CSE 589 - Programming Assignment 2
Implementing Reliable Transport Protocols

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

TIMEOUT SCHEME

In **Alternating Bit Protocol** it was observed that by increasing the time out we experience reduction in throughput. Timeout was changed from 10 which is the average RTT to higher values like 70. Throughput was maximum for 10. This could be because by increasing the timeout we are not utilizing the channel completely. Hence, for timeout a constant value of 10 has been used.

Similarly in **Go-Back-N** it was observed that with increase in timeout the throughput reduces. With increase in window size it was expected that the throughput would decrease because for long window sizes many packets would be transmitted and would cause delay, but the throughput increased with shorter timeout till the tested window size of 500. So, a timeout of 10 has been chosen.

In **Selective Repeat** it was observed that for some seed values the minimum timeout was not resulting in the protocol running for a long time.
For this the timeout scheme applied is that we we're counting the no. of timeouts for which the base packet is not getting ack'ed, when this count reaches a particular limit, we increase the timeout and the timeout count limit by a certain amount. This goes on till a specific upper limit for the timeout, which was decided as a high enough timeout value. The scheme could be extended further to decrease the timeout if the packet count is lower that a limit. But in the current scenario it didn't seem necessary because the system didn't seem to come to the condition where it would reduce the timeout again.

This timeout scheme works well for different window sizes. It was expected that for larger window sizes, the upper limit for the timeout may need to be increased. But good throughputs were obtained without much change in the values.

Implementation variables of SR timeout scheme –

TIMER_UPPER_LIMIT - upper limit to which timeout can be increased

TIMER_PACKET_COUNT_LIMIT - count limit for the no. of timeouts for which a packet can remain in the window before increasing the timeout

TIMER_COUNT_INCREMENT - amount by which the count is incremented on reaching the TIMER_PACKET_COUNT_LIMIT limit

TIMER_INCREMENT - amount by which the timer is incremented on reaching the TIMER_PACKET_COUNT_LIMIT limit

Faster and better throughput was observed after implementing this timeout scheme.

<u>IMPLEMENTING MULTIPLE SOFTWARE TIMERS WITH ONE HARDWARE TIMER</u>

For implementing multiple software timers, a struct with the name ‘timer’ has been implemented. This structure maintains a vector (of struct seq) of size equal to the window size called ‘priority’. This vector is used to store the relative times of the packets which enter the window w.r.t. a initial time when the timer is started.

‘priority’ – As packets receive acks they will be removed from the vector. Newer packets will be added to the vector and the vector will be sorted so that the packets with the closest timeout are approached first.
‘initial_time’ – This is the reference time which is used to calculate the time the packet arrives in the window from the time the first packet arrived
‘index’ – index of the current packets timer in the priority
‘entity’ – This represents the entity whose timer is being called and is set through the constructor
‘seq’ – struct which stores the relative time and the sequence of the packet which is in the window

Functioning –

1. The general logic used in the timer is that the sum of all the smaller relative time differences is TIMEOUT because they have been calculated by taking the mod TIMEOUT. Hence, the timer will execute smaller intervals for each packet and by the time it competes one round and comes back total time before a packet times out would have been TIMEOUT.
2. For triggering the timer for a packet regularly at intervals of T, we consider the whole time range to be divided into intervals of T. By measuring the relative time of a packet we’re measuring the point at which the packet arrives within the interval T (i.e. t for packet arrival within T = (arrival time - initial time) % T). By triggering the timer for each packet at the same point within T it will have a difference of T between its triggers.

```
        Time range
|-----------T-----------|-----------T-----------|-----------T-----------|-----------T-----------|-----------T---------

              t1|-----------T-----------|-----------T-----------|-----------T-----------|-----------T-----------|
```

   Packet arrives at t1 within the first T, when we trigger it at the same position within the next T interval i.e. 2T – t1, which we are calculating the above mentioned formulae, we can get a timeout of T. This condition which is for one packet is extended to all packets.
3. Priority array is empty: When a new packet arrives, the time when the timer is started is the same local time. So the reference time will be added as 0. The A timer will run for the whole TIMEOUT value for this packet, as required.
4. When a new packet comes in the time difference between the first packet (initial time) and this one is recorded as the reference time. When the timer stops running for the first packet, it will be run for this reference time because that is the time period left for the second packet to complete TIMEOUT.
5. Similarly all packets which enter the priority vector are added with their reference time’s i.e. the time after the first packet. But when the timer is run for them it will be run for the difference between their reference time and the reference time of the element before them.

```
-----------------------------------TIMEOUT---------------------------------------
T0-----T1 -------T2 -------T4 --------------T5----------------------T6----------
```

6. In the above diagram T1 – T0 is the reference time for packet 1. After the total timeout run for the first packet the timer will run for this time which will complete TIMEOUT for packet 1, and for packet 2 the timer will run next for time T2 – T1. For the first packet the time is calculated as TIMEOUT – T6 +T0. In case the first packet receives ack and is removed this formula will provide the correct timeout for T1 packet.
7. **Handling packets which may have timeouts at the same time:** 2 or more packets can have the timeout at the same time if they arrive at some time > TIMEOUT after the initial packet. E.g. If a packet arrives at time T = 0 and for a timeout of 5, packets arriving at T = 5, 10 will have timeouts together. For such packets when we calculate the relative time with modulus of TIMEOUT, we get the same time. When timeout occurs all these packets with the same relative times are sent out from the sender.

Functions –
set_timer – this function is passed the sequence no. of a packet and it adds the packet with the reference no. in the priority array in the correct order.

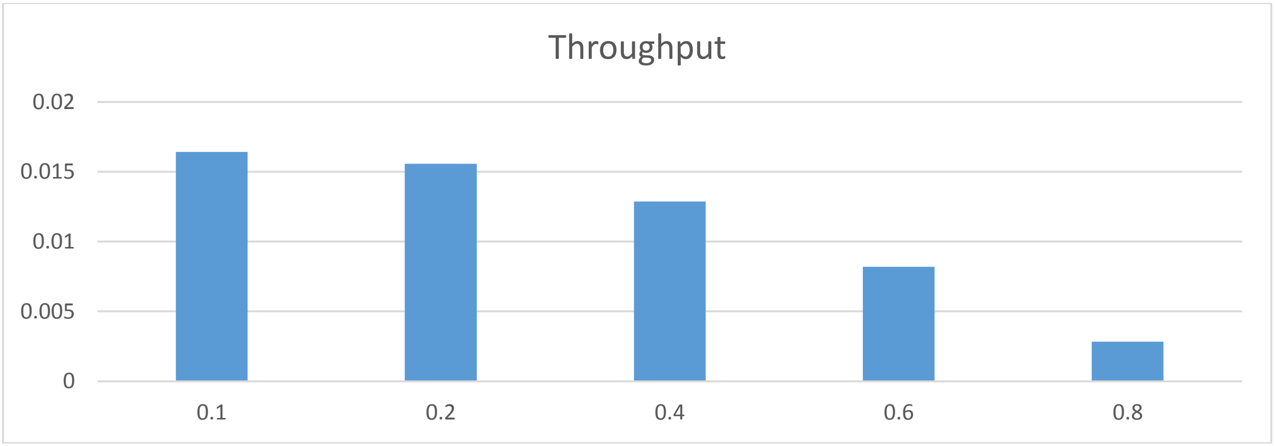
stop – this function is used to stop the timer for a particular packet. It's passed the sequence no. for the packet and the sequence no. is removed from the priority array.

next_timer – this function is called to start the timer for the next packet which has the closest timeout.

get_timedout_seq **–** this function returns all the sequence no.'s which have timed out

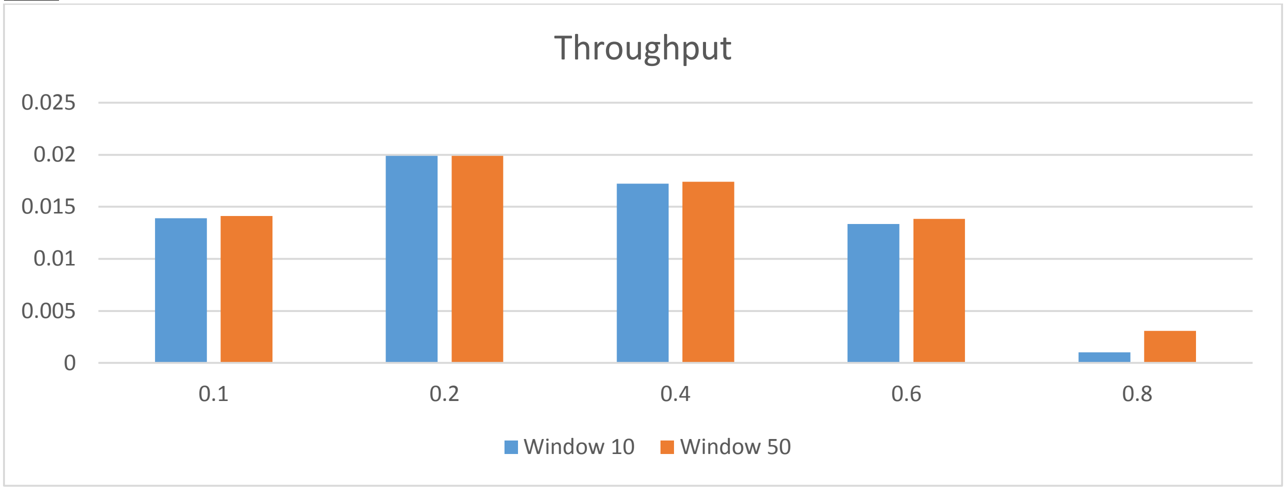
<u>Experiment 1</u>

<u>ABT</u>



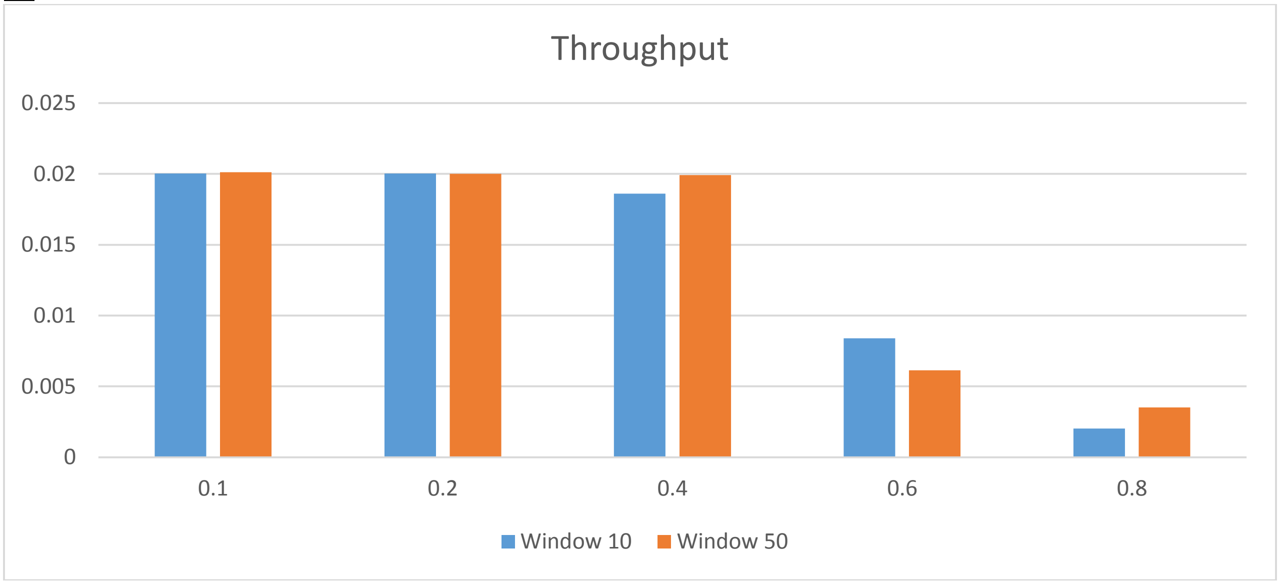
As we can see the throughput for ABT increases with lesser loss probability, which is as expected.

GBN



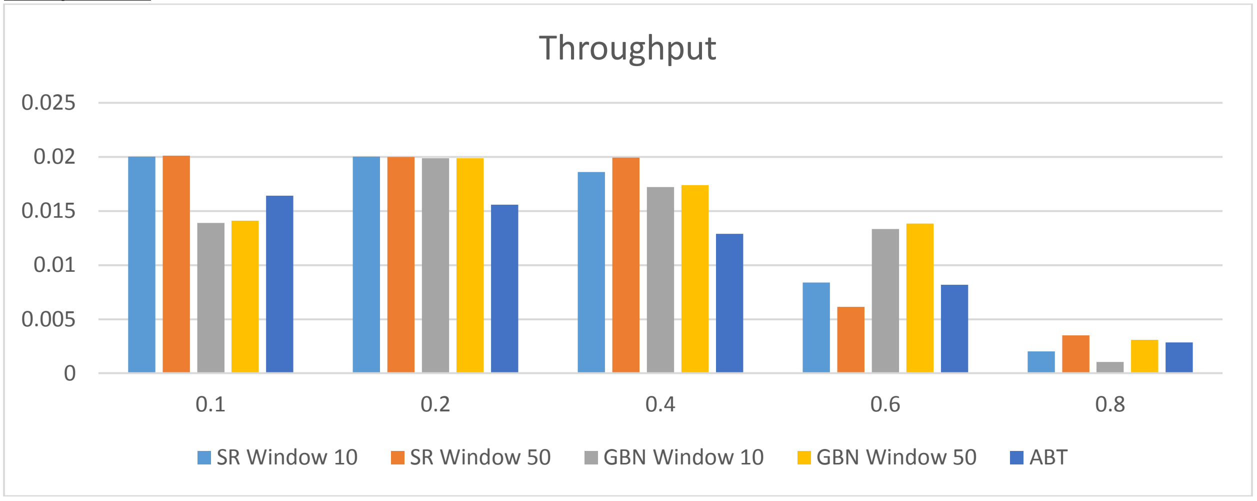
In GBN it was expected that the throughput would be maximum for loss probability of 0.1, but the throughput is maximum for 0.2 loss probability. But as expected, with increase in window size we can see that the throughput has increased for loss probability of 0.8.

SR



Here, we can see that with higher window size the throughput increases generally. This is expected as the receiver and sender can buffer more data and the channel can be used more effectively by sending more packets across.

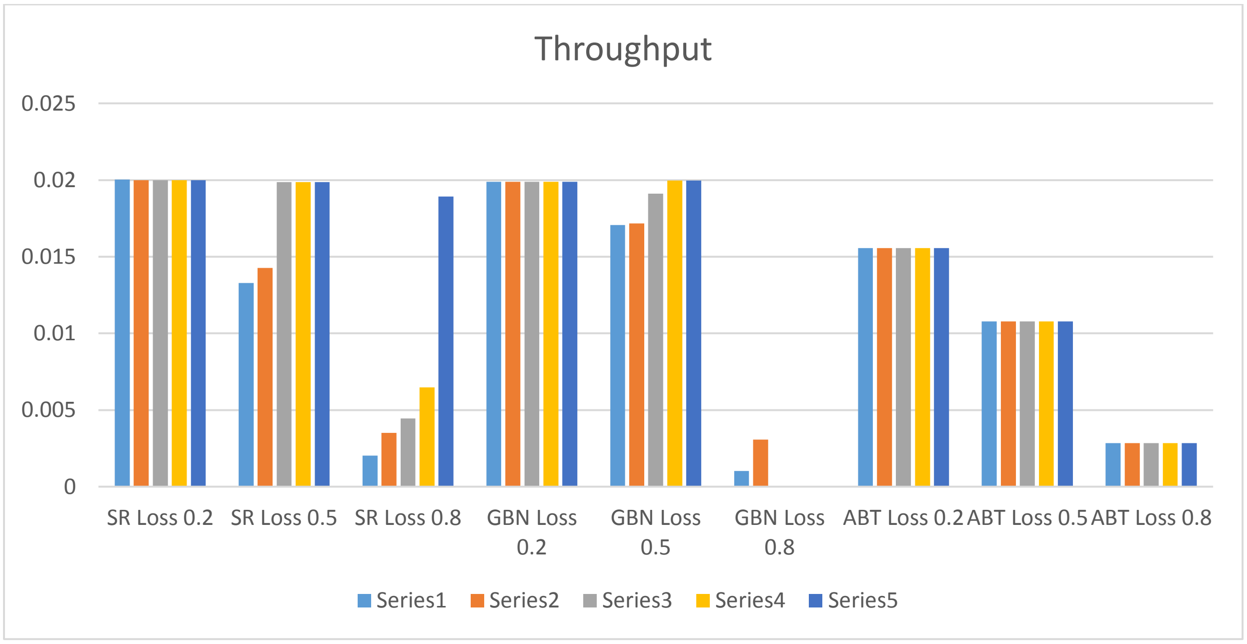
Comparison



Loss	SR Window 10	SR Window 50	GBN Window 10	GBN Window 50	ABT
0.1	0.02002	0.020109	0.013908	0.014119	0.016419
0.2	0.020033	0.020005	0.01989	0.019882	0.015577
0.4	0.018595	0.019923	0.017217	0.017394	0.01288
0.6	0.008398	0.006139	0.013345	0.013825	0.008197
0.8	0.002029	0.003518	0.001035	0.003079	0.002843

We see here that for a fixed size of window SR performs better than GBN which performs better than ABT in general.

Experiment 2



Series 1 – Window 10
Series 2 – Window 50
Series 3 – Window 100
Series 4 – Window 200
Series 5 – Window 500

Window	SR Loss 0.2	SR Loss 0.5	SR Loss 0.8	GBN Loss 0.2	GBN Loss 0.5	GBN Loss 0.8	ABT Loss 0.2	ABT Loss 0.5	ABT Loss 0.8
10	0.020033	0.013298	0.002029	0.01989	0.017068	0.001035	0.0155765	0.010783	0.002843
50	0.020005	0.014269	0.003518	0.019882	0.017176	0.003079	0.0155765	0.010783	0.002843
100	0.020005	0.019869	0.00446	0.019882	0.019114	0	0.0155765	0.010783	0.002843
200	0.020005	0.019869	0.006477	0.019882	0.019964	0	0.0155765	0.010783	0.002843
500	0.020005	0.019869	0.018921	0.019882	0.019964	0	0.0155765	0.010783	0.002843

*The observations marked in red have not been obtained
The observations marked in yellow is for only the first seed run
The observations marked in blue are the average of the first 9 seed runs

From these we can see that similar to Experiment 1, SR for a fixed window size performs better than GBN and ABT.