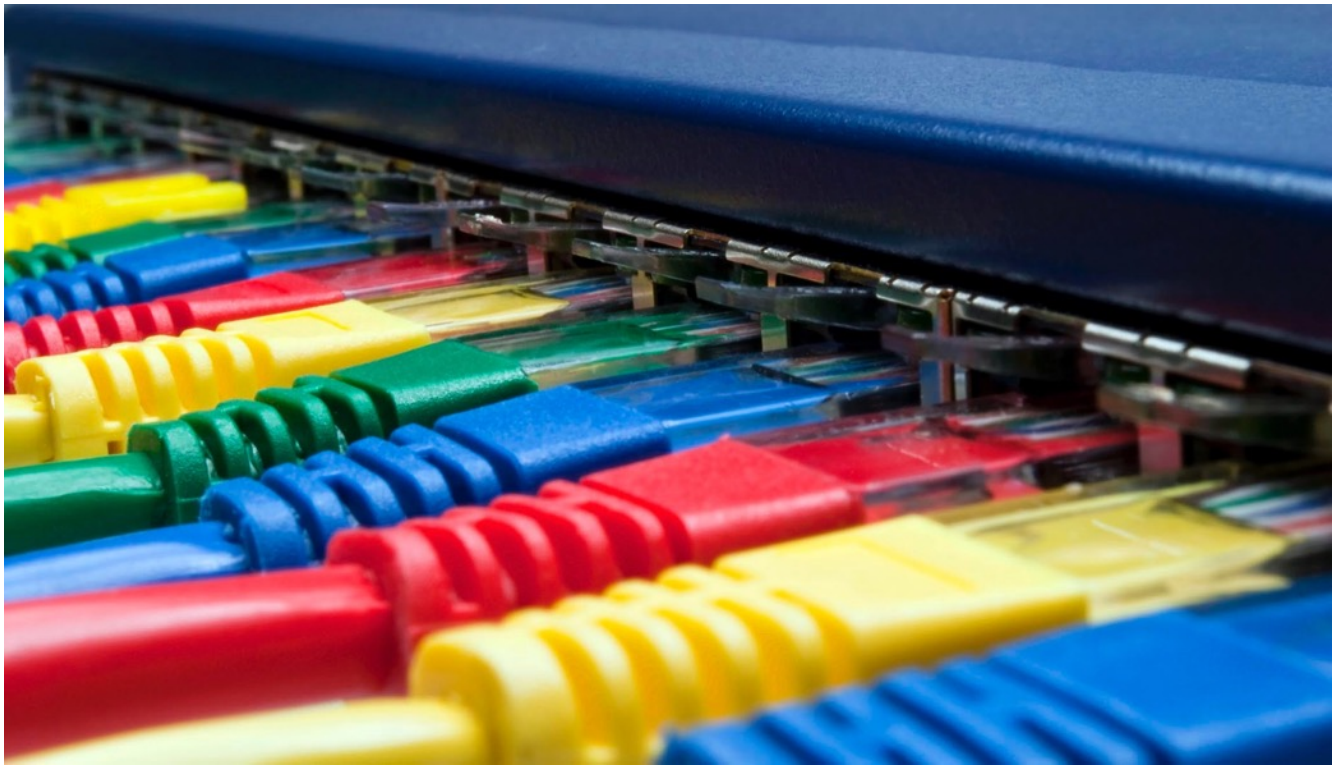
Reliable Transport Protocols

- An Analysis Report

The State University of New York at Buffalo



ABHISHEK GAUTAM
agautam2@buffalo.edu
#50169657

I have read and understood the course academic integrity policy.

- agautam2

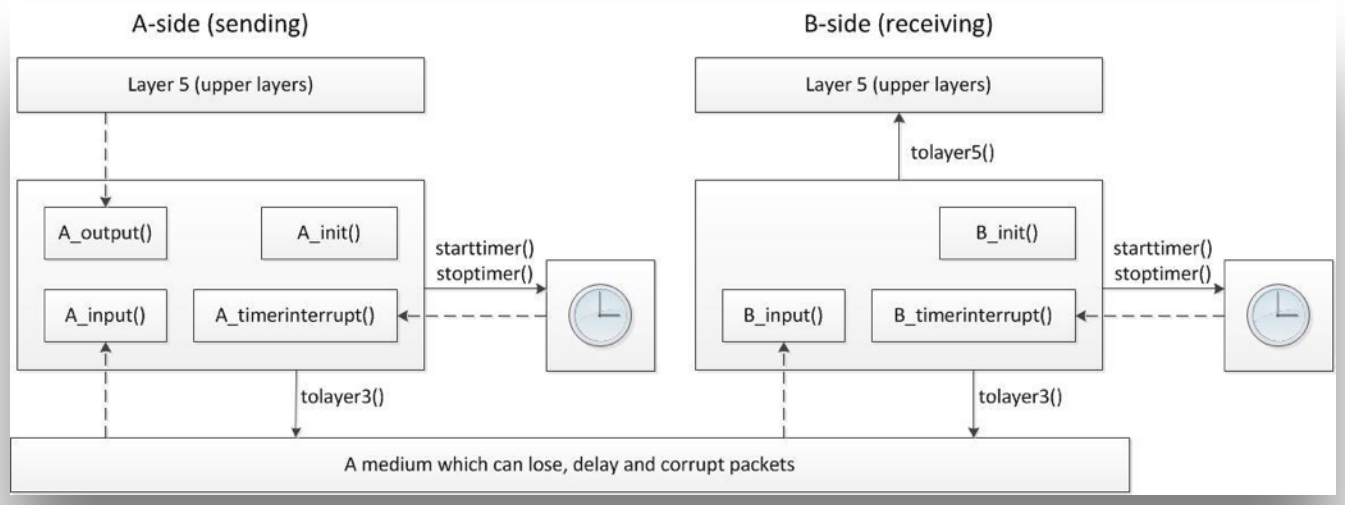
Introduction

In this programming assignment we are supposed to implement sending and receiving transport-layer code for simulating the three versions of the pipelined protocols that are, Alternating Bit Protocol (ABT), Go-Back-N Protocol (GBN) and Selective-Repeat Protocol (SR).

This report covers a brief description about:

1. Design of timer strategy algorithms for each of the three protocols to achieve the optimum throughput.
2. Involves performance comparison on the following grounds:
   - Throughput comparison of the three protocols using different window sizes.
   - Throughput comparison of the three protocols using different loss probabilities.

The structure of the simulated network environment, that we use for this assignment is shown below:



Timeout Schemes

ABT: For ABT, I have used a fixed timeout strategy. I closely monitored the behavior of the protocol under different combinations of timeouts and loss probabilities and used a timeout value of 11.0.

GBN: For GBN, I have varied the timeout value with the varying values of the window size. I observed that the protocol performed better if a larger timeout value was used for a bigger window size. I started with a timer of 15.0 and kept on incrementing it as I ran my tests. I observed that for initial window sizes (10,50 and 100) a variation in the timeout value didn't alter the results to a great extent. I fixed the values when I got the optimal throughput results using this approach. The below code snippet explains my approach:

```
switch (windowSize)
{
        case 200:
                timerIncrement=28.0;
                break;
        case 500:
                timerIncrement=30.0;
                break;
        default:
                timerIncrement=22.0;
                break;
}
```

Our assignment uses the window sizes of 10,50,100,200 and 500. So a timeout of 28.0 is selected when the window size is 200, 30.0 in the case of 500 and 22.0 in all other cases.

SR: For SR, I used a slightly different implementation. I used a timeout value on the per packet basis. Each packet is assigned a time value at the time it is sent. Then after every one time unit, the elapsed time for all the packets that are sent (from A) but not delivered (at B) is checked against the timeout value. If the elapsed time for a packet exceeds the timeout value, it is retransmitted.
In addition to this, I monitored the performance of the protocol after every 20 messages were sent. At every 20 messages, I measured the efficiency of the protocol which can be thought of as *a ratio found by taking in account the packets delivered at B to the packets sent by A.* I conceived that when the efficiency is high, the loss is less. In such an event, the timeout value can be high. But when the packet loss is high, (efficiency is low) the timeout should be a smaller

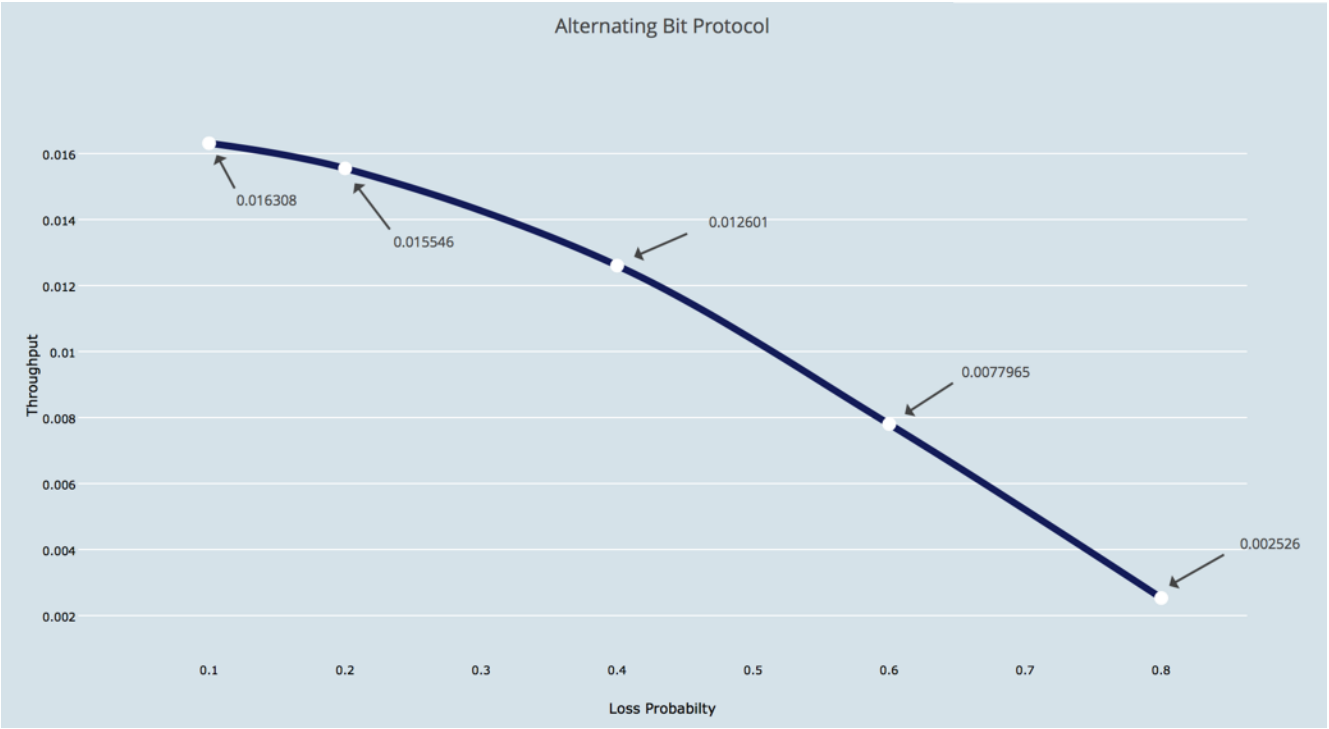
value to detect a loss and the protocol should be ready to retransmit the packet in time. So in a communication of 1000 messages, this was checked 50 times thus making the protocol more effective. I noticed a slight increase in the value of the throughput after deploying this logic.

Experimental Analysis

Experiment 1: Below table explains the environment for experiment 1. Window Size being 10 and 50 for GBN and SR.

Protocol	-m (Messages)	-l (Loss Probability)	-c (Corrupt Probability)	-t (mean time from L5)
ABT	1000	0.1, 0.2, 0.4, 0.6. 0.8	0.2 (fixed)	50.0
GBN	1000	0.1, 0.2, 0.4, 0.6. 0.8	0.2 (fixed)	50.0
SR	1000	0.1, 0.2, 0.4, 0.6, 0.8	0.2 (fixed)	50.0

ABT: The Throughput v/s Loss graph shows the behavior of ABT protocol in the testing environment pertaining to Experiment 1.

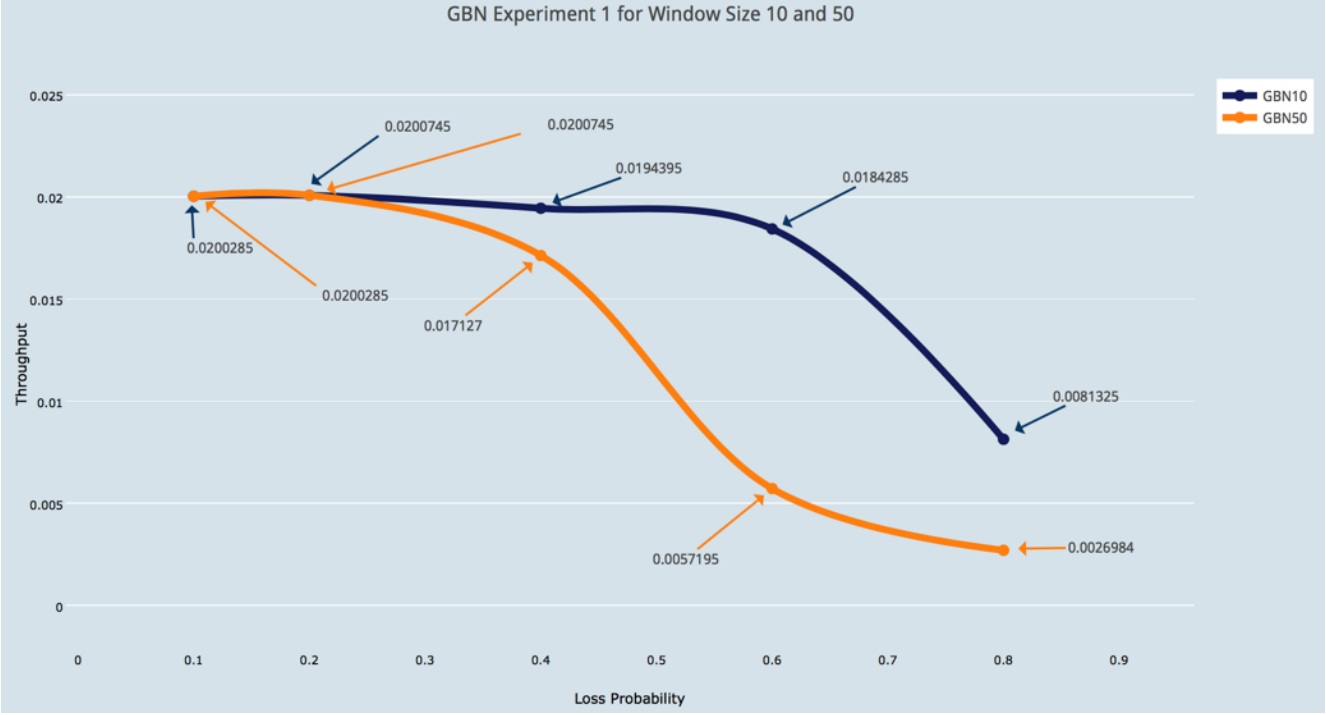


Findings:

1. The throughput decreases gradually with increasing loss probabilities. This is visible at loss probability markings i.e. 0.1, 0.2, 0.4, 0.6 and 0.8

2. The sender A is ready to send only when it gets an ack from the receiver B. So in this event, packets coming from Layer 5 are simply dropped. In my implementation, I observed that 104 packets were dropped/ignored as the sender was not ready, when there was no loss and corruption and 870 packets were dropped when loss probability was 0.8 and corrupt probability was 0.2. This is a clear drawback of ABT as the sender lacks the judgement of buffering messages and sending them when it can.

GBN: GBN was tested for two window sizes (10 and 50). The below graph shows the performance comparison of the two cases in terms of throughput with varying loss probabilities.



Findings:

1. GBN with window size 10 does not suffer much in terms of performance up to loss probabilities of 0.6. A throughput of ~ 0.0184 is achieved at loss probability 0.6. After that, there is a gradual decrease when the loss reaches higher values of 0.8 and 0.9.
2. For window size 50, the protocol started showing a dip in the performance at loss probabilities of around ~ 0.35 - 0.4 and keeps on decreasing from there.

3. For both the window sizes, the protocol had bad performance at higher loss probabilities (>0.7).

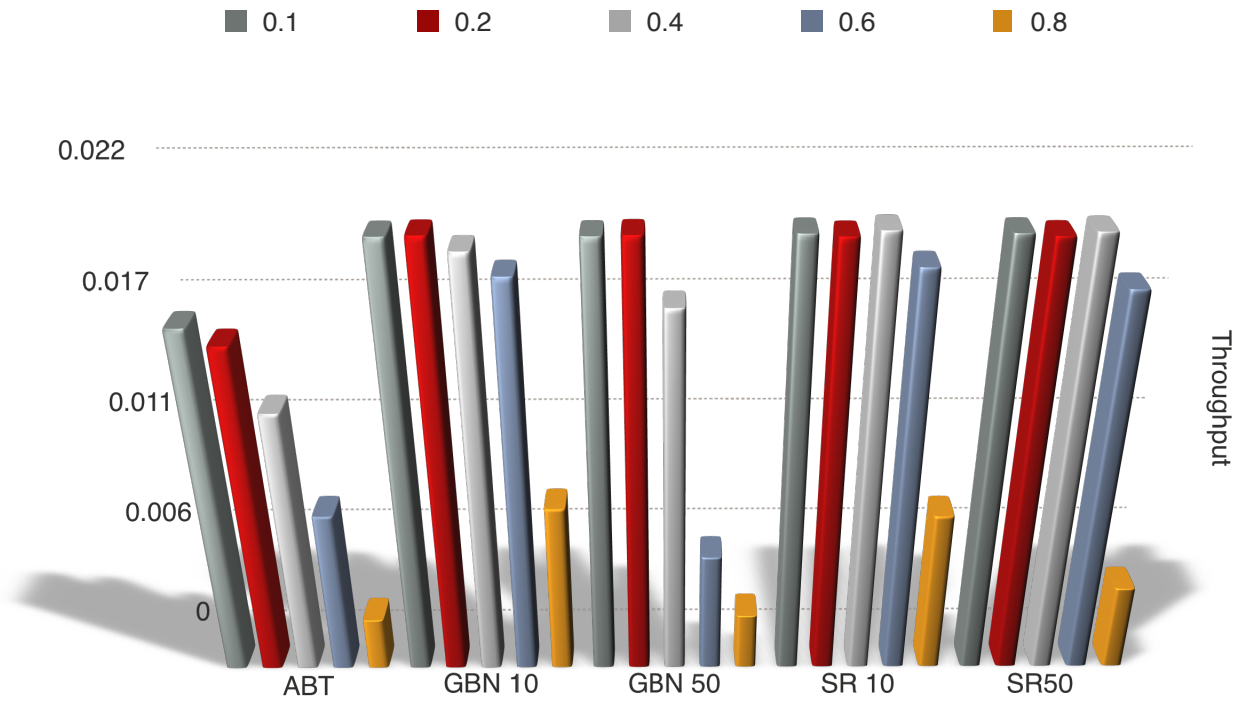
SR: SR was also tested for two window sizes (10 and 50). The below graph shows the performance comparison of the two cases in terms of throughput with varying loss probabilities.

Findings:

1. Better performance even at high loss rates. Performance till 0.6 is quite optimum. The efficiency is high as the receiver is capable of buffering packets. So out of order packets are not discarded by the receiver thus lightening the burden of the sender to retransmit them.
2. Almost similar performance graphs for both window sizes (10 & 50). This implies that window size does not play a significant role in changing the throughput as it did in the case of GBN.
3. Using a timeout selection scheme per 20 messages sent, showed a slight increase in the performance in terms of throughput.

Conclusions - Experiment 1

The below bar graph and table compares the experimental throughput data for the 3 protocols at different values of loss probabilities.



Loss Probability	ABT	GBN -w 10	GBN -w 50	SR -w 10	SR -w 50
0.1	0.016308	0.0200285	0.0200285	0.0201185	0.0201185
0.2	0.015546	0.0200745	0.0200745	0.0199995	0.0199995
0.4	0.012601	0.0194395	0.017127	0.020239	0.0201765
0.6	0.0077965	0.0184285	0.0057195	0.018765	0.017855
0.8	0.002526	0.0081325	0.0026984	0.007727	0.0040705

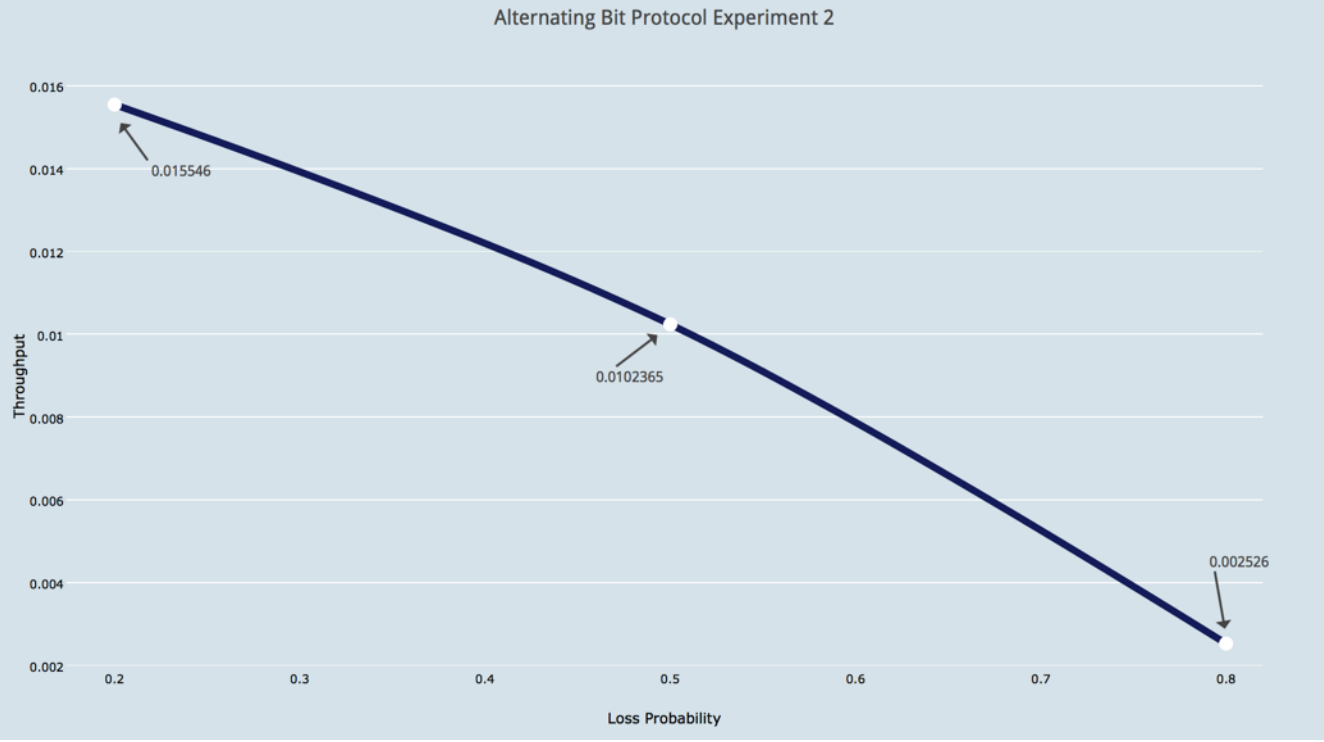
Analysis from data and graphs:

1. ABT is the worst performer of the three due to simple FSM design, lack of sliding window logic, and no buffering at either end.
2. Performance of GBN is better in smaller window sizes for higher loss probabilities (This reasoning will be solidified in Experiment 2). This points to an important observation that low window sizes are better at significant losses. Thus GBN can perform better if window size adapts itself according to the loss values.
3. SR outperforms ABT and GBN.

Experiment 2: Below table explains the environment for experiment 2. Window sizes being 10, 50, 100, 200 and 500 for GBN and SR.

Protocol	-m (Messages)	-l (Loss Probability)	-c (Corrupt Probability)	-t (mean time from L5)
ABT	1000	0.2, 0.5, 0.8	0.2 (fixed)	50.0
GBN	1000	0.2, 0.5, 0.8	0.2 (fixed)	50.0
SR	1000	0.2, 0.5, 0.8	0.2 (fixed)	50.0

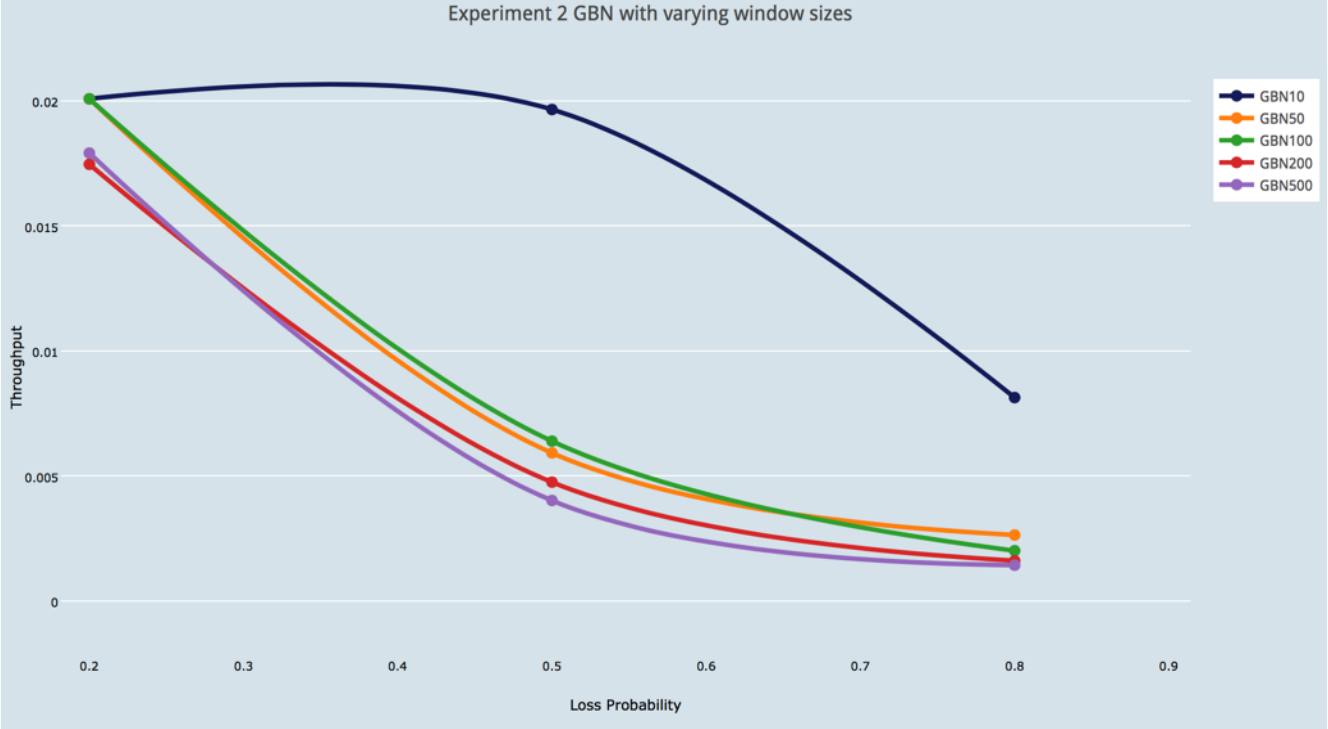
ABT: The Throughput v/s Loss graph shows the behavior of ABT protocol in the testing environment pertaining to Experiment 2.

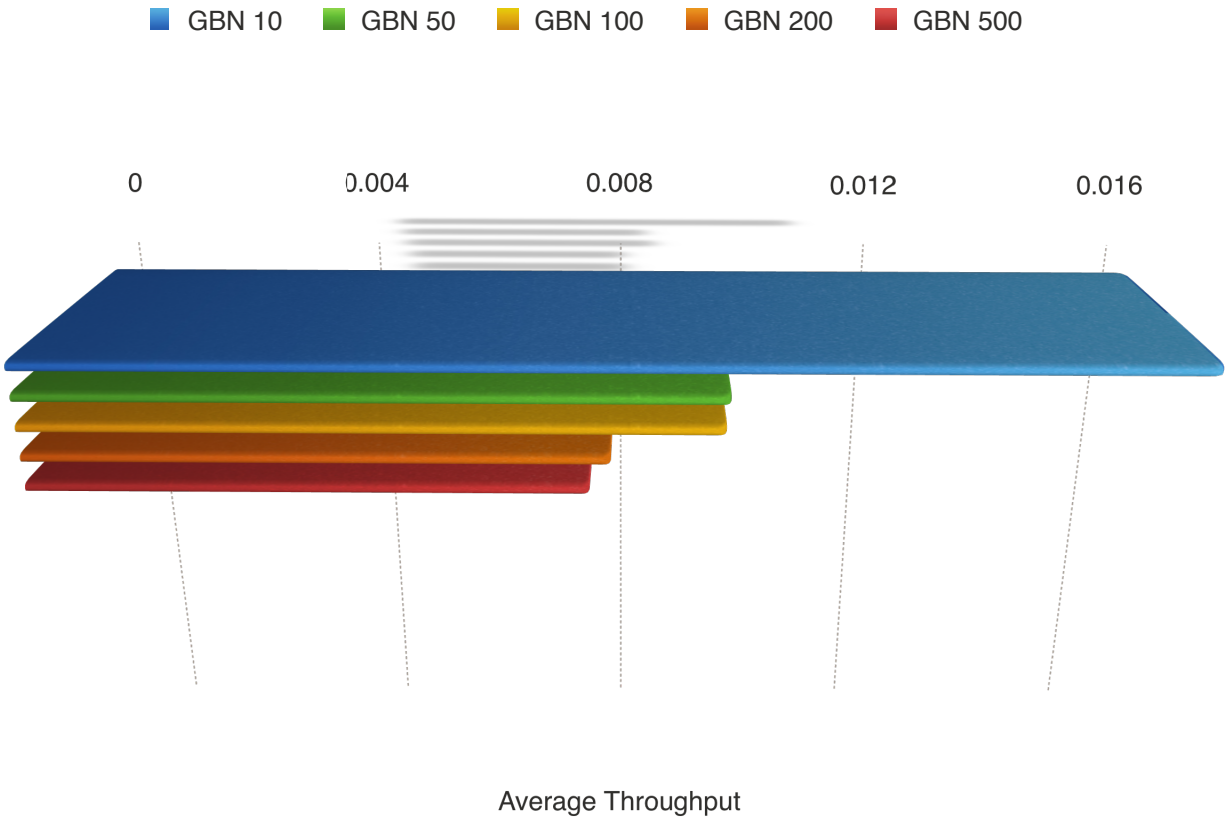


Findings:

1. The findings are similar to experiment 1. As before, the throughput decreases gradually with increasing loss probabilities.

GBN: The Throughput v/s Loss graph shows the behavior of GBN protocol at varying window sizes in the testing environment pertaining to Experiment 2.



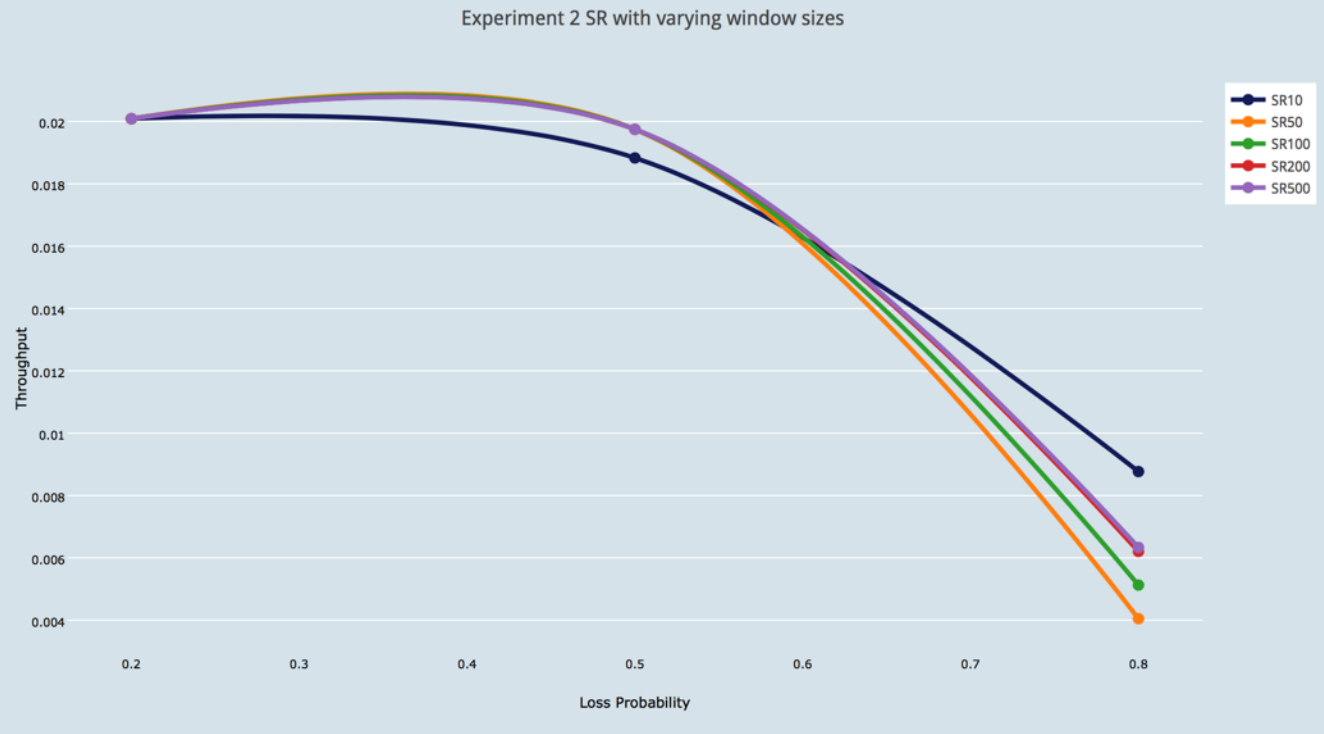


Findings:

1. Smaller window sizes give better performance if a smaller timeout is used. I observed the performance of the protocol at different timeout values for different window sizes and came up with the timings as explained in the *Timeout Scheme* section.

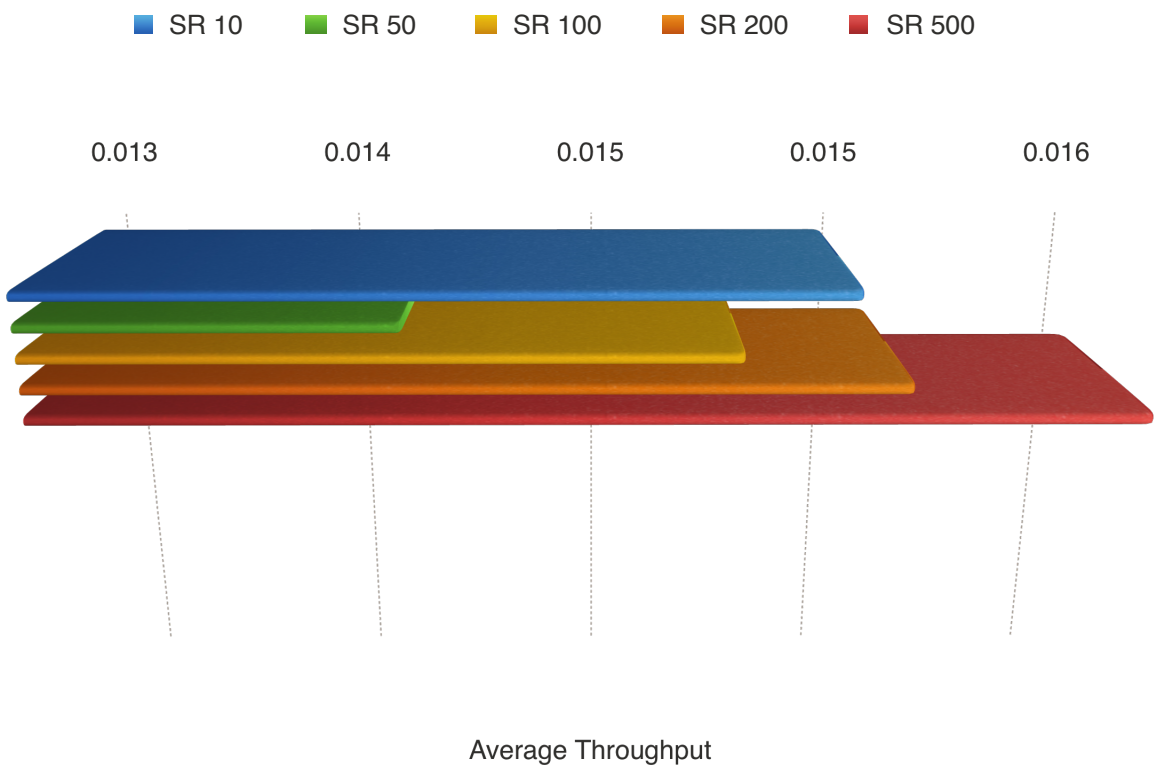
2. As observed from Experiment 1, the throughput indeed drops with increasing window size. The above bar chart shows a comparison of average throughput at all loss values (0.2, 0.5, 0.8) with varying window sizes. Throughput decreases from cyan to red. Thus GBN suffers at bigger values of window size.

SR: The Throughput v/s Loss graph shows the behavior of SR protocol at varying window sizes in the testing environment pertaining to Experiment 2



Findings:

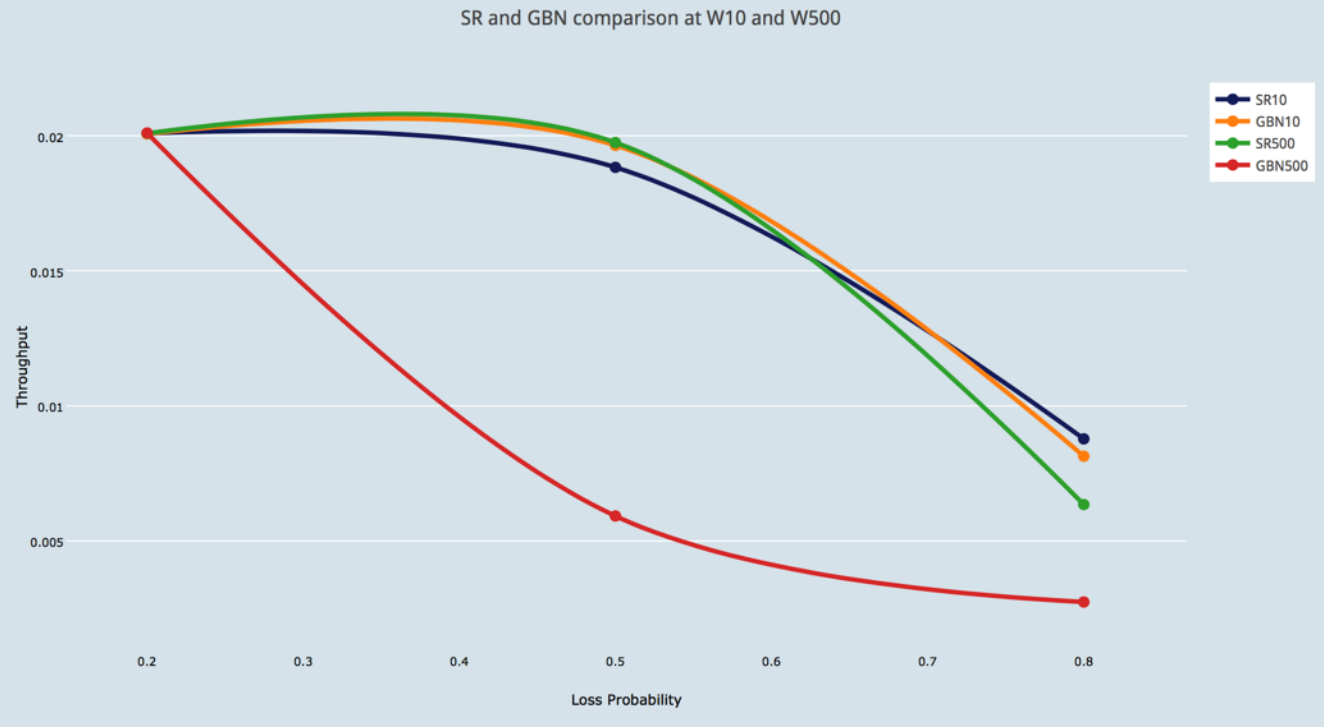
1. As the loss rate increased, SR with higher window size performed better. (Purple>Red>Green>Orange in the graph). Though SR with window size 10 witnessed a greater throughput at loss of 0.8.



2. Though performance was nearly ~0.014 for all window sizes, SR performed better at a higher window size. The above bar chart shows a comparison of average throughput at all loss values (0.2, 0.5, 0.8) with varying window sizes.

Conclusions - Experiment 2

1. I observed that SR and GBN give comparable performance at a small window size (10 in the graph above).
2. Though at a big window size like 500, GBN suffered a big dip.



References

The assignment and its analysis would not have been successful without the following:

1. Guidance and help of Professor Dimitrios Koutsonikolas and TA Swetank Kumar Saha.
2. Computer Networking - A Top Down Approach by Ross and Kurose.
3. Network Emulator code by J.F. Kurose
4. http://stackoverflow.com
5. Important MIT coursework
6. Research paper [1]
7. Simulatons - http://www.ccs-labs.org/teaching/rn/animations/gbn_sr/
8. Graphs generated at - https://plot.ly