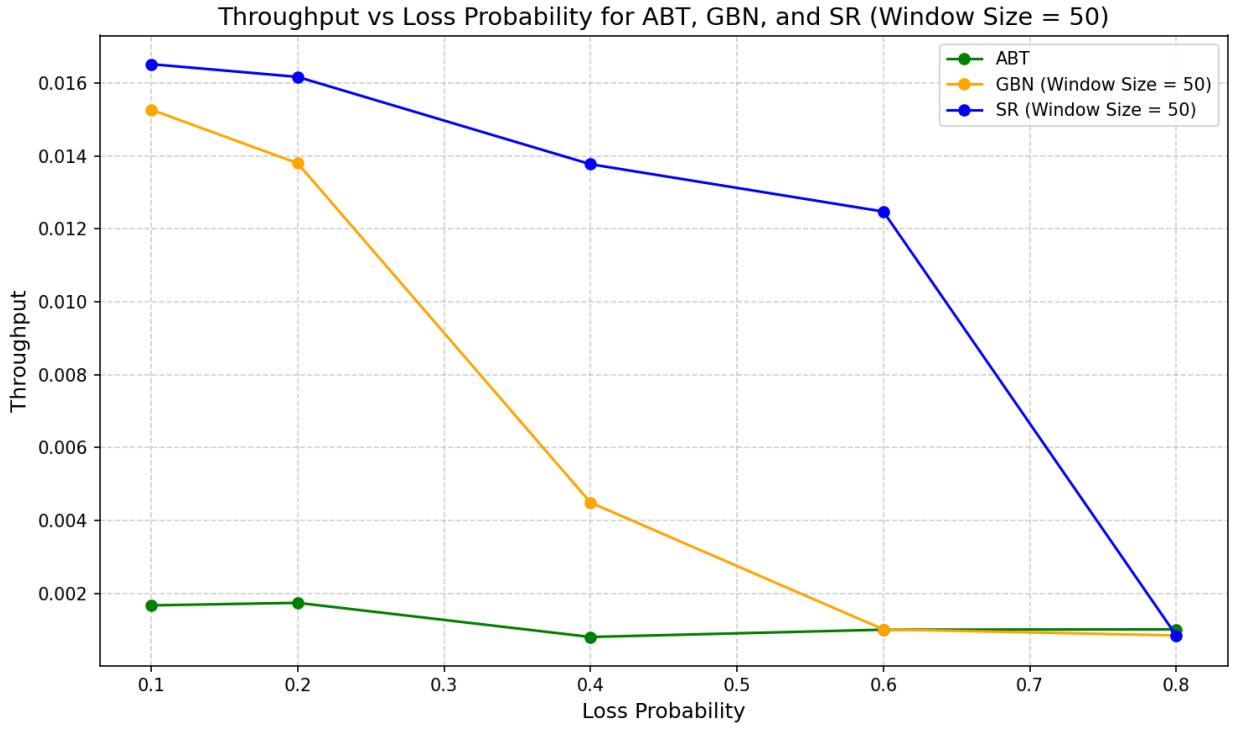
Repetition with window size of 50;

Selective repeat with Window 50.

Test Case	Loss (-l)	Corruption (-c)	Throughput
1	0.1	0.0	0.016517
2	0.2	0.0	0.016168
3	0.4	0.0	0.013773
4	0.6	0.0	0.012474
5	0.8	0.0	0.000847

GBN with Window 50

Test Case	Loss (-l)	Corruption (-c)	Throughput
1	0.1	0.0	0.015269
2	0.2	0.0	0.013800
3	0.4	0.0	0.004492
4	0.6	0.0	0.001013
5	0.8	0.0	0.000847

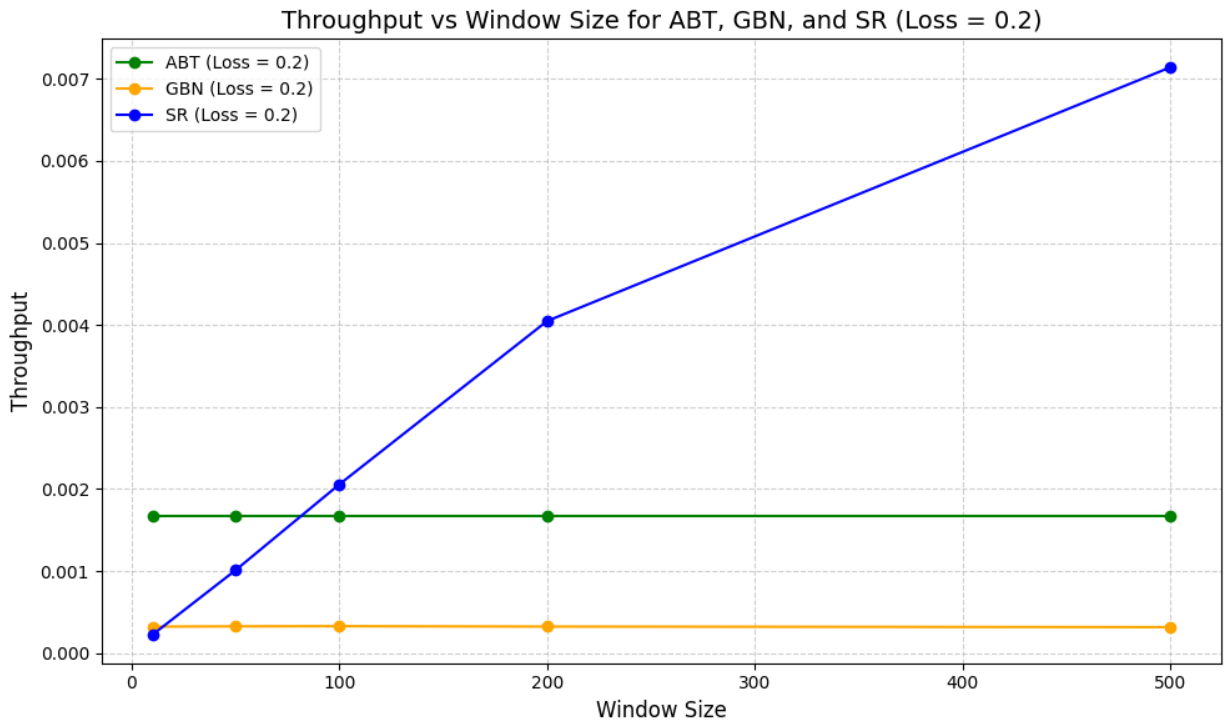


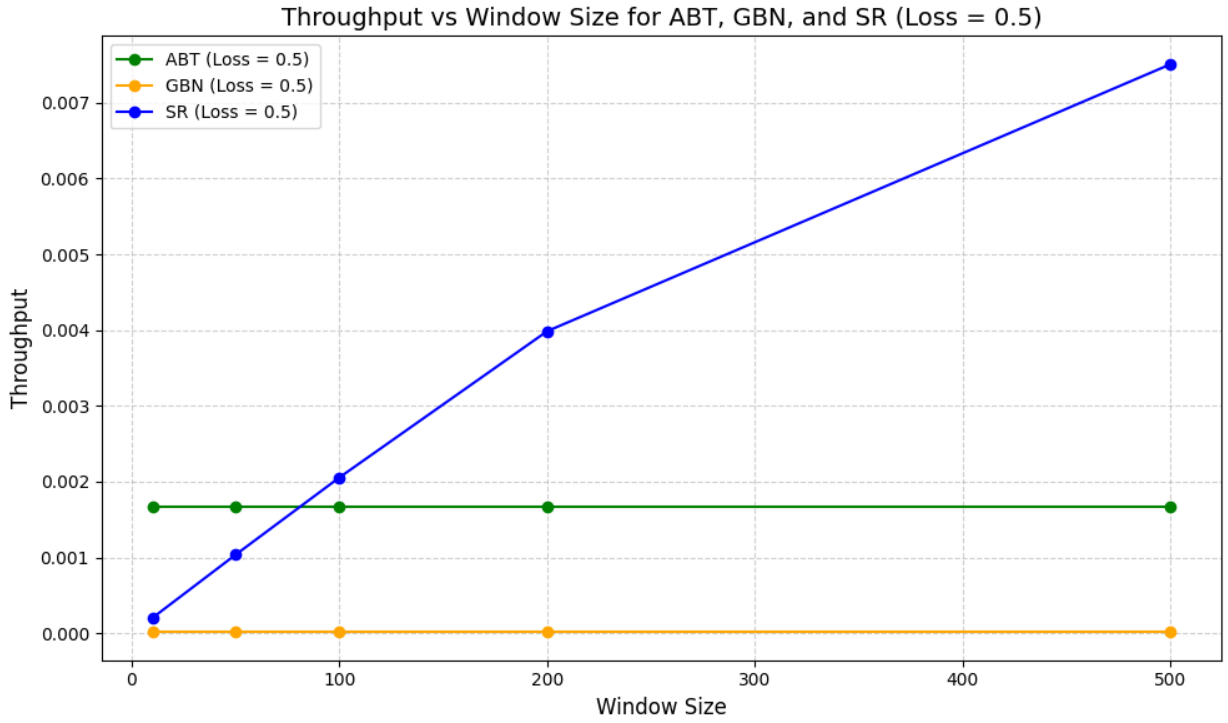
Observation: After considering window 50, ABT and GBN was not significantly impacted but SR was. In conclusion, SR is the most effective protocol for lossy networks and it meets my expectations, because of its better throughput and ability to handle losses and out-of-order packets, and even gets better throughput as the window size increases.

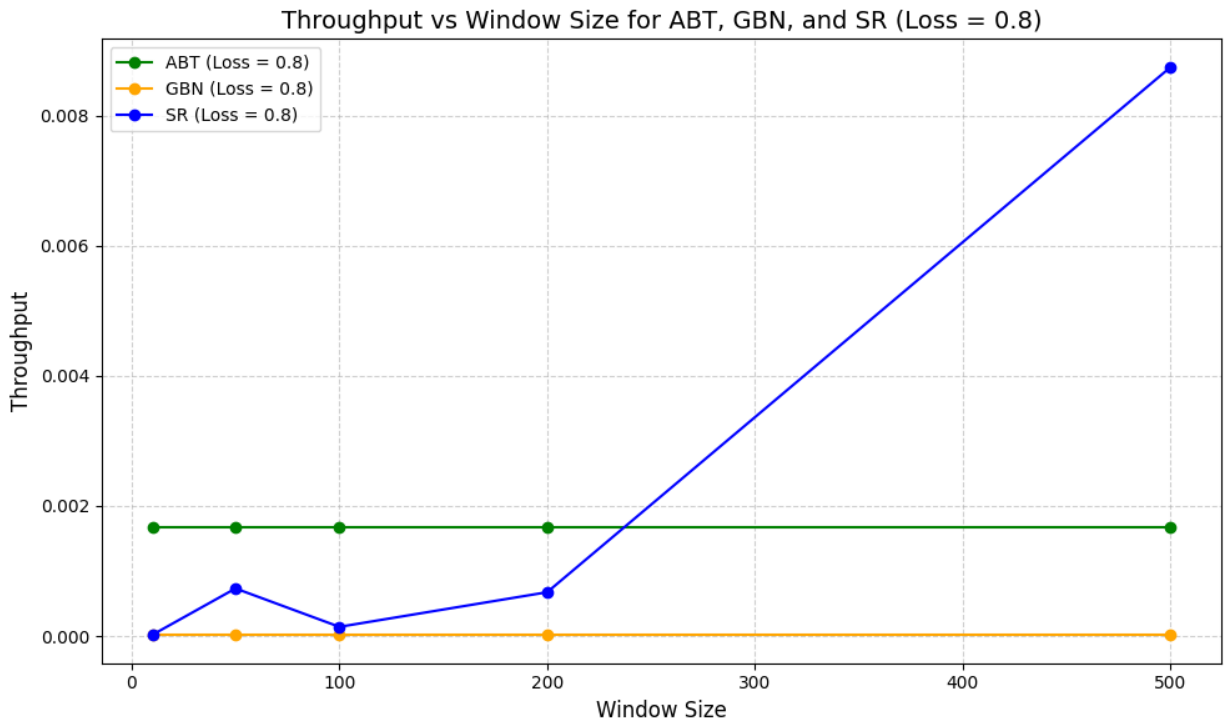
Experiment 2

Protocol	Loss	Window 10	Window 50	Window 100	Window 200	Window 500
GBN	0.2	0.000322	0.000326	0.000328	0.000323	0.000316
SR	0.2	0.000223	0.001010	0.002057	0.004048	0.007141
GBN	0.5	0.000020	0.000020	0.000020	0.000020	0.000020
SR	0.5	0.000203	0.001036	0.002056	0.003986	0.007508
GBN	0.8	0.000020	0.000020	0.000020	0.000020	0.000020
SR	0.8	0.000020	0.000732	0.000142	0.000674	0.008740

In this experiment, at the **Selective-Repeat (SR)** protocol with a window size of 500 at timeout of 50, I noticed that packets were continuously dropped, causing the terminal to hang. I had to manually terminate the simulation, after adjusting timeout to 100 under a loss probability of 0.2. The throughput achieved was **0.007141**, then I did the same to -l 0.5, however -l 0.8 did not show any issue.


Throughput vs Window Size for ABT, GBN, and SR (Loss = 0.2)
ABT (Loss = 0.2)
GBN (Loss = 0.2)
SR (Loss = 0.2)
Throughput
Window Size
0.000
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0
100
200
300
400
500


Throughput vs Window Size for ABT, GBN, and SR (Loss = 0.5)
ABT (Loss = 0.5)
GBN (Loss = 0.5)
SR (Loss = 0.5)
Throughput
Window Size
0.000
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0
100
200
300
400
500



The findings from the second experiment demonstrate that the Selective Repeat protocol effectively manages packet losses and maximizes throughput as the window size increases. This highlights its capability to enhance network performance. In contrast, the graphs in experiment 2 showed that both ABT and GBN did not exhibit significant improvements due to their inherent limitations stated in the first experiment. Additionally, ABT, being a stop-and-wait protocol, does not fully utilize the available window size, while GBN retransmits all unacknowledged packets whenever just one packet is lost. As a result, the advantages of increasing the window size are minimal or negligible for these protocols.

Challenges I faced and how I overcame the situation;

- Continuous dropped packet causing the terminal to hang especially in SR implementation, I overcame this by adjusting timeout and applying different window sizes, in the end, there was no terminal hangout and I was able to measure the throughput.
- Failing to pass sanity and advanced test on GBN and SR a couple of times. I reached out during the office hours of TA, debugged, and found out that I was starting the timer while another timer was running, so I had to adjust A_timerinterrupt and timer logic, eventually, it worked and both tests passed after proper adjustment.
- Unsure of how to implement the buffer, reached out to TA, I set it to 1000 and it worked.

[i] Discourse. (2020, March 25). *Organizing code into functions—am I doing it right?* JuliaLang. https://discourse.julialang.org/t/organizing-code-into-functions-am-i-doing-it-right/35228/7

[ii] Kurose, J. F., & Ross, K. W. (2021). *Computer networking: A top-down approach* (8th ed.). Pearson Education

[iii] Postel, j. 1981. TCP, RFC 793, https://www.ietf.org/rfc/rfc793.txt