Jiawei Zhao
#50207069
April 2, 2017

CSE 589 Modern Networking

Project Assignment 2 Report

I have read and understood the course academic integrity policy.

1. Brief description of of the timeout scheme used in the implementation and why choose that scheme.

Ideally, it's better for us to use adaptive timer for transport layer algorithms, in that case, the RTT is taken into consideration on the determination of our timeout interval since the network condition may vary. However, for this simulator, this situation is unlikely to occur, so instead I use fixed value for timeout interval.
Also, I noticed that for different timeout interval value, the process time for tests in this assignment may vary significantly, it's very obvious for go-back-n and selective repeat. The process time may be faster for higher timeout interval, yet it sacrifices reliability since it takes longer to commit a retransmission if the packet is lost.

2. Brief description of how to implement multiple software timer in SR using a single hardware timer.

For my implementation, I noticed that the absolute time of the simulator is available to us via the function get_sim_time(), so it's very handy to use this function as an indicator of timer for packets. Basically, we can calculate on what time should one unacked packet fires a timer interrupt, which is the time of the time this packet begins transmission plus a timeout interval. But how can we know

when the time of the simulator should trigger a interrupt for certain packets? This is the tricky part for this method, for my implementation, I set a magic timer as 5 units time, which is the average time of a packet from A gets B, and for every 5 units time, it fires a timer interrupt, and check if the current time is greater or equal to any of unacked packets, if so, resend the packet and reset the timer.

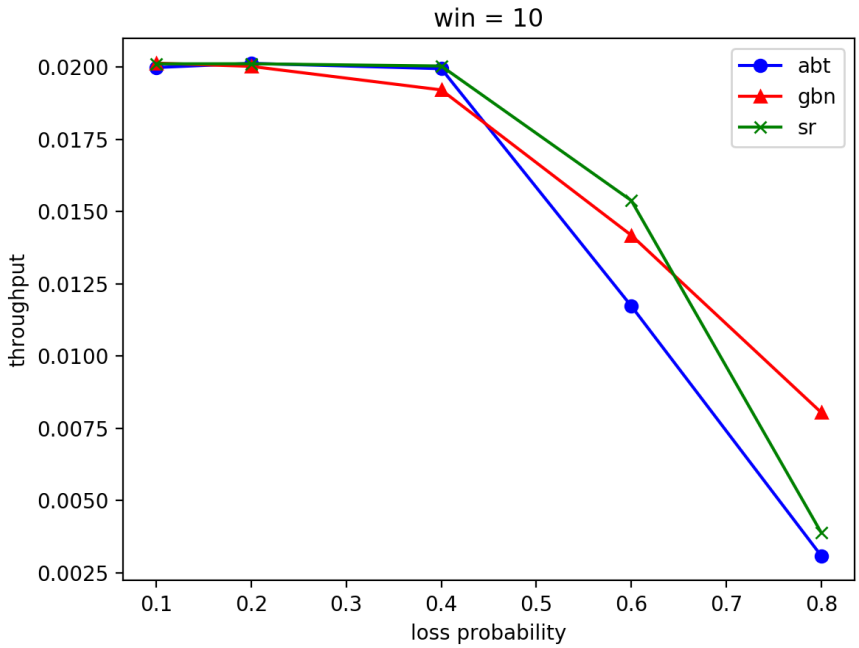
For this to work, I create a new structure srpkt, which has three components: 1) *struct pkt* content, 2) *float* timer and 3) *int* acked. The first part is the content that should be sent to B, the timer stores the absolute time relative to the simulator as when the timer interrupt should fire, and third part works as a flag to check if the packet within sender window has already been acknowledged or not.

The shortcoming of this timer is lack of accuracy. For example, if a packet should fire a timer interrupt in time 16, however, the recent time of timer interrupt may occur in 15, and in which time, the timer should not fire, since 16 > 15. Then it has to wait for next timer interrupt, which in my implementation, is on time 20, and on that time, the actual interrupt time is postponed 4 unit time. A fix to this problem is to maintain the timer in the structure as relative time to the previous timer interrupt, and also change the timeout interval accordingly, which is harder to implement as every time a timer fires, there're a lot of values need to be altered.

3. Performance experiment

3.1. Experiment 1

3.1.1.window size = 10, loss probability varies

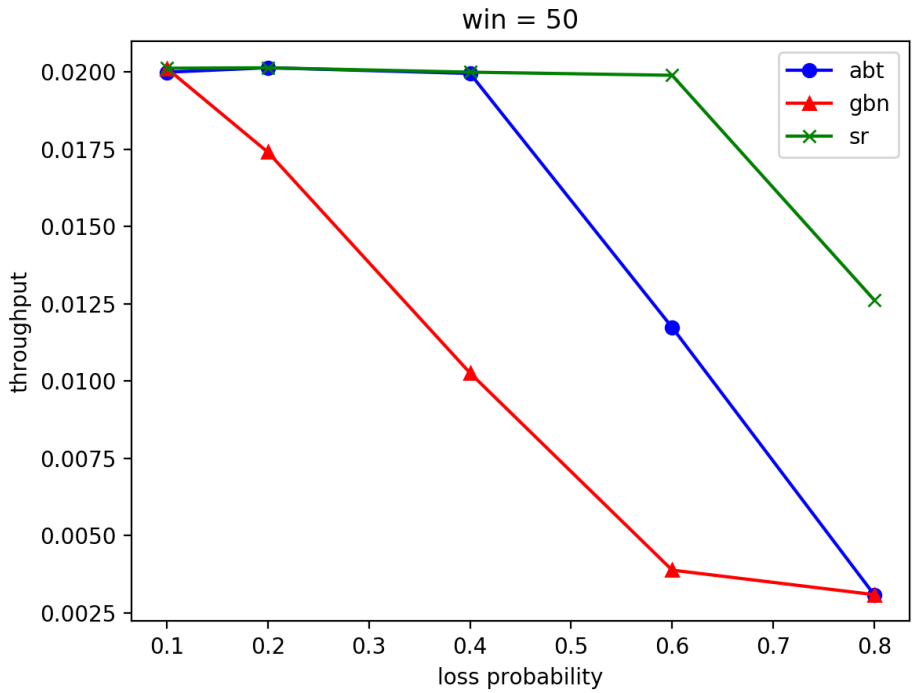


Observation: It's pretty clear that as the loss probability increases, the throughput of all three protocols drops significantly. Also, the throughput drops dramatically after certain loss probability, in this graph, is 0.4.

Explanation: As the loss probability increases, less and less packets are able to make their ways to the receiver side, result in the decrease in the throughput. As for the steadiness of throughput from loss probability from 0.1 to 0.4 and then drastically drops may because the arriving packets number is greater than certain threshold *t* before 0.4, and throughput is determined by *t*, however, after 0.4, the arriving packets become less than *t* , and thus the throughput begins to drop.

Performance comparison: In the beginning, low loss rate make gbn's retransmission redundant, harm the throughput. In the end gbn has a better performance since excessive retransmission is doing something good to the performance since most of the packets are expected to be lost.

3.1.2.window size = 50, loss probability varies



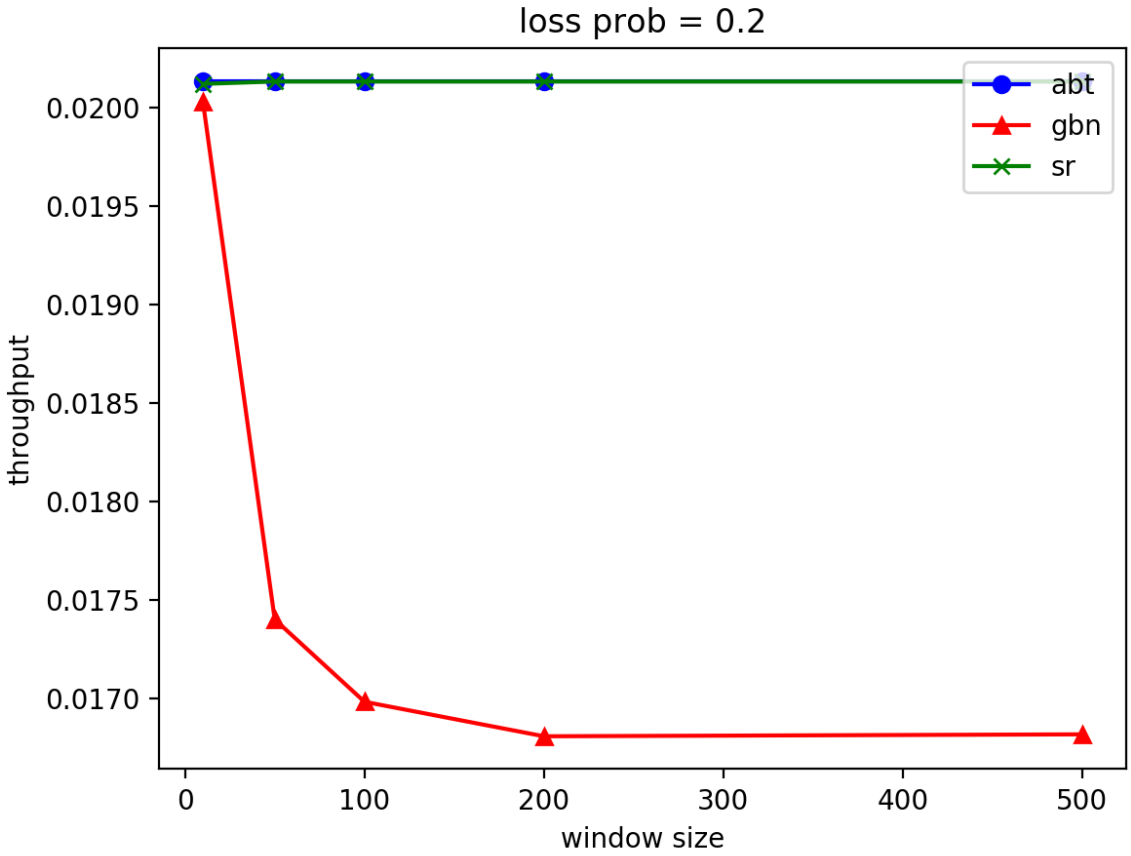
Observation: Rather than start drastic drop from 0.4 probability of loss as abt, gbn starts dropping from the beginning, yet sr starts dropping with a very high loss probability of 0.6.

Explanation: For abt the throughput line is quite similar to the experiment with window size set to 10 since window size has no effect on this protocol. However, for gbn, since the window size is bigger, the retransmission becomes more expensive and hamper the throughput, but notice between 0.6 and 0.8, the drop of throughput slows down which because the increasing loss of packets make the retransmission less expensive. For sr, since it selectively resend lost packets, so it shows a very good throughput rate, and it is less sensitive to the change of loss probability, starts dropping after 0.6.

Performance Comparison: Clearly, sr wins! gbn suffers from its excessive retransmission.

3.2.Experiment 2

3.2.1. Loss probability = 0.2, window size varies

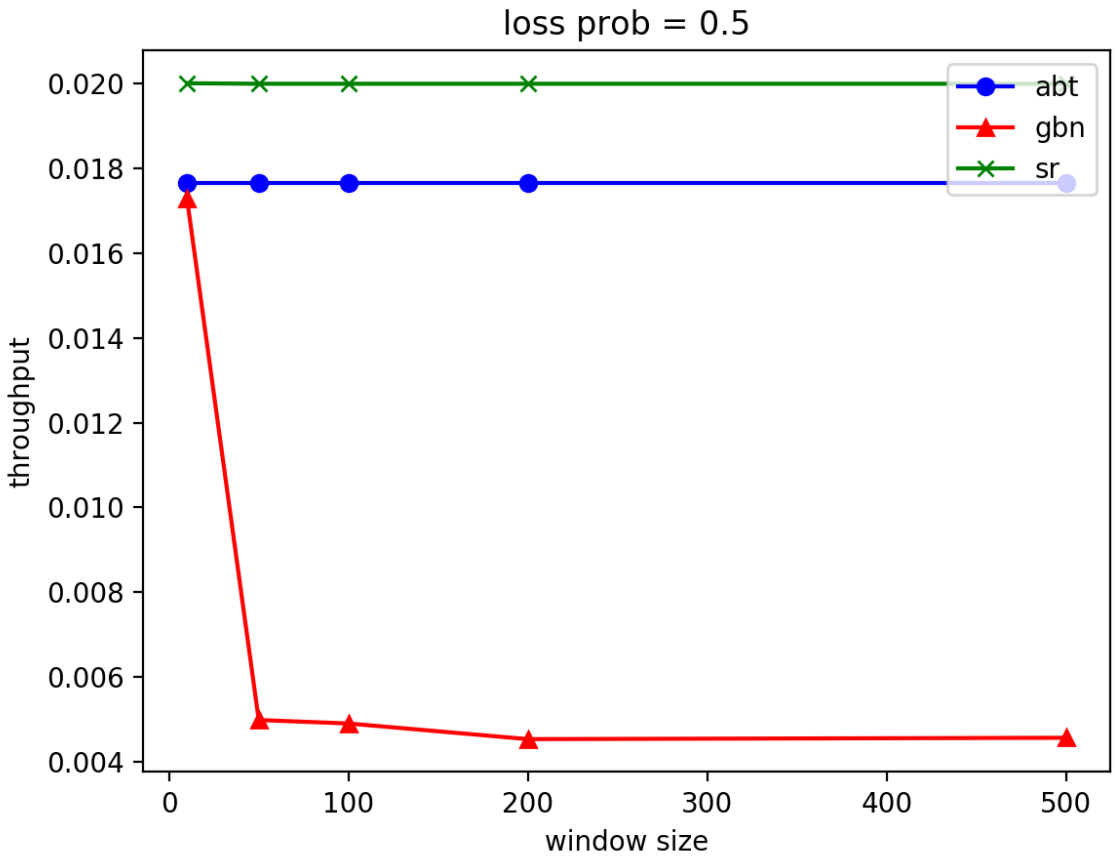


Observation: For abt and sr, the throughput stays in a high value constantly, however, for gbn, the throughput is not bad in the beginning, but it begins to drop instantly after the window size starts to expand.

Explanation: Since the loss rate is fairly low, so the abt can exhibit a performance as good as sr since there won't be too much delay staying in one state(0, 1). As for gbn, again, its excessive retransmission is the major reason responsible for its drop on throughput.

Performance Comparison: gbn has bad performance due to its excessive retransmission, abt and sr has similar performance.

3.2.2. Loss probability = 0.5, window size varies

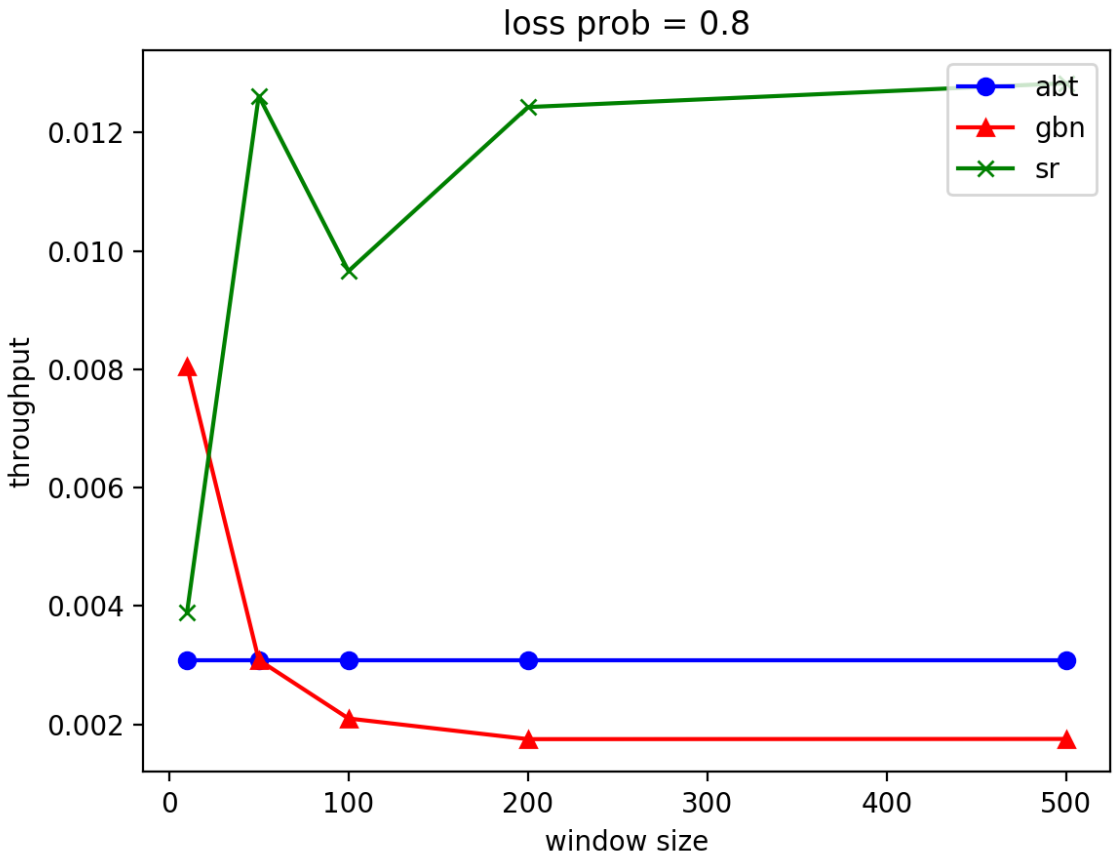


Observation: As we can see abt and sr are immune to variance of the window size, however the performance of abt is not as good as sr in all window sizes. As before, gbn has a not bad performance with smaller window size, but once the window size starts to increase, the throughput drops drastically.

Explanation: For the difference between sr and abt, sr can send multiple packets simultaneously whereas the abt can be stuck in one state if the packet loses during transmission hence sr has a better throughput than abt. Similarly gbn still suffers from excessive retransmission and limit the throughput greatly.

Performance Comparison: sr has the best performance while abt has a also not bad performance, however, gbn's performance becomes very bad when the window size is big.

3.2.3. Loss probability = 0.8, window size varies



Observation: In the beginning, gbn actually has the best throughput, but as the window size increases, sr gradually becomes the best protocol that has the highest throughput, as for abt, it's very steady but its throughput is also very low. Gbn becomes the worst when window size is bigger than 50.

Explanation: Since the loss probability is 0.8, which means most of the packets A sends will be lost, so when the window size is small, excessive retransmission like gbn actually improves the performance. The low throughput of abt is because the loss of packets may result in no progress for abt. Gbn, again, suffers from excessive retransmission when the window size is large enough.

Performance Comparison: For high loss rate transmission, sr has a significant advantage over all three protocols.