Analysis Report for PA2 CSE 589 Modern Networking Concepts

Vibhav Virendra Yawalkar - 50321090

<u>I have read and understood the course academic integrity policy.</u>

1.Brief description of the timeout scheme you used in your protocol implementation and why you chose that scheme.

Answer: In the handout for the project we are given an average one way timeout 5 for reference. Hence the RTT is 10 (two way, sending message and then receiving the acknowledgement, assuming zero transmission time). Now, in ABT I am using the timeout of ~10, since in ABT we expect to receive the acknowledgement from the receiver before sending another packet. However, this assumption doesn't work for protocols like GBN and Selective Repeat. Here we can have a lot of packets in the network between the sender and the receiver and which can lead to different values of the RTT every time we send a packet. For GoBackN, I did trial and error, initially I set the timeout to 10 as in case of ABT, but then I was getting a lot of retransmissions due to pre-mature timeouts and hence I decided to gradually increase the timeout which could lead to a trade off between excessive retransmissions and waiting too long. I was able to arrive at timeout 20 after doing these multiple attempts. Similarly trail and error was done for Selective repeat timeout(details in answer for question 2).

2.Brief description (including references of the corresponding variables and data structures in your code) of how you implemented multiple software timer in Selective Repeat (SR) using a single hardware timer.

Answer: Data structure that I used for the selective repeat timer implementation is

```
Struct senderMsg {

Char message[20];

Float start_time;

Int acked; };

Struct senderMsg senderBuffer[1001];
```

Here, I store the message sent and the time during which it was sent, the time is denoted by start_time variable. I populate the start_time in the senderMsg struct after I send the packet to the transport layer. To get the time I use the get_sim_time() function.

So, for every packet I am storing the time when the packet is sent and I have a timeout of t = 40 which is the RTT estimate (considering the traffic in the system using trial and error for different throughput values). Now, in the A_init() method itself I have started the timer starttimer(0, 1.00), so after every 1 unit of time the A_interrupt method is called to check if the timer for any of the packet in the window is expired or timeout. We detect the timeout in the A_interrupt method where I calculate the difference between the current_time and the time stored in the senderBuffer structure mentioned above when the packet was sent to the transport layer by the sender. If the difference between the current time and the start time is greater than the RTT and the acknowledgement was not received for the packet, then we retransmit the packet. So, this is kind of a polling approach where we keep checking for the timeouts of the various packets. This approach is accurate and we are able to detect the timeouts correctly, however we can make this efficient by storing the relative expiration time of the packets sent.

In the A_interrupt method, at the end we again start the timer of 1 unit of time. Thus the hardware timer will give us interrupt every 1 unit of time, and thus we maintain the software timers for multiple packets and detect their timeouts.

3.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:

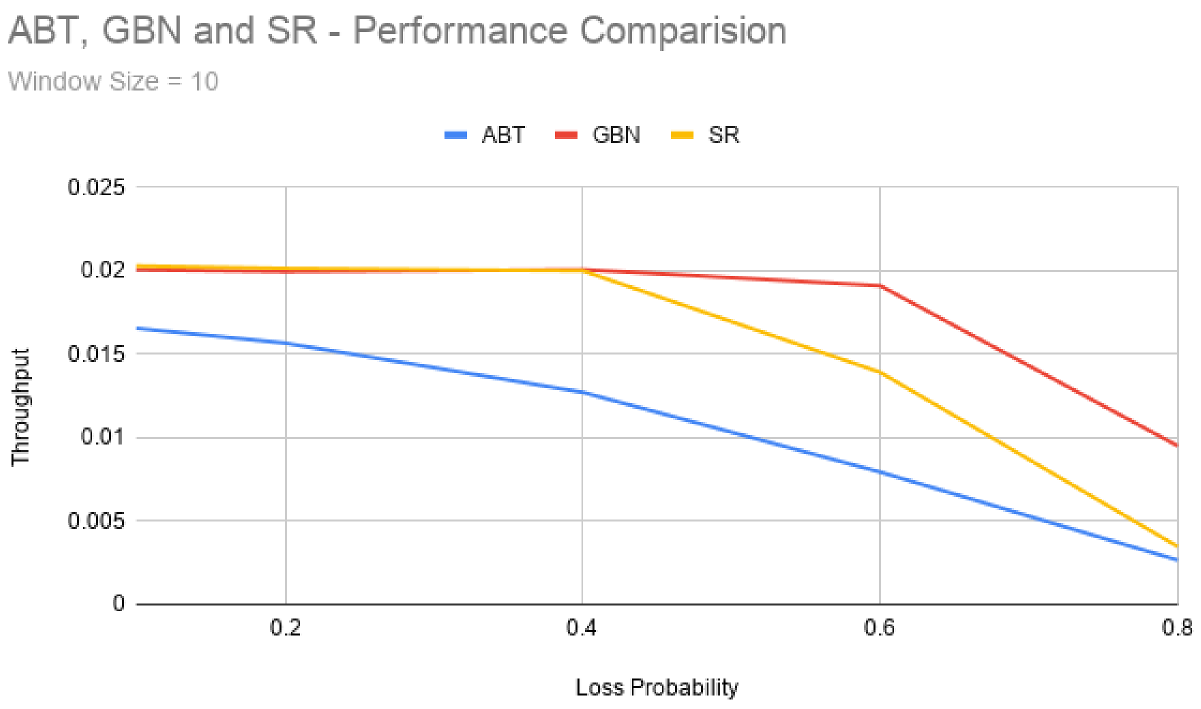
What variations did you expect for throughput by changing those parameters and why?

Do you agree with your measurements; if not then why?

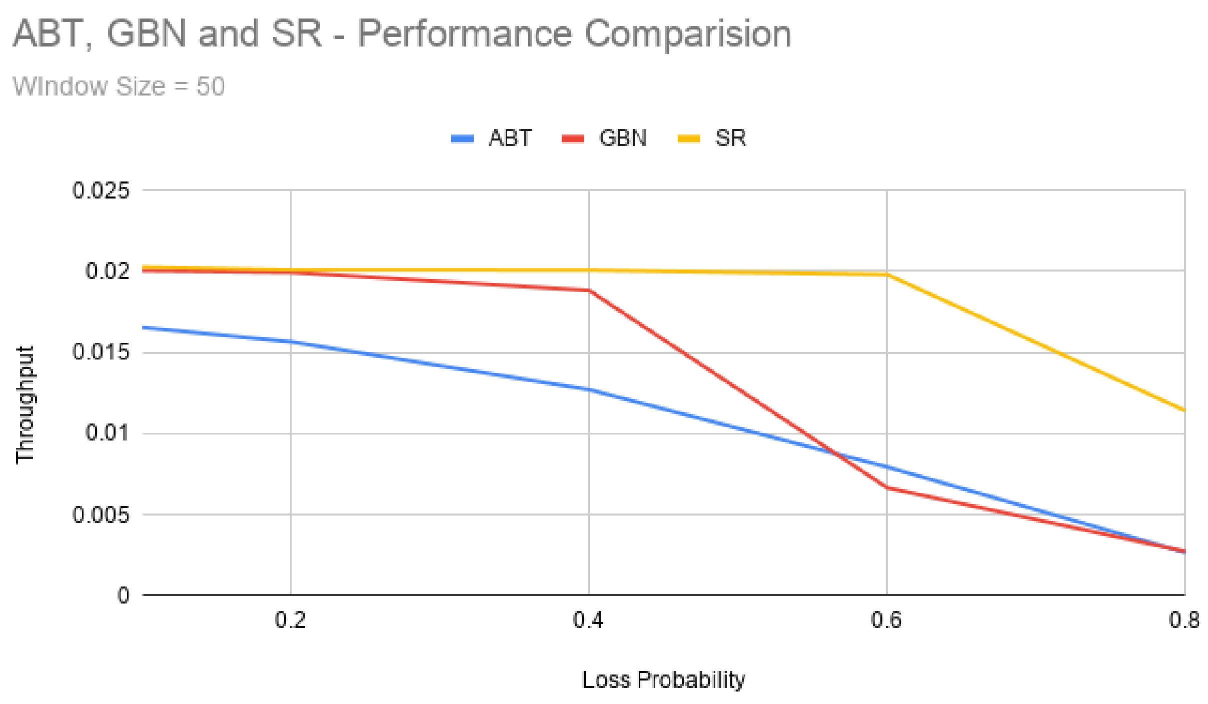
Experiment 1:

With loss probabilities: {0.1, 0.2, 0.4, 0.6, 0.8}, compare the 3 protocols' throughputs at the application layer of receiver B. Use 2 window sizes: {10, 50} for the Go-Back-N version and the Selective-Repeat Version.
<u>Expected Graphs</u>

- Window size: 10; X-axis: Loss probability; Y-axis: Throughput (ABT, GBN and SR) in one graph/plot.
- Window size: 50; X-axis: Loss probability; Y-axis: Throughput (ABT, GBN and SR) in one graph/plot



Observations and Analysis: Here the window size is fixed to 10 and we can see as the loss probability increases, more and more packets don't reach the receiver, throughput is decreased for all the protocols. Particularly after the 0.4 probability we can see the sudden dip in the throughput. We can see that eventually GBN performs better, when the loss probability is high, due to its excessive retransmission mechanism. When we go on increasing the loss we can see that SR is converging to ABT like behavior, this can be attributed to the property that SR only retransmits the un-acked packets. The performance is as one would expect from these protocols.



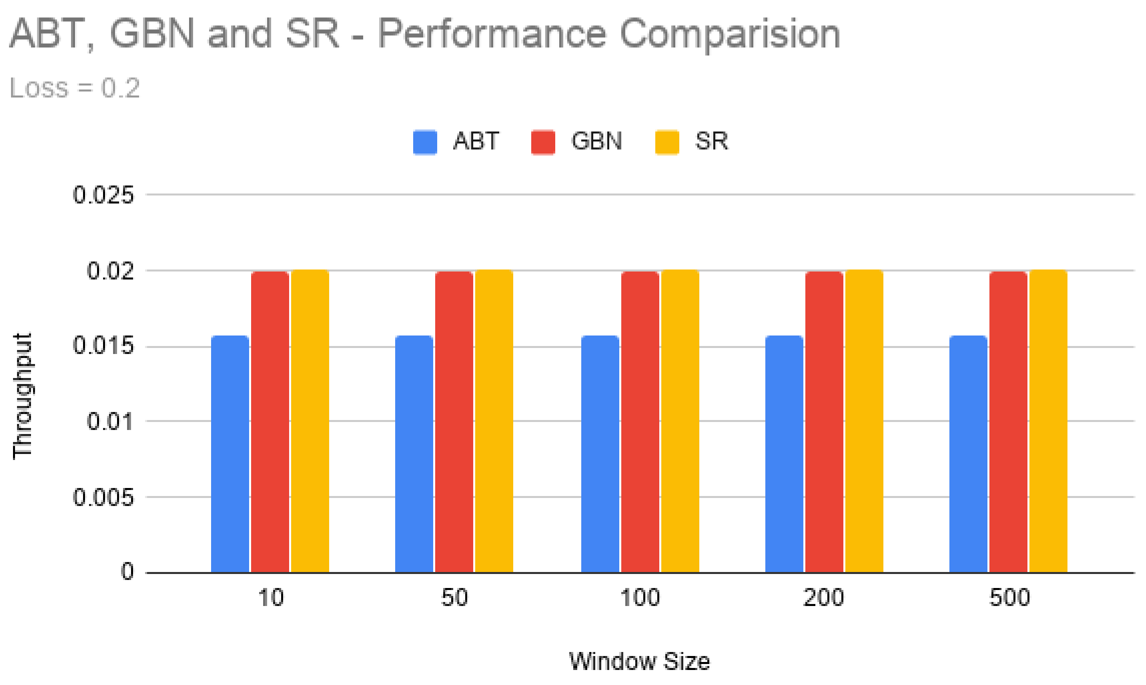
Observations and Analysis: Here we can see that the window size is increased to 50, ABT is window size agnostic hence its throughput stays the same, however in GBN we can a significant drop in throughput due to as more packets will get retransmitted in case the window size is large for a single packet which is not acked. In this case we can see that SR is performing better than GBN and GBN is performing almost equivalent to ABT, this is the impact of increasing the window size. We don't see a significant impact on the performance of SR by increasing the window size.

SR performs the best clearly.

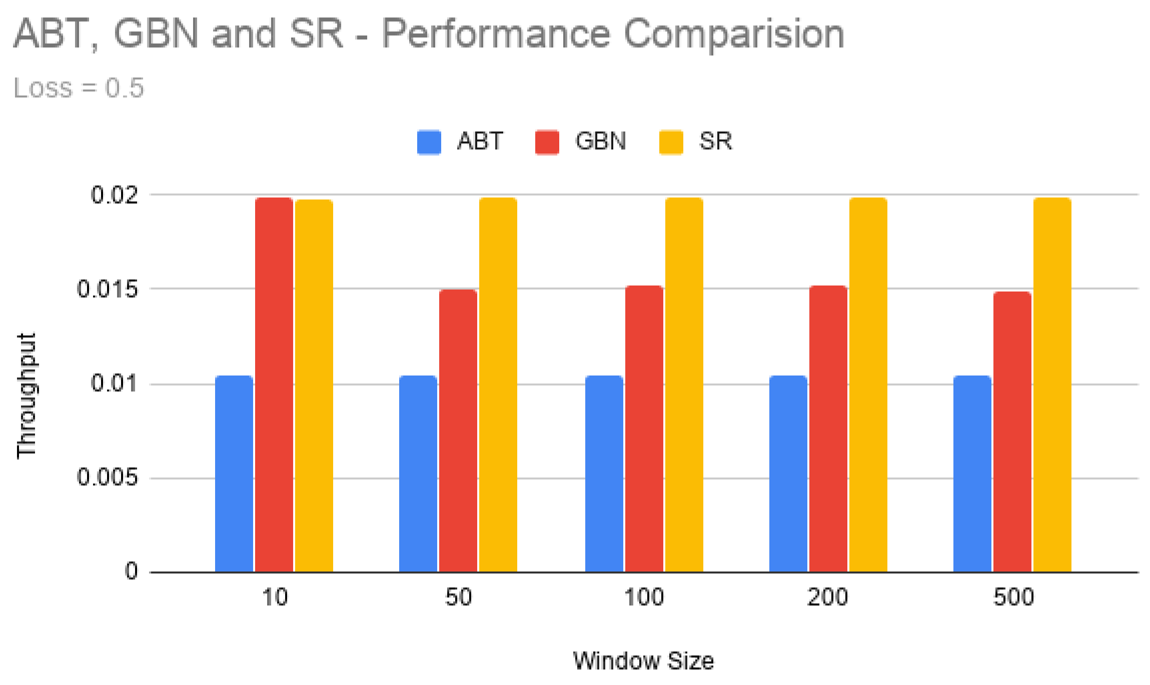
- **Experiment 2**

  With window sizes: {10, 50, 100, 200, 500} for GBN and SR, compare the 3 protocols' throughputs at the application layer of receiver B. Use 3 loss probabilities: {0.2, 0.5, 0.8} for all 3 protocols.
  Expected Graphs

  - Loss probability: 0.2; X-axis: Window size; Y-axis: Throughput (ABT, GBN and SR) in one graph/plot.
  - Loss probability: 0.5; X-axis: Window size; Y-axis: Throughput (ABT, GBN and SR) in one graph/plot.
  - Loss probability: 0.8; X-axis: Window size; Y-axis: Throughput (ABT, GBN and SR) in one graph/plot.



Observations and Analysis: Here we can see that with increasing window size ABT throughput is not impacted as it is window size agnostic, doesn't use windowing. Also, since the loss rate is less we see that increasing the window size has least impact on the performance of the GBN protocol, which is quite similar to the SR in the case. If loss probability increases, we will see later that GBN performs worse due to retransmission of a large window size.



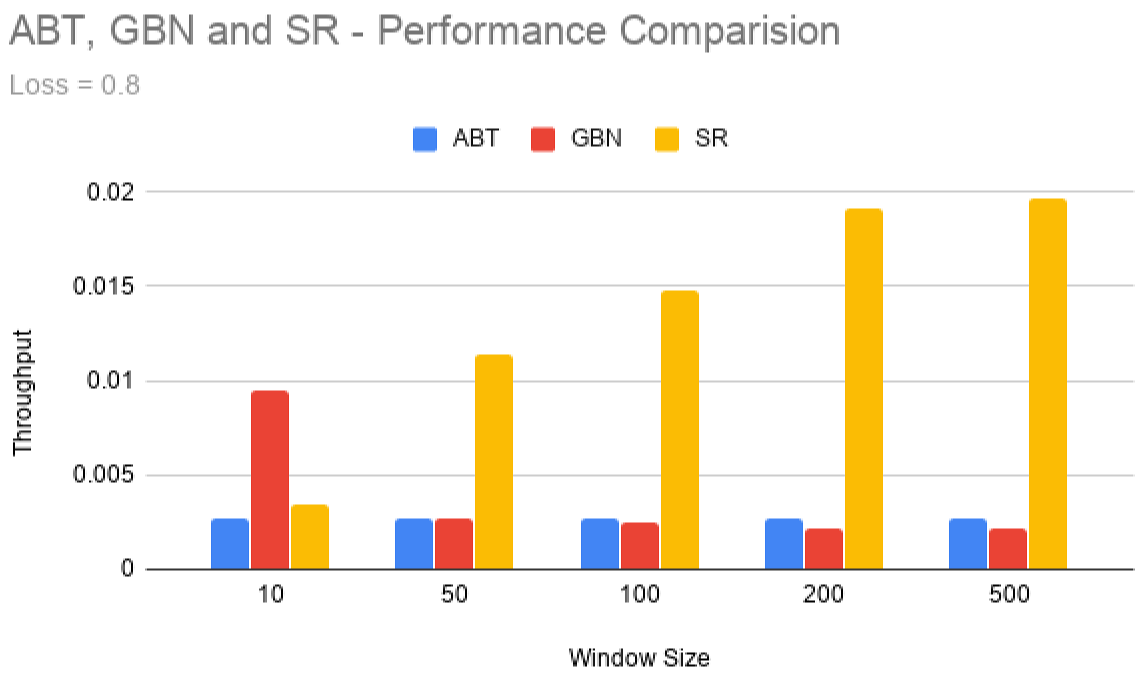
Observations and Analysis: Here we can see that with increasing window size ABT throughput is not impacted as it is window size agnostic, doesn't use windowing. However, with the increase in window size we find that the GBNs performance deteriorates as compared to Selective repeat as GBN does retransmissions of the entire window of size N if the packet is missed and if the window size is large this will lead to retransmission of large number of packets thus deteriorating the performance. However, SR performs the best in case of both loss probabilities of 0.5 and 0.8, since it only retransmits the un-acked packets.

With loss probability 0.8 we can see that GBN performance (throughput) decreases even further. It performs worse that ABT in some cases or close to ABT when the window size is large. This is expected behavior and our experiment matches that.

With just one exception for the reading for window size 10, which is not as one would expect.

SR performs better than ABT as it is able to transmit multiple packets, without waiting for acknowledgement and it also has a cumulative acknowledgement.

SR performs the best as expected.


ABT, GBN and SR - Performance Comparision
Loss = 0.8
ABT
GBN
SR
Throughput
0.02
0.015
0.01
0.005
0
10
50
100
200
500
Window Size