CSE 589

Programming Assignment 2

Report

(i) Academic integrity statement

I have read and understood the course academic integrity policy located under this link: http://www.cse.buffalo.edu/faculty/dimitrio/courses/cse4589_f14/index.html#integrity

(ii) Timeout scheme

For the GBN protocol, while selecting the TIMEOUT value the general observation was that a for a given window size, seed, and corruption probability, the throughput increased if a smaller timeout value was chosen, when the loss probability was small. This was probably because smaller timeout meant more retransmissions per unit time and at lower loss probability, frequent retransmissions helped move the window forward and get a higher throughput.

But for larger values of loss probability, the throughput began to decrease, when the timeout was small. This was probably because although smaller timeout meant more retransmissions per unit time, at higher loss probability a packet is lost more frequently, and the window couldn't move forward, decreasing the throughput.

Thus, an optimum timeout value existed, and after experiments was chosen to be 25.0,

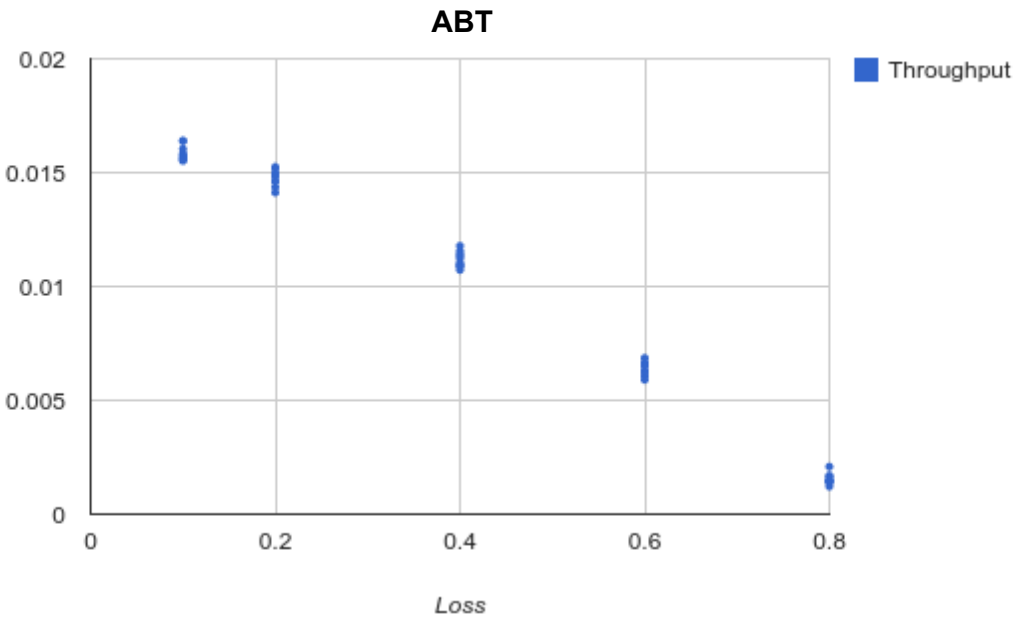
(iii) Multiple timers implementation in SR

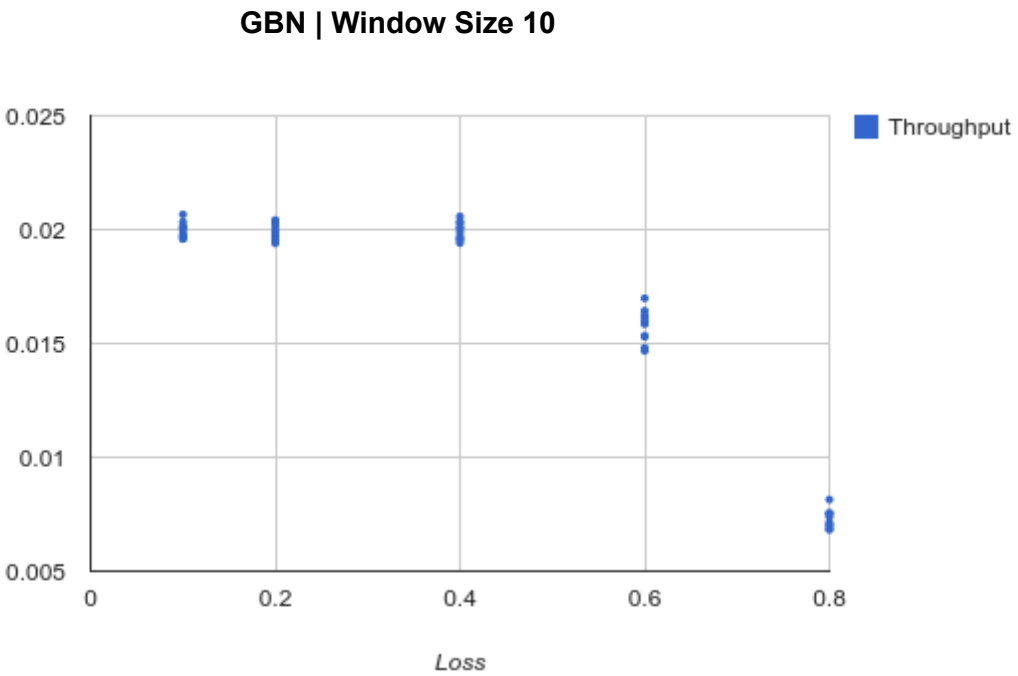
Only as single hardware timer was given, but one timer was required for each transmitted packet in SR. The basic idea is, whenever the timer goes off, we stop the timer and start it again such that it times out at the time the packet with the next least endtime is supposed to timeout.
The scheme implemented was as follows:

1. Maintain an array `endtime_a[]`, that stores the time at which each packet is supposed to timeout. Whenever a packet with a given sequence number is received in A_output, update the endtime_a[] array at the appropriate index. If a timer has not been started already (identified by a flag `timerison_a`), then it is started, meant to timeout at current `time + TIMEOUT`.
2. When an ACK is received for a packet in `A_input`, search through the `endtime_a[]` array to find if the ACK was for the packet with the lowest endtime value. If yes, stop the timer and start it again for the packet with the *next* least endtime. But now, the timer should time out at the value specified by the entry in the array.
3. When the timer times out and `A_timerinterrupt` is called, search through the `endtime_a[]` array to find the minimum value and its corresponding index, because this is the packet for which the timer timed out. Retransmit that particular packet. Now find the *next* minimum in the array and start the timer for that packet, such that it times out at the value specified by the entry in the array.

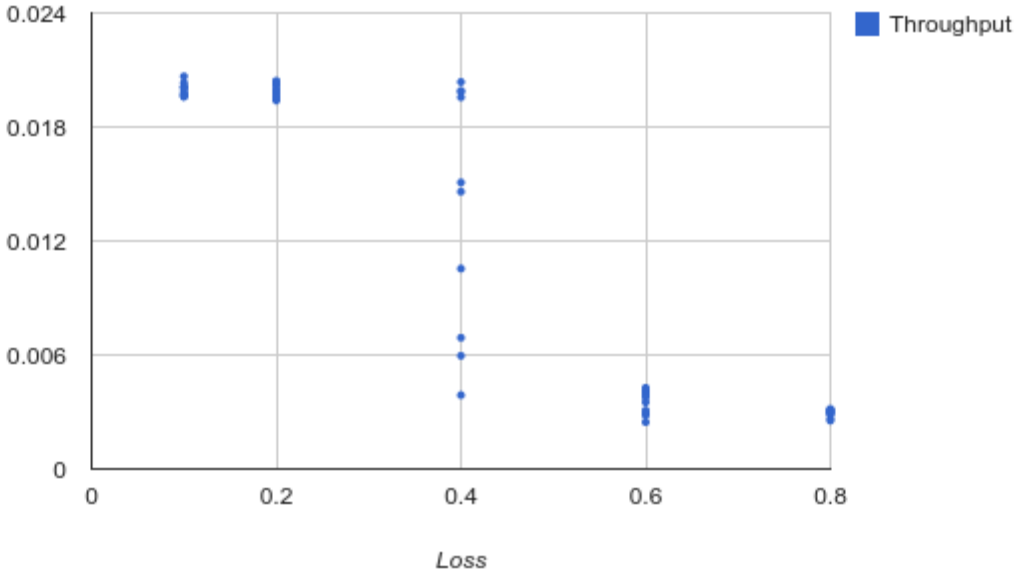
(iv) Experiment 1

In each of the graphs below, the throughput is plotted as a function of loss probability, for each of the 3 protocols. For a given loss probability, different values of throughput are obtained via different seed values. For the GBN and SR protocols, two window sizes 10 and 50 are used.

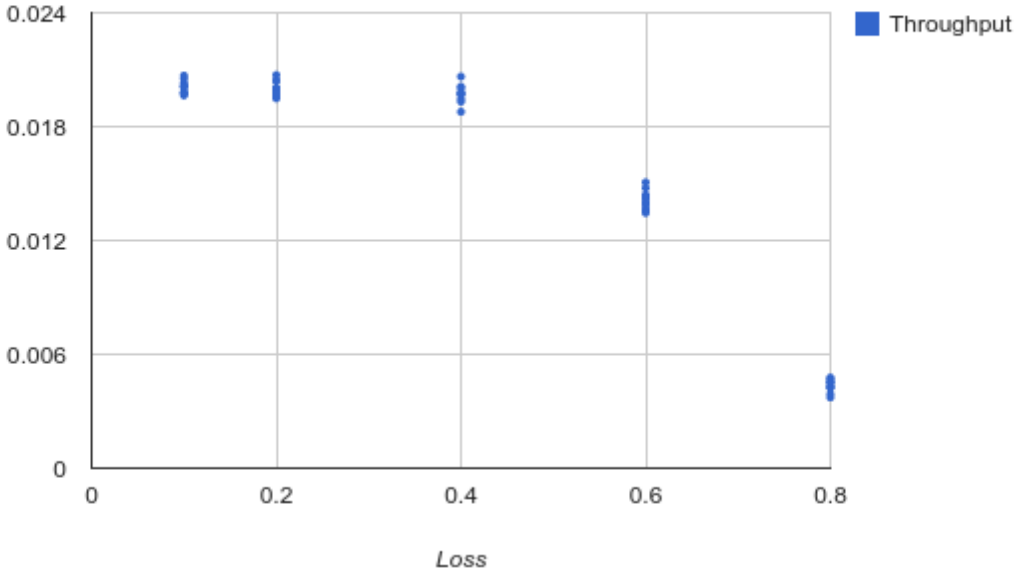




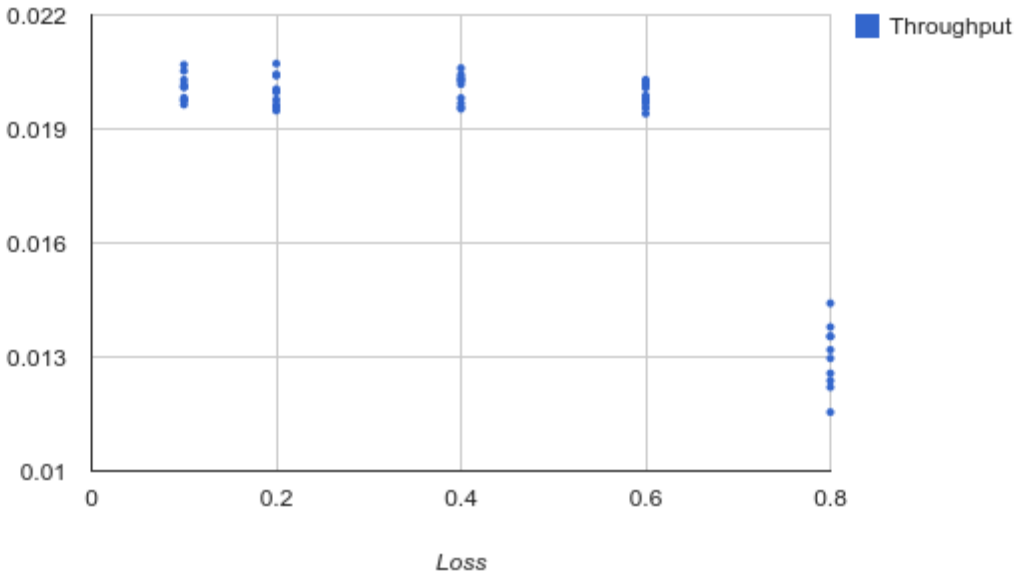
GBN | Window Size 50



SR | Window Size 10



SR | Window Size 50



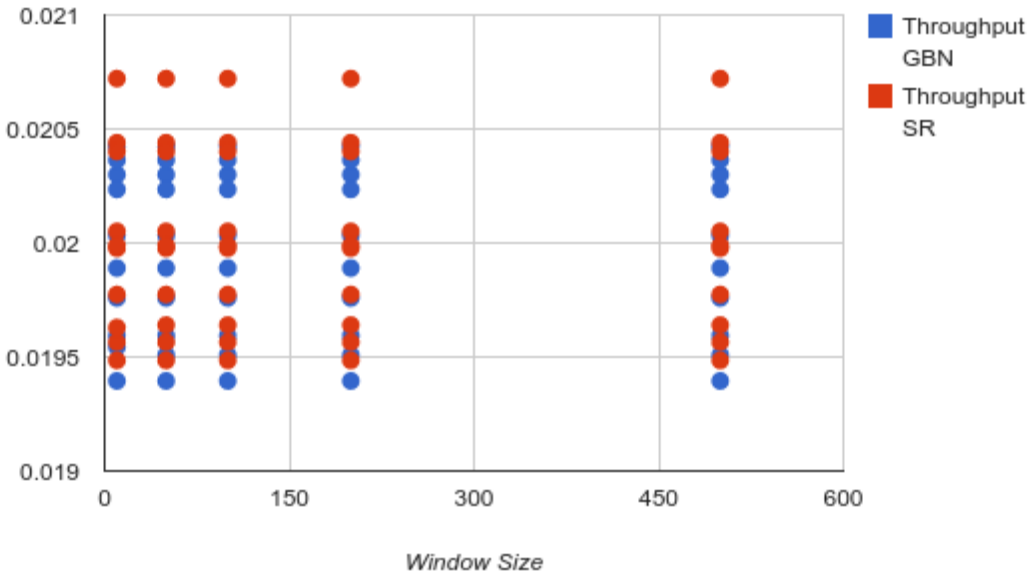
As seen from the graph, For the ABT protocol, as the loss probability increases, the throughput steadily (almost linearly) decreases. For the GBN protocol, the throughput is steady until until a certain loss probability (0.4 here), after which the throughput decreases steadily with increase in loss probability. For the SR protocol, the throughput is maintained steady for even higher loss probabilities. When it begins to decrease, it seems to decrease exponentially.

As throughput is a measure of performance, the conclusion that could be drawn here is that the SR protocol performs the best as it maintains a high throughput even for high loss probabilities. After this comes the GBN protocol, as it does give a better throughput that the ABT protocol, for lower loss probability, but does not maintain that performance for higher loss probabilities like SR. The ABT protocol seems to give the worst performance.

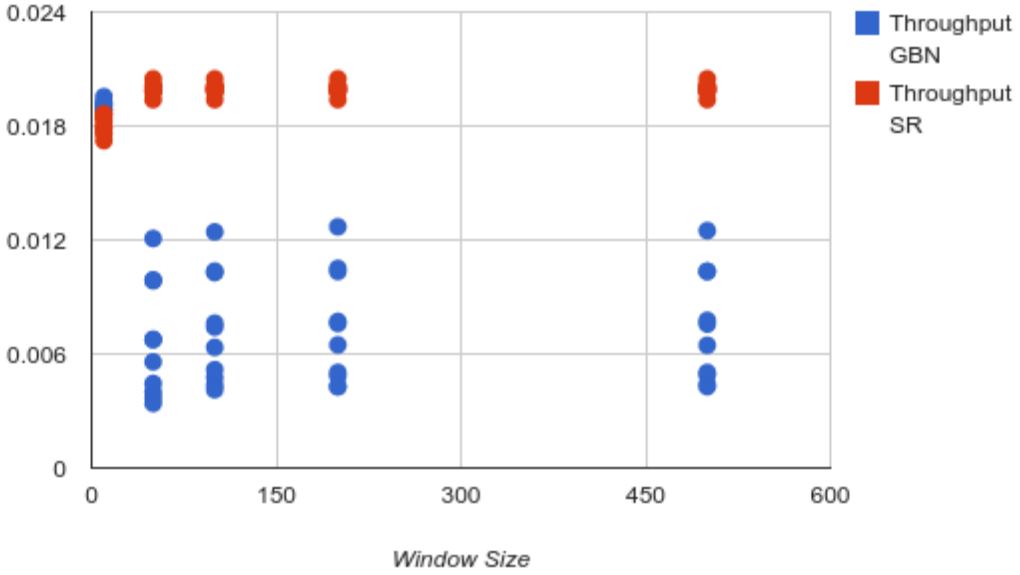
(v) Experiment 2

In each of the graphs below, the Throughput is plotted as a function of Window Size {10, 50, 100, 200, 500} for the 2 protocols SR and GBN. For a given window size, different values of throughput are obtained via different seed values.

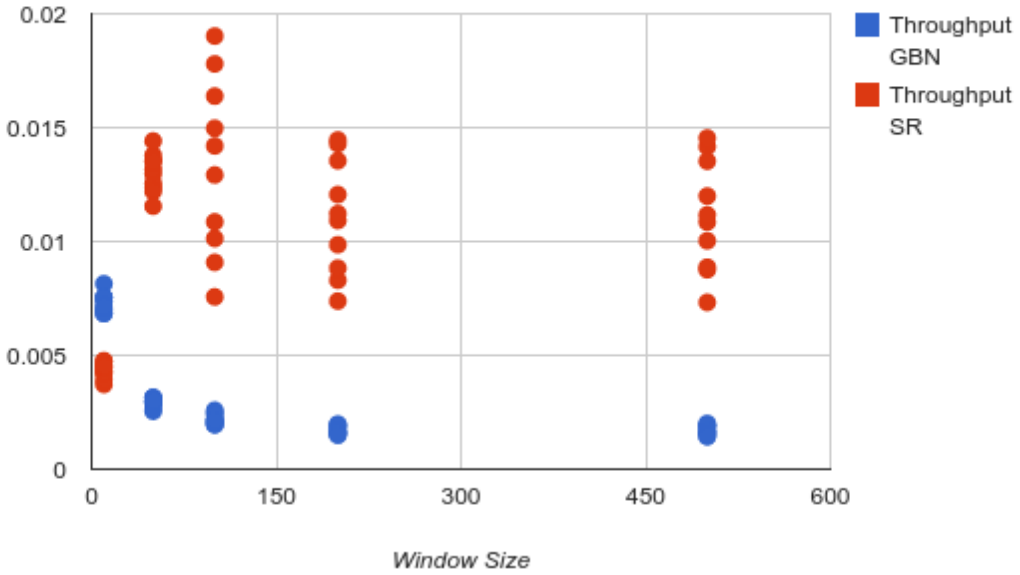
Loss probability 0.2



Loss probability 0.5



Loss probability 0.8



We expect that for smaller window sizes, GBN should perform better than SR. As GBN uses cumulative acknowledgement, any ACK greater than or equal to the base will move the window forward, while SR has to selectively acknowledge each packet, and it takes time to move the window. This is good for small windows, where packets in transit are less. In case of a large window, as number of packets in transit is more, SR should perform better, as selectively acknowledging and retransmitting packets is intuitively better. In GBN, we retransmit the whole window size if the sender times out (in case or loss or corruption), which will very wasteful.

As seen from the graphs, for a low loss probability of 0.2, the throughput for both protocols is almost the same, varying between 0.19 and 0.21, for all the window sizes. For a higher loss probability of 0.5, the throughput of SR is definitely better, for all the window sizes greater than 10. The throughputs of SR range from 0.018 to 0.024 for all window sizes greater than 10, whereas they are quite low for GBN, in the range 0.001 to 0.015. For an even higher loss probability of 0.8, like for 0.5, SR performs significantly better for window sizes greater than 10.

Thus our observations seem to match what we expect.