Programming Assignment 2 report

I have read and understood the course academic integrity policy.

-Puneeth Pepalla (50206906)

Timeout Scheme:

Abt:

Timer in abt is set to 15 time units because in the description it is said that a packet takes 5 time units to reach the other side of the medium when there is no traffic. So minimum timeout that needs to be set is 10 units. To be on the safe side and taking congestion and traffic into consideration I set it to 15 time units. This shouldn't be set too high. So it should be a trade-off value.

Go-Back-N:

In go back n, since we send too many messages in parallel. This leads to congestion in the medium. The magnitude of the congestion depends on the window size. So timer should definitely be higher than abt since gbn has more congestion that abt which has less congestion or no congestion. So I set the timer to 60 time units. I changed this value couple of times but it gives optimum results for 60. I tried setting the values for 15,20,70,100 etc. Even I tried some adaptive timers like loss*12 and winsize*12. So after many trial runs with different values for timeouts 60 seemed to give better results and more throughput.

Selective repeat:

In selective repeat, timer should be set for every single packet transmitted. So we need multiple timers. But in this case we are given only one hardware timer. So we have to make use of this one timer to set timeout values for all the packets. So I used a small tweak in my code to use this one hardware timer efficiently for all the packets. The idea is to set the timeout for the packet which is likely to expire first. That is for example , the first packet has to expire first and then the second packet and so on. . So in order to keep track of time when the packets arrived we make use of the routine that is given to us. We use get_sim_time() to store the timeout values for different packets. So as soon as the first packet is received from application layer, the timeout value for the first packet is set to get_sim_time()+t where 't' is the amout of time before which the ack should be received after the packet is sent. I chose this value of t as 30 since it gave optimum results than for other values of t. So if the packet arrives at a time 23 then acknowledgement for the packet should be received before 23+t. We maintain a list of

packets with their virtual timers after being sent to layer 3. So everytime a new packet is sent to layer 3 the timeout is set to get_sim_time()+t and added to the list of packets that are in the yet to be acknowledged. But the timers keeps running for the first packet in the list.Then there are three cases that can happen:

Case-1: The ack for the first packet can be received before the timer is expired. In this case we just stop the timer for the first packet and start timer for the packet which arrived after that which in this case is second timer. We do this be seeing the timeout value for the second packet that has already been set and subtracting get_sim_time() from that value. So the timer value for the second packet or the next packet will be timeout value for the next timer minus get_sim_time().

Case 2: If timeout happens, then the first packet is retransmitted and new timeout value for that packet will be get_sim_time()+t and is removed from the front of the list and is pushed to the back of the list with the new time out value.

Case 3: if some other ack is received other than the first packet in the list then simply we iterate the entire list and remove that packet from that list and the timer still keeps running for the first packet in that list.

Main variables and data structures used in the code:

Abt:

Struct pkt previous: to store the packet that has recently been transmitted and waiting for the acknowledgement. This is stored because in case if it times out the packet has to be retransmitted.

Vector<struct pkt> buffer: This is used to store the packets in buffer that are received when a packet is already waiting for acknowledgement.

Int State: this is used to see in which state the control is when a packet is received from application layer. 1 means a packet that is sent is yet to be acknowledged and 0 means there are no packets that are waiting for acknowledgement.

Int Seq: This is used to keep track of sequence numbers of packets that are sent. This can only be either 0 or 1 in abt so everytime a new packet is received the value of this seq is updated as seq++%2.

Go-Back-N:

Int Window-to keep track of window size.

Int winsize- This is initialized in the A_init() function when the user enters the argument for window size.

Int Seq-to see the current sequence number of the packet.

vector<struct pkt> buffer: This is used to store or buffer the messages that are entered when the window is full.

vector<struct pkt> previous: This is used to store the messages that are sent and waiting for acknowledgement. The size of this will never be greater than window size. Because at a given time the utmost number of packets that can wait for an acknowledgement is window size.

Selective repeat:

Struct Node: In selective repeat I created a struct to and maintained a list of that structs to store timeout values for different packets. The data structure used in node and it contains two member variables **timer** which is of type float and **seq** which is of type int. the value of the timer is timeout value for a specific packet which is get_sim_time()+t as mentioned in the timeout scheme for selective repeat and seq is the sequence number of that packet. And name used for that list pq which is mentioned below.

list<node> pq: This is used to store the list of structs created to keep a track of timers. Everytime a new ack is received a struct from this list is removed which seq's value matches with the acknowledgement number of the received packet.

vector<struct pkt> buffer: This is used to store or buffer the messages that are received from application layer when the window is full. This cannot be greater than senders window size. This is nothing but the senders window.

vector<struct pkt> previous: This is used to store the messages that are transmitted to the sender's transport layer and waiting for an acknowledgement.

vector<struct pkt> rbuffer: Here in selective repeat even the receiver has to maintain a buffer. So this is nothing but the receiver's window. The senders window and the receivers window are of the same size. So this cannot be greater than window size of the sender.

Experiments

Experiment 1:

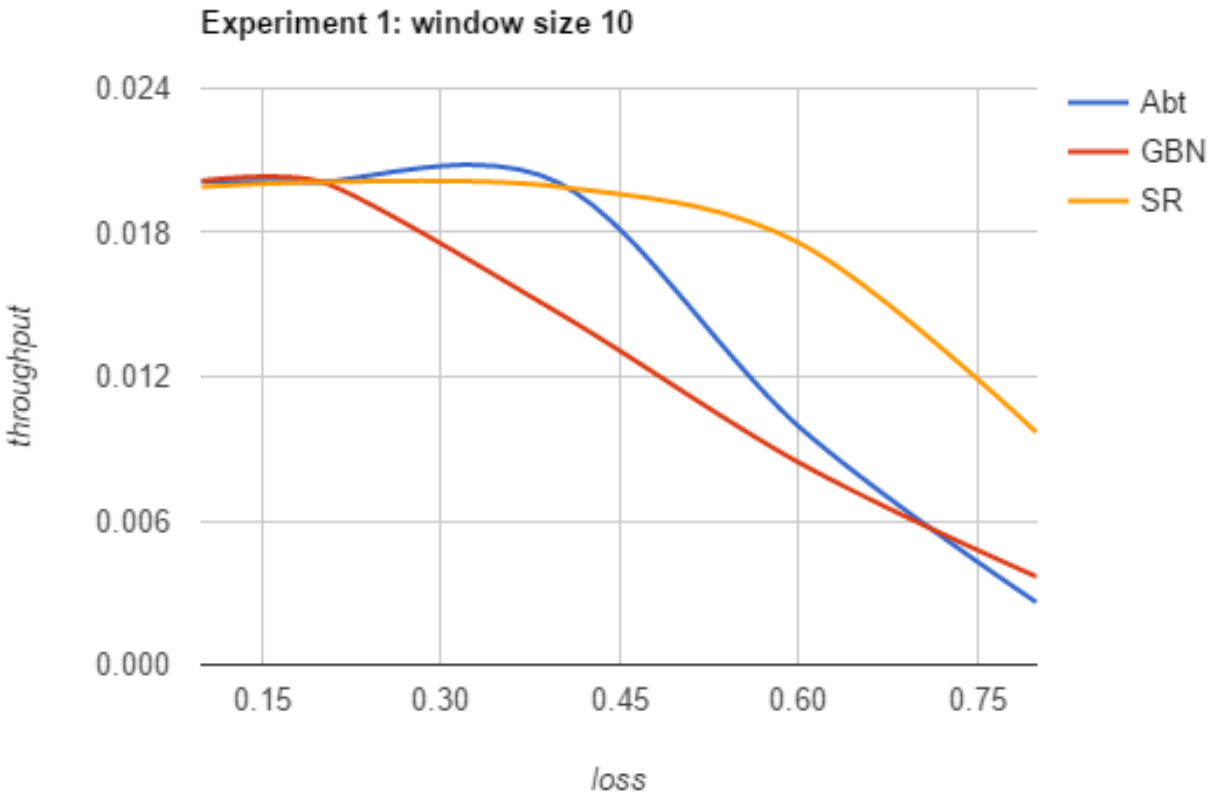
Settings for exp 1: number of messages:1000 time between two messages:50 corruption : 0.2

Case 1: Window size:10 and loss probabilities of 0.1,0.2,0.4,0.6,0.8

Values obtained: These values are calculated by taking the average of all the ten seeds by using the script run_experiments.

Loss/throughput	ABT	GBN	SR
0.1	0.0201194	0.0201576	0.019906
0.2	0.0201072	0.0201197	0.0201027
0.4	0.0200582	0.0146333	0.0199058
0.6	0.0099556	0.0084462	0.0175967
0.8	0.0026058	0.0036719	0.0096834

Graph:



Observation:

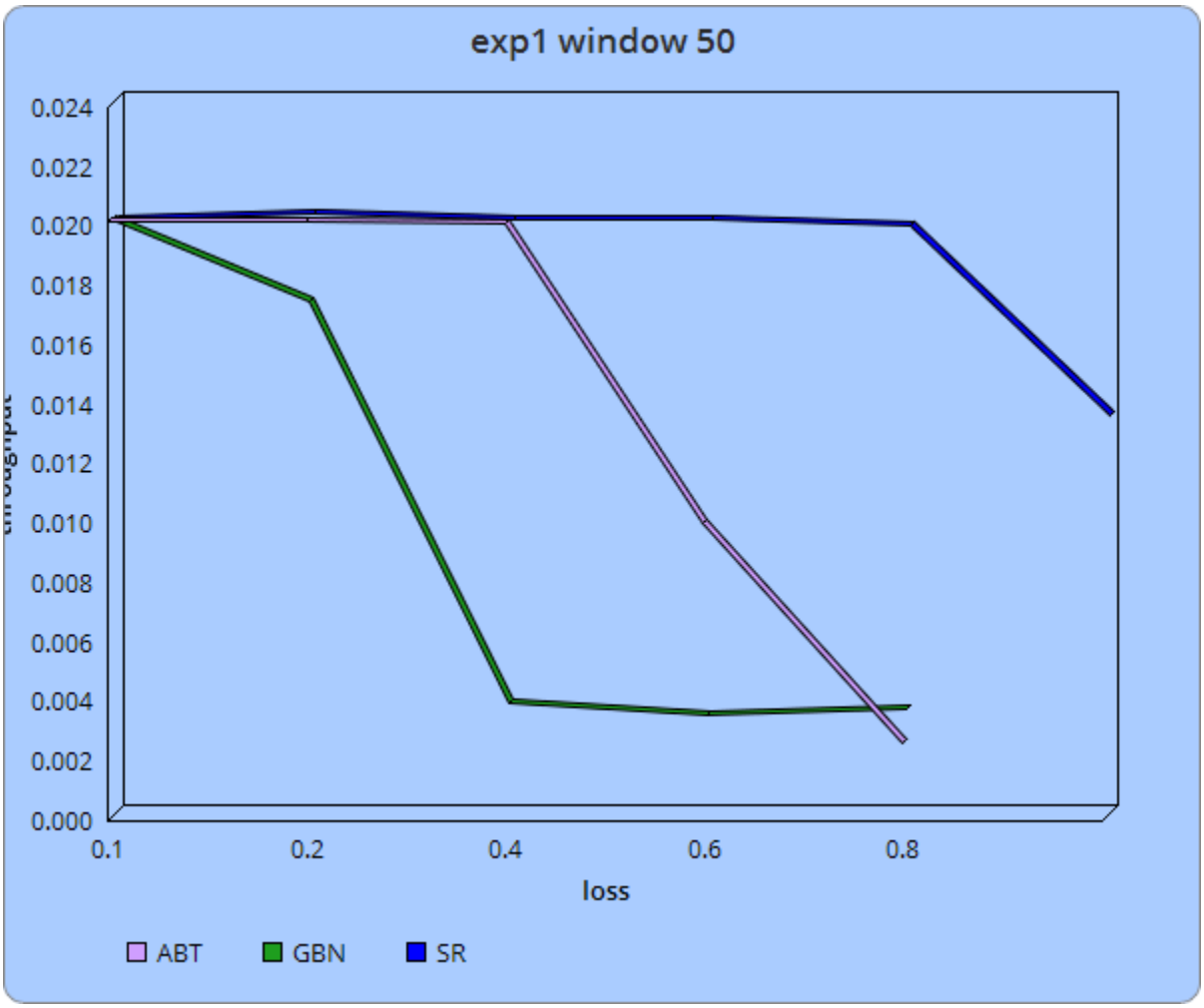
From the graph obtained above we can observe that at the beginning we have almost equal throughput for all the three protocols but as the loss keeps on increasing, sr has higher throughput. So selective repeat performs better than abt and gbn when loss values are high resulting in higher throughput.

Case 2: Window size:50 and loss probabilities of 0.1,0.2,0.4,0.6,0.8

Values obtained: These values are calculated by taking the average of all the ten seeds by using the script run_experiments.

Loss/throughput	ABT	GBN	SR
0.1	0.0201194	0.0200958	0.019906
0.2	0.0201072	0.0172784	0.0201027
0.4	0.0200582	0.0037956	0.0199058
0.6	0.0099556	0.0033942	0.019915
0.8	0.0026058	0.003572	0.0197387

Graph:



Observation:

When we set the window size to 50 we do not find much difference from the previous case except for gbn. We can therefore infer that as the window size increases the throughput for gbn is decreasing. And when the window size and loss both increase the throughput keeps dropping drastically. But sr performs better than both the protocols even with higher window size and with higher loss.

Experiment 2:

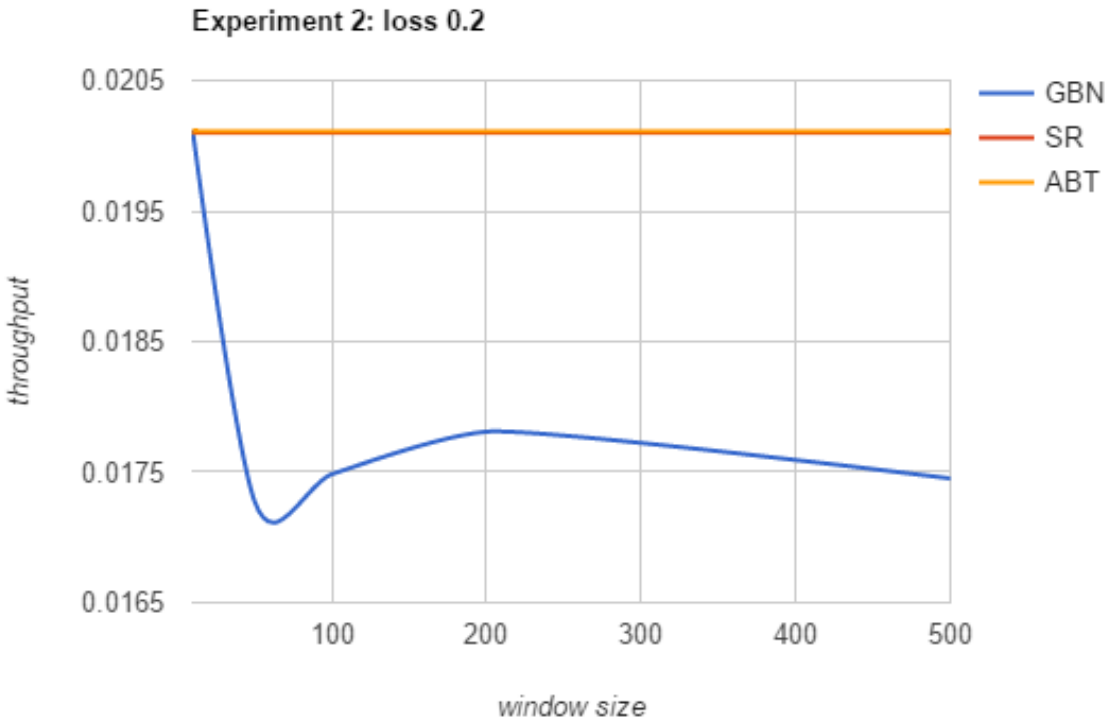
Settings for exp 1: number of messages:1000 time between two messages:50 corruption : 0.2

Case 1: loss:0.2 and window sizes of 10,50,100,200,500

Values obtained: These values are calculated by taking the average of all the ten seeds by using the script run_experiments.

windo/throughput	ABT	GBN	SR
10	0.0201072	0.0201197	0.020101
50	0.0201072	0.0172784	0.020101
100	0.0201072	0.017484	0.020101
200	0.0201072	0.0178095	0.020101
500	0.0201072	0.0174513	0.020101

Graph:



Observation:

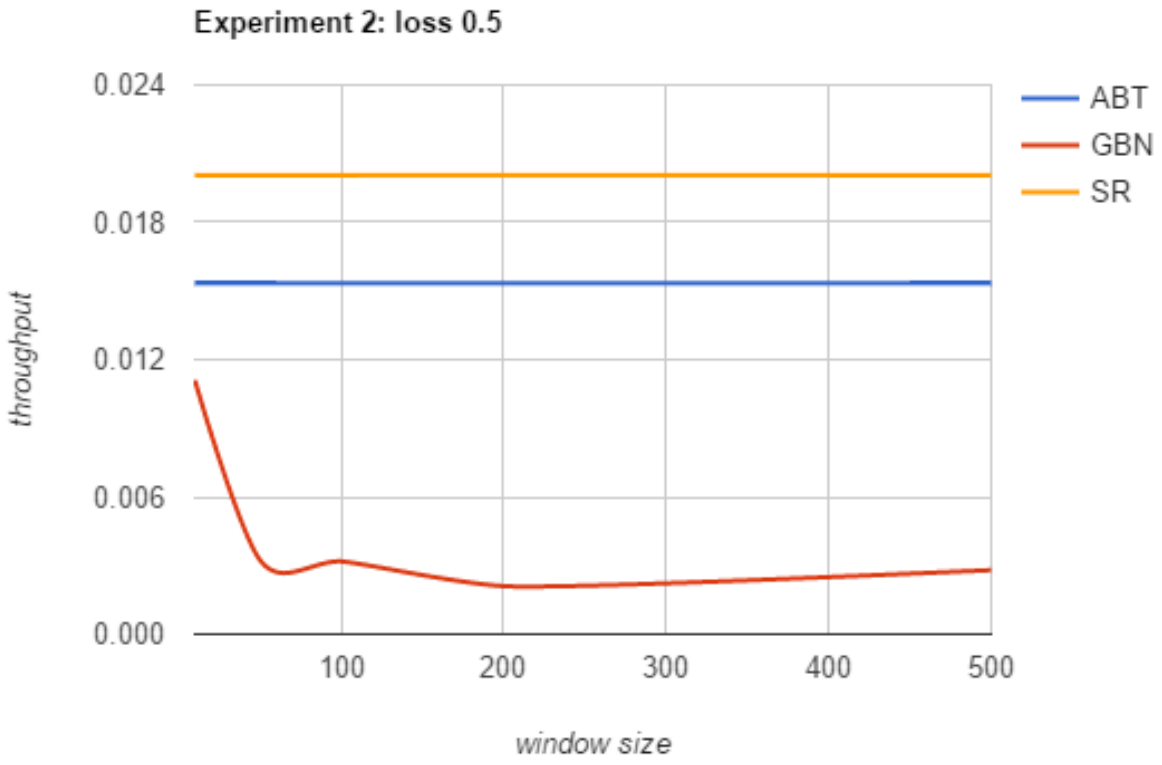
Here, this experiment is different from the previous one because in this case we are comparing throughput with window size rather than loss. When loss is 0.2 all the three protocols perform equally well at minimum window size, but as the window size keeps on increasing, the throughput of gbn is dropped by a large amout. Abt on the other hand does not depend on window size and performs equally for all the window values. Sr also performs equally good with abt at loss 0.2.

Case 2: loss:0.5 and window sizes of 10,50,100,200,500.

Values obtained: These values are calculated by taking the average of all the ten seeds by using the script run_experiments.

Window/throughput	ABT	GBN	SR
10	0.015337	0.0111066	0.0200574
50	0.015337	0.0032486	0.0200574
100	0.015337	0.0031842	0.0200574
200	0.015337	0.0021012	0.0200574
500	0.015337	0.0028021	0.0200574

Graph:



Observation:

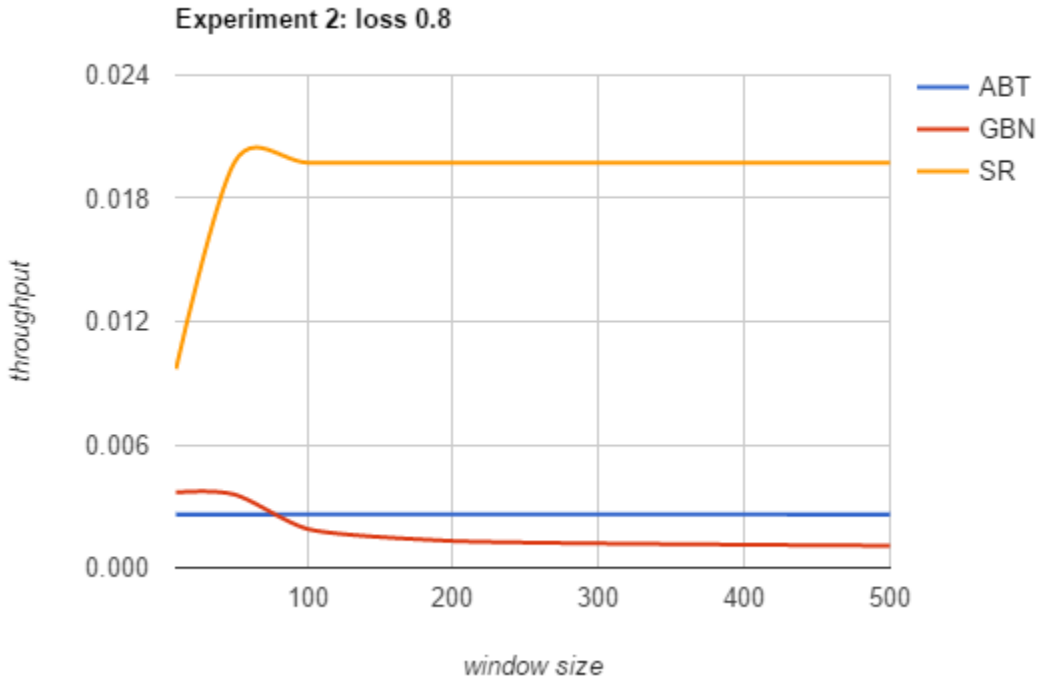
Here it is again proved that gbt has lower throughput at higher window sizes. This would have been because of congestion. Since everytime timer expires the whole window has to be retransmitted again leading to higher congestion and lower throughput. But as the loss increases abt throughput decreases. But sr performs better than all the two protocols even at higher loss and higher window size.

Case 3: loss:0.8 and window sizes of 10,50,100,200,500

Values obtained: These values are calculated by taking the average of all the ten seeds by using the script run_experiments.

Loss/throughput	ABT	GBN	SR
10	0.0026058	0.0036719	0.0096834
50	0.0026058	0.003572	0.0197387
100	0.0026058	0.0018942	0.0197387
200	0.0026058	0.0013053	0.0197387
500	0.0026058	0.0010765	0.0197387

Graph:



Observation:

In this case when the loss is set to 0.8 all the three protocols suffer a lower throughput at the beginning. Abt and gbn have lowest throughput at higher loss. But selective repeat starts with a better throughput. As the widow size increases sr performance becomes better. So finally we can conclude that sr performs better in all the cases.

Expectations vs Results:

ABT: For abt results are almost close to what I expected. Since abt sends one by one message and there will be no congestion in abt. So as the loss increases or corruption increases the same packet has to be sent again until the packet is received without any loss or corruption. And then the next packet is transmitted. So the same packet keeps transmitting number of times which leads to delay and less throughput by the time the simulator exits.

GBN: For gbn I expected the results to be little bit higher even for higher window sizes. But gbn turns out to perform lower at higher window sizes. But at higher loss and higher corruption I expected lower throughput and lower results just the way they turned out. So, by this I came to know that use of gbn is definitely faster and makes use of better bandwidth than abt but it leads to higher congestion. So when the window size is larger number of messages in the medium becomes more and leading to congestion.

SR: This protocol gave me surprising results. Even though I was expecting better performance for selective repeat than abt and gbn, I never expected there would be this much of a difference. I expected a small or average gain in the throughput. Even for losses up to 0.8 selective repeat almost transmits more than 95% of the packets received from the application layer of A. So at loss .9 since almost every packet is lost so by the time the simulator exits after the last message, selective repeat transmits more than 20% of the packets which is quite impressive.

Conclusion:

From the experiments conducted, we can conclude that selective repeat performs far better than all the other two protocols even when the window is higher and loss is higher. But abt performs good at lower loss and corruption values but as the loss and corruption increases abt has lower throughput. But gbn performs good at lower loss and corruption but as the loss increases gbn throughput decreases. Gbn is also dependent on window size as the window size increases gbn throughput decreases.

Selective repeat on the other hand performs best. Even at higher losses, higher corruption values and even at higher window sizes. So selective repeat is a better protocol that gives optimum results.