CSE 589

Analysis and Report for PA2

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

Name: **Dhairya Hardikbhai Desai**

UB Number: **5024 5434**

Timeout Scheme

- ABT Protocol:

It is known that a packet sent into the medium takes 5 time units to arrive on the other side when there are no packets in the medium. Thus the total RTT for a packet is 10 time units when there are no packets in the medium.

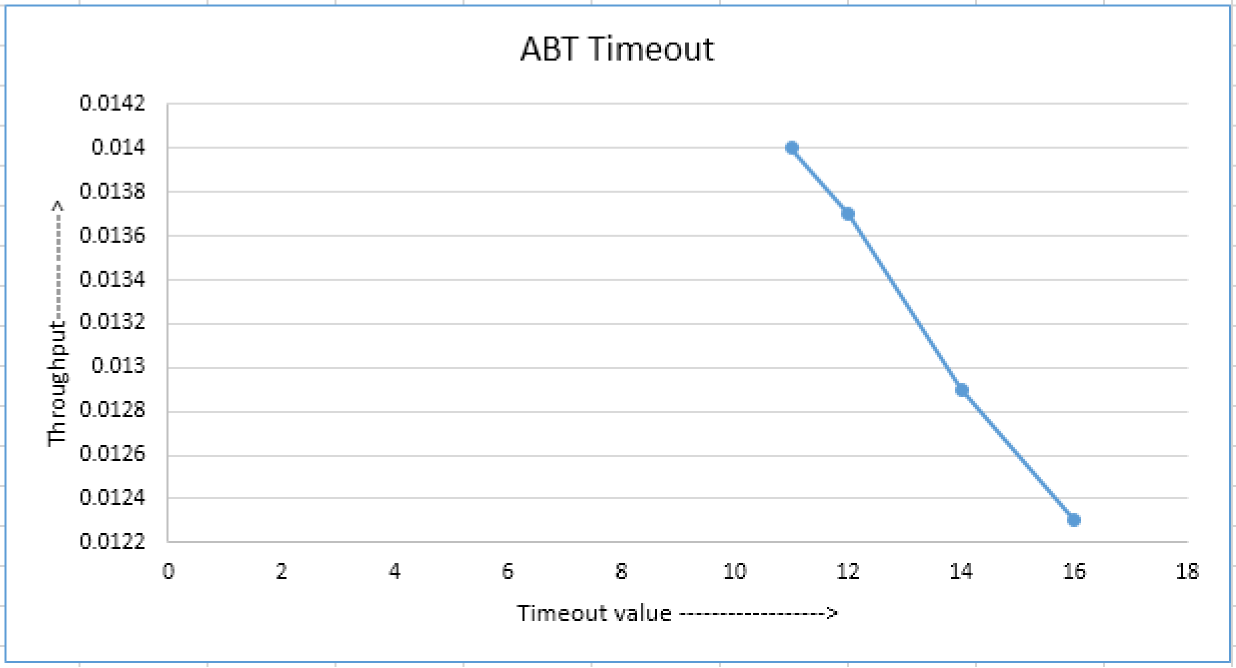
Since ABT is a stop and wait protocol, there is only 1 inflight sender packet at any given point of time.

Here we can consider queueing delay and propagation delay to be zero.

Transmission delay dTran = 1/5 time units

Considering some processing delay and Transmission delay and adding it to 10 time units, the timeout value that I have chosen is **11 time units**.

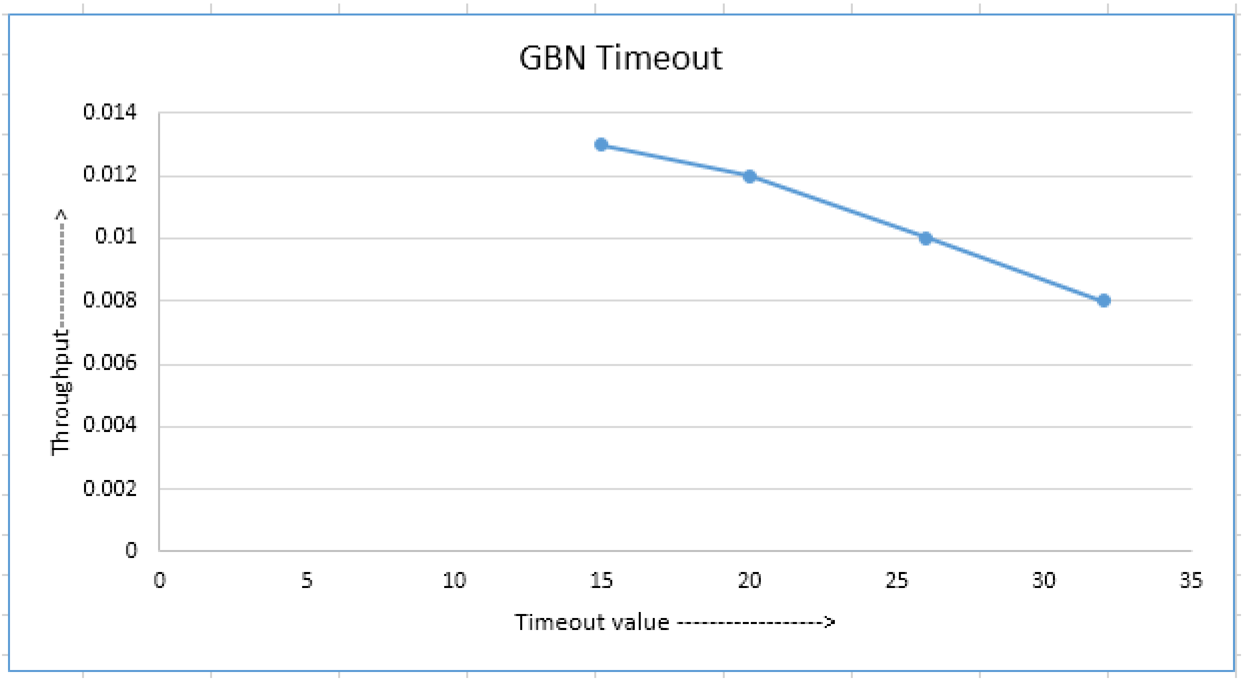
Consider the below graph of throughput vs timeout. I calculated the average throughput for various cases and keeping different timeout values starting from 11. We can see that the throughput decreases as the value of timeout increases.



- GBN Timeout:

For GBN, the number of inflight packets increases. Also due to multiple retransmission, the queuing delay and transmission delay will be more. Hence the timeout value should be reasonably greater than 10 time units.

Consider the below graph of throughput vs timeout. I calculated the average throughput for various cases and keeping different timeout values starting from 15(chosen a number randomly little more than 10).



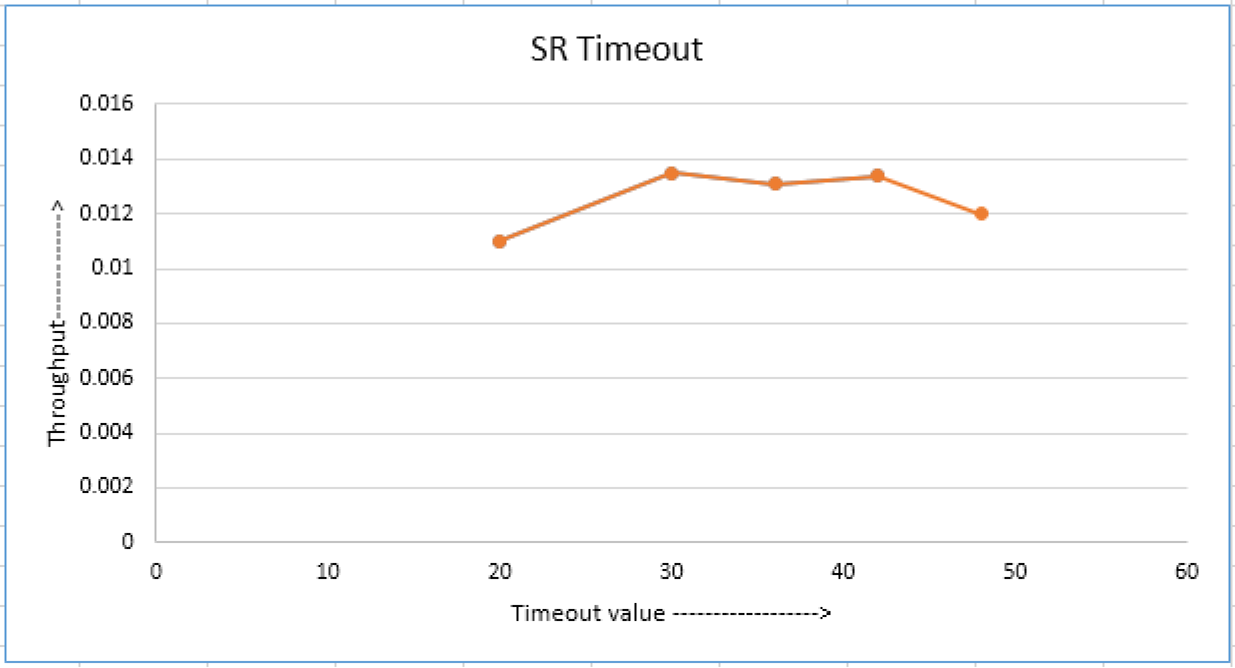
The throughput is maximum for timeout value of 15.

Thus the timeout value is **15 time units.**

- SR Timeout:

In case of SR protocol, there are multiple timers.

In order to decide the timeout value, consider the graph below for throughput vs timeout where I have calculated the average throughput for different cases, keeping different values for timeout starting from 20



From the above graph, the throughput increases for values 30 and 42. I have chosen the greater of the 2 values i.e 42 as a strategy for congestion control. Since increased timeout is favorable when congestion increases.

Thus the timeout value that I have chosen is **42 time units.**

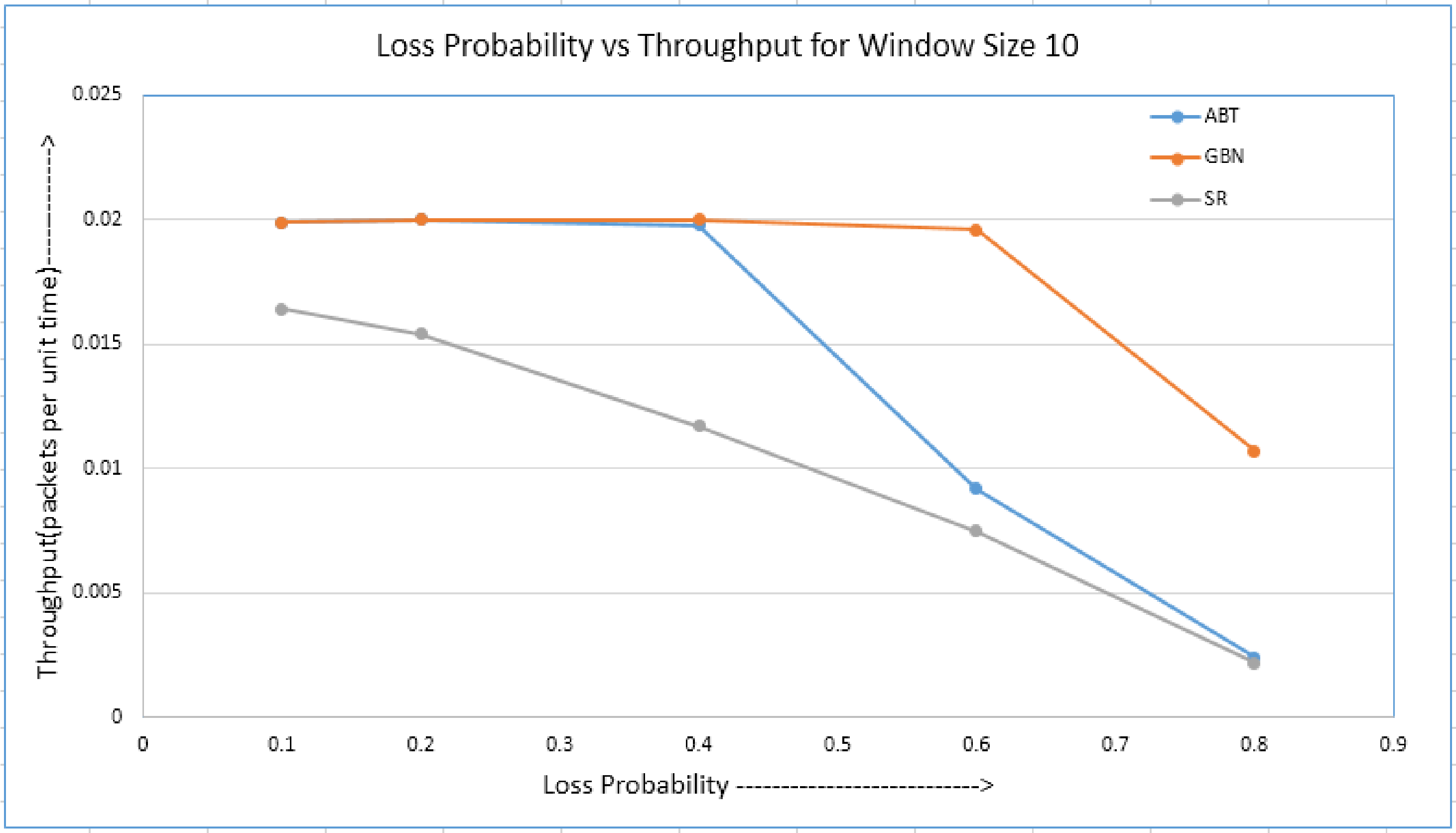
Multiple Software Timer Implementation

For implementing multiple software timers with a given hardware timer, the strategy that I have implemented is as follows:

- When a packet is sent from Sender A and if the timer is not on, the timer is started with the decided timeout value. Also I capture the time at which this packet is sent in a variable **last_timer.**
- Now when the next packet is to be sent from A and the timer is on, I capture the time at which this packet is sent. Also I calculate the time difference between current time and time at which last packet was sent and I store this time difference in an array **timer_start_time** and store the packet sequence number in the corresponding index of another array **timer_seq_number.** Theses arrays are maintained as Queues and they are the backbone of the implementation.
- When ACK is received for any of the packets, I do not restart the timer but simply mark that ACK is received for packet with particular sequence number. This information is stored in the array **packet_ack.**
- Now, whenever a timer interrupt is called, I check the very first packet in the array **timer_seq_number** to see if its ACK is received or not**.** (I do not empty the queue since for larger window sizes, it will add unnecessary overhead. Hence the arrays that I have taken are of sufficiently large size. So, I check the packet at the base pointer)
- ACK is received or not can be checked with the help of array **packet_ack.** If the ACK was not received, I retransmit that packet, again calculate the time difference between current time and time at which last packet was sent and I store these details in timer_start_time and timer_seq_number in the next available slot. (can be obtained by maintaining a pointer)
- Then I increment the base pointer by 1, and start the timer for the next available packet in the queue. I restart the timer with the value obtained from **timer_start_time.**
- In this way I can maintain multiple timers by maintaining a queue of timings obtained by subtracting the time at which last packet was sent from the current time at which current packet is sent.

Experiment 1

1. Window size: 10; X-axis: Loss probability; Y-axis: Throughput (ABT, GBN and SR)

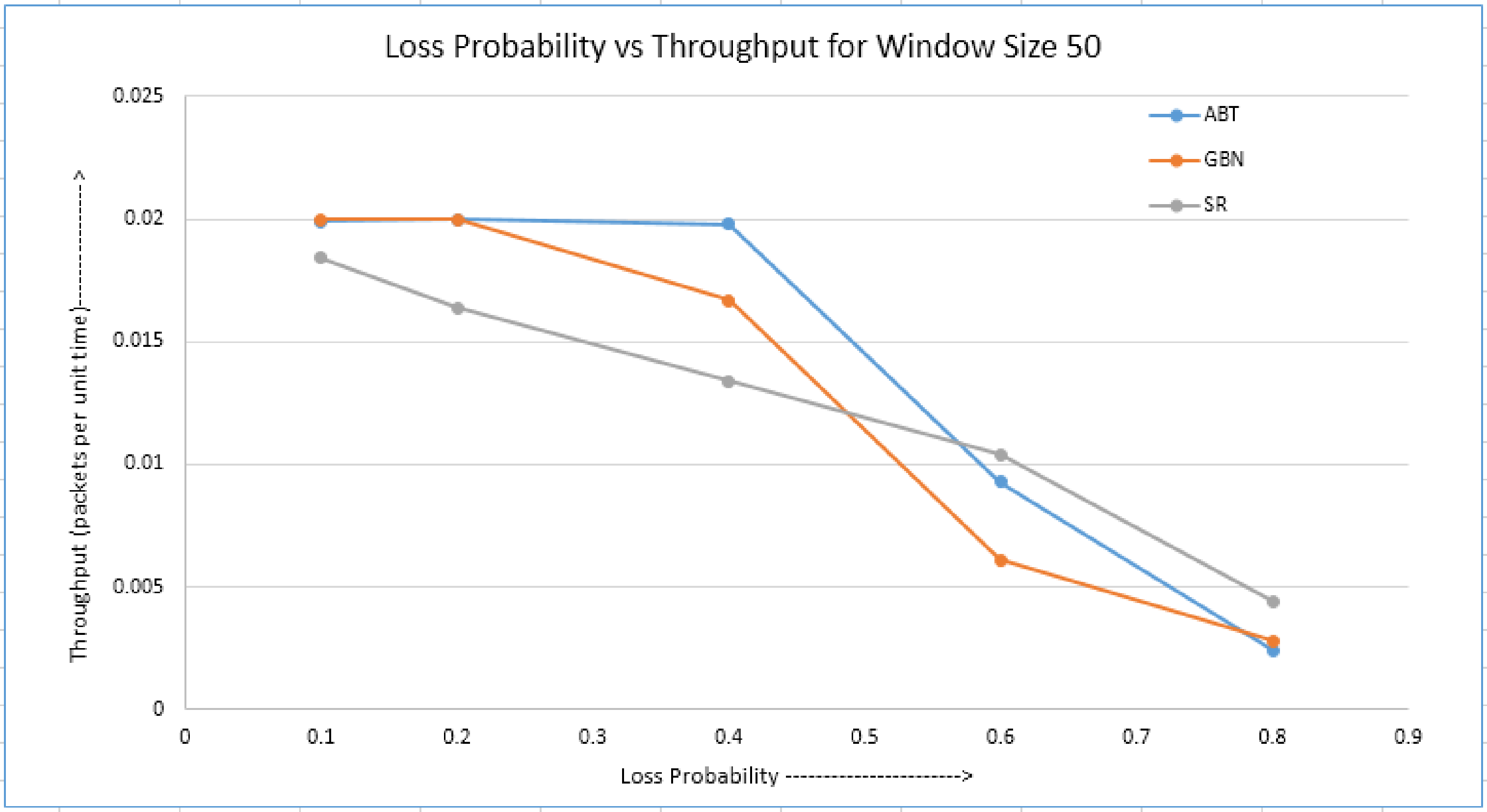


Here kindly note that I have taken average of throughputs obtained for different seed values for a given experiment.

<u>Observation Points based on above graph:</u>

- For all the three protocols, the throughput decreases as the loss probability increases.
- For loss probability till 0.4, the throughput for ABT and GBN are almost same, but for higher loss, GBN provides better throughput as compared to ABT
- As the loss increases, there is sudden dip in the throughputs of ABT and GBN, but in case of SR, the curve is almost linear i.e unlike ABT and SR, there is no greater decrease in the throughput with little increase in loss probability.
- Also, the overall throughput of SR is less than ABT and GBN which is not the expected behavior. A possible reason for this is increased processing delay (due to multiple software timers and non-cumulative ACK) in case of SR protocol

2. Window size: 50; X-axis: Loss probability; Y-axis: Throughput (ABT, GBN and SR)



Here kindly note that I have taken average of throughputs obtained for different seed values for a given experiment.

Observation Points based on above graph:

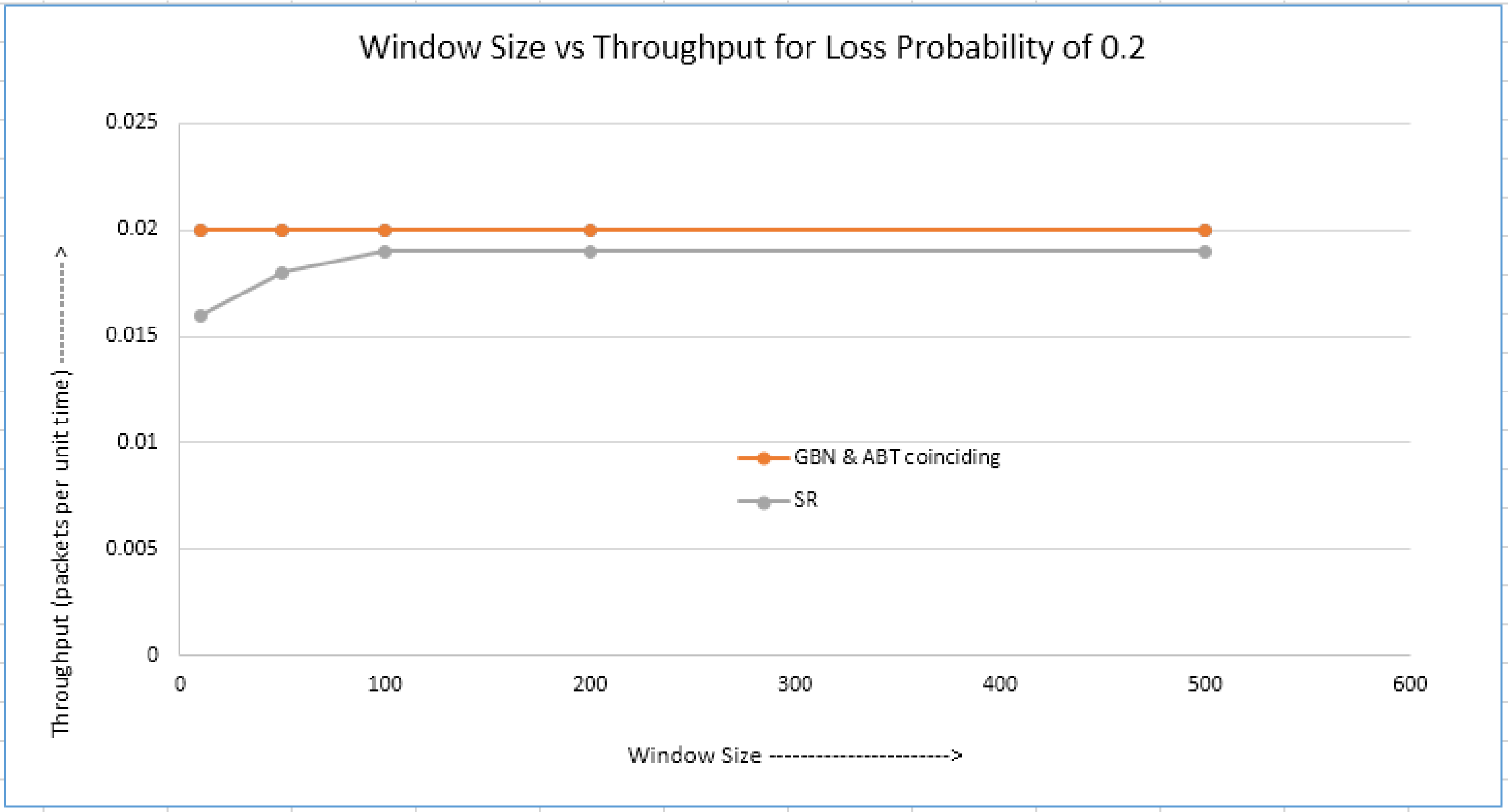
- For all the three protocols, the throughput decreases as the loss probability increases.
- For higher loss probability (greater than 0.6), the throughput of SR is more than that of ABT and GBN.
- There is almost linear relation between loss and throughput for SR. Where as in GBN and ABT, the throughput substantially decreases for certain increase in loss probability.
- Also GBN gives worst performance for increased loss probability.

Conclusion based on observations of Experiment 1:

- When loss probability is more, SR gives better performance as compared to GBN and ABT irrespective of the window size. Thus, SR protocol is to be preferred in a lossy network.
- Performance of GBN is maximum when the loss probability and window size is small.
- Performance of ABT is good when loss probability is less, but it decreases drastically with increases in loss because of its stop and wait nature. Thus ABT should only be preferred for lossless network.
- Most of the observation is as per expectation except for the fact that for lower window size, throughput of ABT and GBN exceeds the throughput of SR. As mentioned, the reason for that may be processing delay due to complexity of SR. Another reason for this behavior can be increased timeout value of SR as compared to ABT and GBN.
- Thus we can conclude the timeout value should be more carefully chosen especially for a more complex sliding window protocol like SR.

Experiment 2

1. Loss probability: 0.2; X-axis: Window size; Y-axis: Throughput (ABT, GBN and SR)

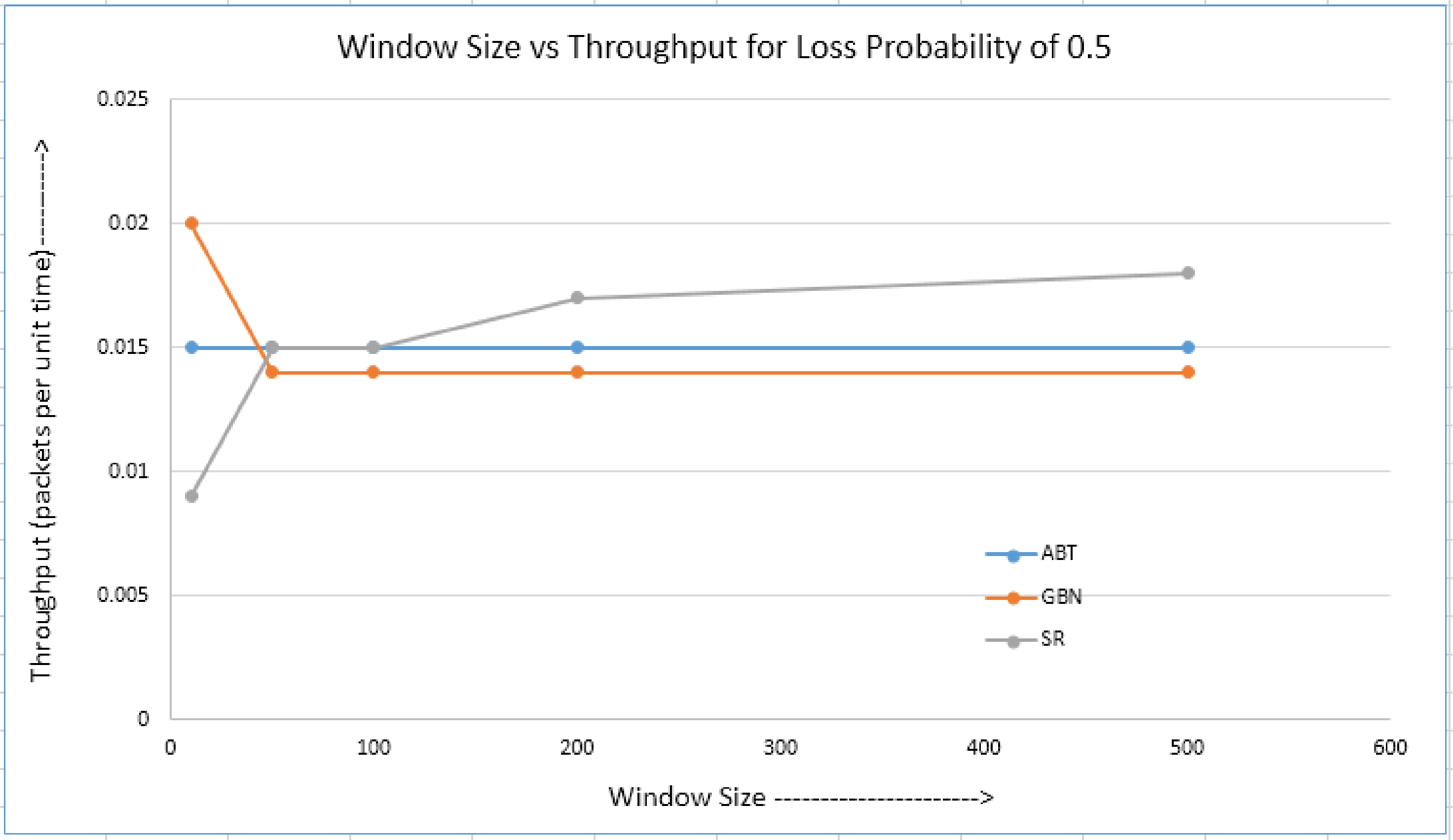


Here kindly note that I have taken average of throughputs obtained for different seed values for a given experiment.

<u>Observation Points based on above graph:</u>

- For GBN, the throughput remains the same irrespective of the size for loss probability of 0.1 and this coincides with the throughput of ABT protocol.
- For SR, the throughput is relatively less for very small window sizes (10 and 50), and then increases and remains same for the increase in window size.

2. Loss probability: 0.5; X-axis: Window size; Y-axis: Throughput (ABT, GBN and SR)

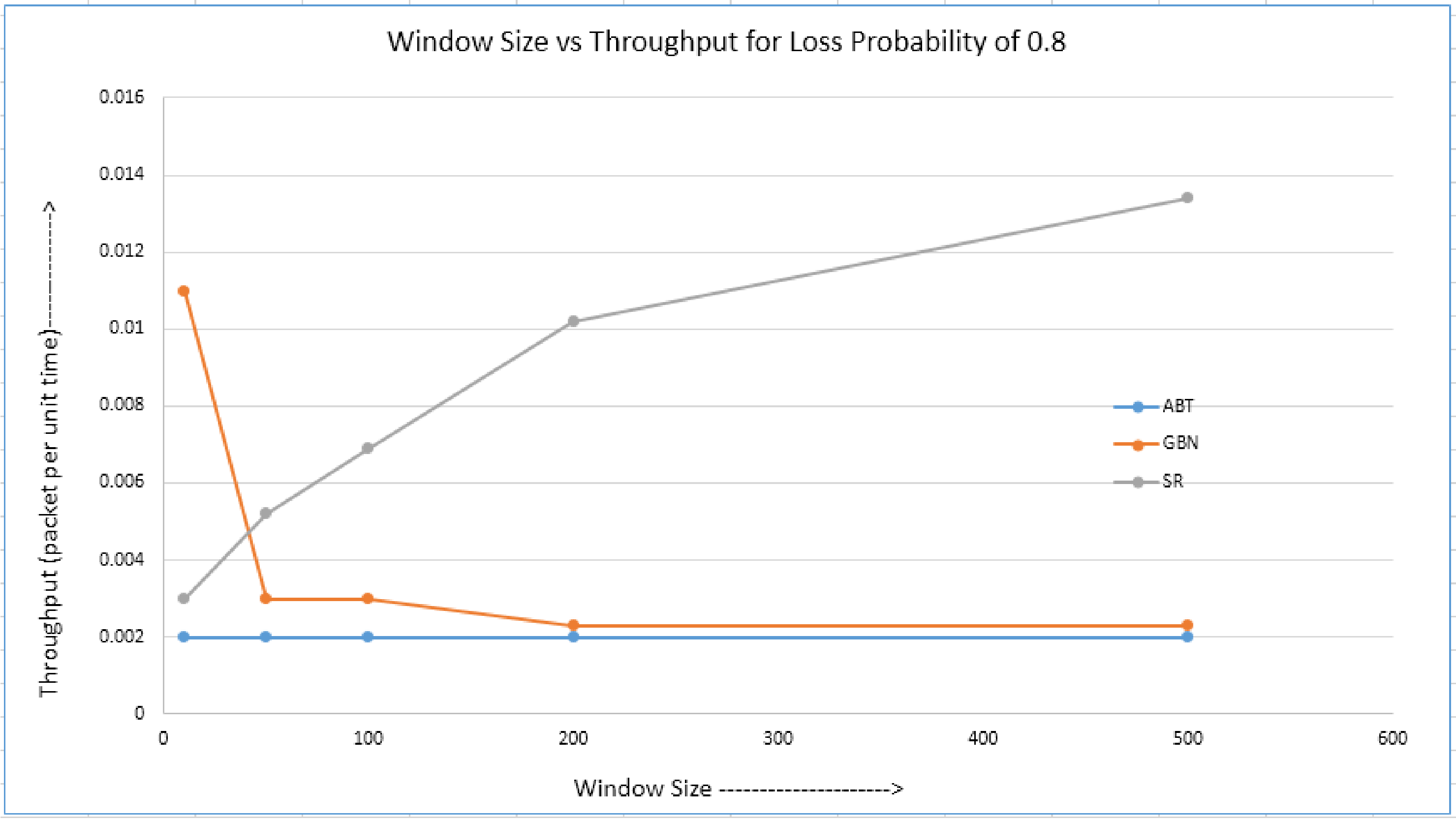


Here kindly note that I have taken average of throughputs obtained for different seed values for a given experiment.

<u>Observation Points based on above graph:</u>

- For GBN, initially the throughput decreases and then remains with constant with increase in window size
- For SR, the performance marginally improves with increase in window.
- For greater window size, performance of SR is most superior. Even ABT gives better throughput then GBN when the window size of GBN is more.

3. Loss probability: 0.8; X-axis: Window size; Y-axis: Throughput (ABT, GBN and SR)



Here kindly note that I have taken average of throughputs obtained for different seed values for a given experiment.

Observation Points based on above graph:

- For GBN, here the throughput is decreasing drastically with increase in window size.
- The throughput of ABT is also very low.
- The throughput of SR is increasing with increase in the window size.
- Also, for relatively smaller window size, the performance of GBN is greater the SR, but for larger window sizes, SR performs exceptionally well as compared to GBN.

Conclusion based on observations of Experiment 2:

- The performance of GBN degrades and performance of SR increases with the increase in window size for greater loss. This is expected behavior as for larger window sizes, GBN leads to a lot of congestion and delay.
- Also when loss probability is less, the throughput is almost independent of the window size for both the sliding window protocols.
- For lower loss probabilities, the performance of ABT and GBN are almost identical and more than that of SR. This is somewhat unexpected since sliding window protocol is expected to give better performance than stop and wait protocol. A reason for that can be the fact that timeout value of SR is much more than timeout value of ABT. But we cannot decrease the timeout of SR as low as timeout of ABT to cope with cases of increased loss and corruption.
- The above mentioned unexpected behavior can be eliminated my choosing a more optimum timeout value or using an adaptive timing techniques for Sliding Window protocols.

Based on the above two experiments, I would like to conclude that:

ABT: Should be only implemented for a lossless communication network and when the available bandwidth is very low. Basically ABT can be implemented for lower level systems.

GBN: Can be implemented for a communication network with relatively high level of losses and should only be implemented with lesser window size.

SR: Can be implemented for a communication network with high loss probability and should be implemented with sufficiently high window sizes (depending on the bandwidth). For SR, the timeout value should carefully chosen or an adaptive timeout scheme is recommended. Also, implementation of SR is complex. So its implementation should be such that runtime is relatively small so that additionally processing delay is not incurred.