Analysis Assignment2

RELAIBLE TRANSPORT PROTOCOLS

I have read and understood the course academic integrity policy.

Name: Manjeet Singh

Person Number: 50169105

Default Timer Implementation (ABT and GBN):

I have used a static timer set to a default value "**default_timeout = 15.0".**
Why 15?

For ABT:

I tried timer values at 10 and 15. Among these timer values, when using a value 10, the number of packets received at the application layer of B is lesser that those received when the timer value is 15. This means the chosen timer scheme provides a much better throughput value for this transport protocol

e.g. ./abt –s 1 –w 50 –m 20 –l 0.0 –c 0.0 –t 50 –v 1

Consider this case where we are sending 20 messages with no loss or corruption over the communication channel.

At timer 10: 20 packets sent from Application layer to Sender A.

23 packets sent from Transport layer of A

22 packets received at Transport layer of B

16 packets received at application layer at B

Throughput: 0.015353

At timer 15: 20 packets sent from Application layer to Sender A.

23 packets sent from Transport layer of A

22 packets received at Transport layer of B

19 packets received at application layer at B

Throughput: 0.016399

Here we can see that using timer=15 offers a much better throughput and hence I persisted with 15.

Why not greater than 15?

I Didn't use values greater than 15 since, they tend to hamper the performance when working with higher loss rate and a large number of messages for GBN and for a honest comparison between all the protocols, I decided to keep the timeout scheme common for all the protocols.

For GBN: Similar to ABT, I tried different timer values such as 10, 15, 18 and 20.

I persisted with timeout period = 15, since it offers better throughput value and performance in comparison to other values.

Why 15?

e.g. ./gbn –s 1 –w 50 –m 200 –l 0.0 –c 0.0 –t 50 –v 1

Consider this case where we are sending 20 messages with no loss or corruption over the communication channel.

At timer 10: 200 packets sent from Application layer to Sender A.

339 packets sent from Transport layer of A

338 packets received at Transport layer of B

199 packets received at application layer at B

Throughput: 0.018852

At timer 15: 200 packets sent from Application layer to Sender A.

237 packets sent from Transport layer of A

236 packets received at Transport layer of B

199 packets received at application layer at B

Throughput: 0.019738

Here we can see that using timer=15, the number of packets sent between A and B is significantly lower than those when timer=15.

Also, the chosen timeout scheme, offers a much better throughput value.

Why not greater than 15?

I discarded timer values such as 18, 20, since at these timeout values, when I run the Experiment 1 with loss of 0.6 and 0.8 respectively, corruption at 0.2, with a 1000 messages over a window and average time between messages at 50, it takes longer than 6 minutes to finish the execution of the test cases and thus I persisted with timeout = 15. Thus the timeout scheme chosen, ensures that GBN performs fast if not optimal while transmitting over the communication channel.

Selective Repeat Timeout Scheme:

```
struct interrupt_pkt {
        float interrupt;
        pkt packet;
};
```

Variables used:

```
vector<interrupt_pkt> interrupts_vctor;
timeout_period = 15.0
```

How does the SR timer work?

```
A_output (struct msg message) {
```

When a new packet arrives, we check the sequence number of this packet at A,

get the current simulator time and add it to the the timeout_period

```
float clock_ticker = timeout_period+get_sim_time();
```

Add the clock ticker and the packet to the vector of struct interrupt_packet i.e interrupts_vctor.

Doing this, we ensure that we know the next time the current packet will be interrupted.

We check if the sequence number is equal to the window base

```
If(next_sequnum_A == base_A) {
        Starttimer(A, timeout_period)
}
```

When an interrupt occurs, we get the current simulator time. Now for all the unacknowledged packets which were added to the vector, and have an index position >= window base at A; we check for the interrupt values in the vector which are <= the current simulator time.

For all such packets, it means that at the current interrupt, these packets had timed out and need to be re-transmitted to B.

Now we need to update the interrupt values for all these packets to match the next expected interrupt value. Hence we add the timeout_period to all the packets that are being re-transmitted to B from A_timerinterrupt()

Code logic:

```
A_timerinterrupt() {

float interrupt_time = get_sim_time();

for(int i from base_A till last element of interrupts_vctor)

        check if the interrupt value <= interrupt_time and the packet is unacknowledged

        if yes

update packet interrupt as retransmit_packet.interrupt= retransmit_packet.interrupt+timeout_period

}
```

Logic for starting the timer again:

For all the unacknowledged packets present in the interrupts_vctor, we choose the packet with the smallest value of the expected interrupt time. We subtract this value from the current interrupted packet's interrupt value and start the timer again.

```
float temp_interrupt = next lowest interrupt value – interrupt_time

starttimer(A, temp_interrupt);
```

If there are no packets with interrupt values greater than the current interrupt value, we start the timer without default timer_interrupt i.e. starttimer(A, temp_interrupt).

Consider the following example to show how the described timer logic works:

Default timer = 15, get-sim_time() = 1 ticks for every A_output

Assume the interrupts_vector has the following values: {15,16,17,18,19,20}

Assume the base_A i.e window base at A is at 16.

Now say, the inuerrupt occurred at time 16. Out vector will now be updated as follows:

Interrupts_vctor: {30, 31, 17, 18, 19, 20}

So the next expected interrupt is at 31 and we start the clock at 17-16 = 1 i.e starttimer(A, 1); which ensures the next interrupt will be for the unacknowledged packet with interrupt set at17.

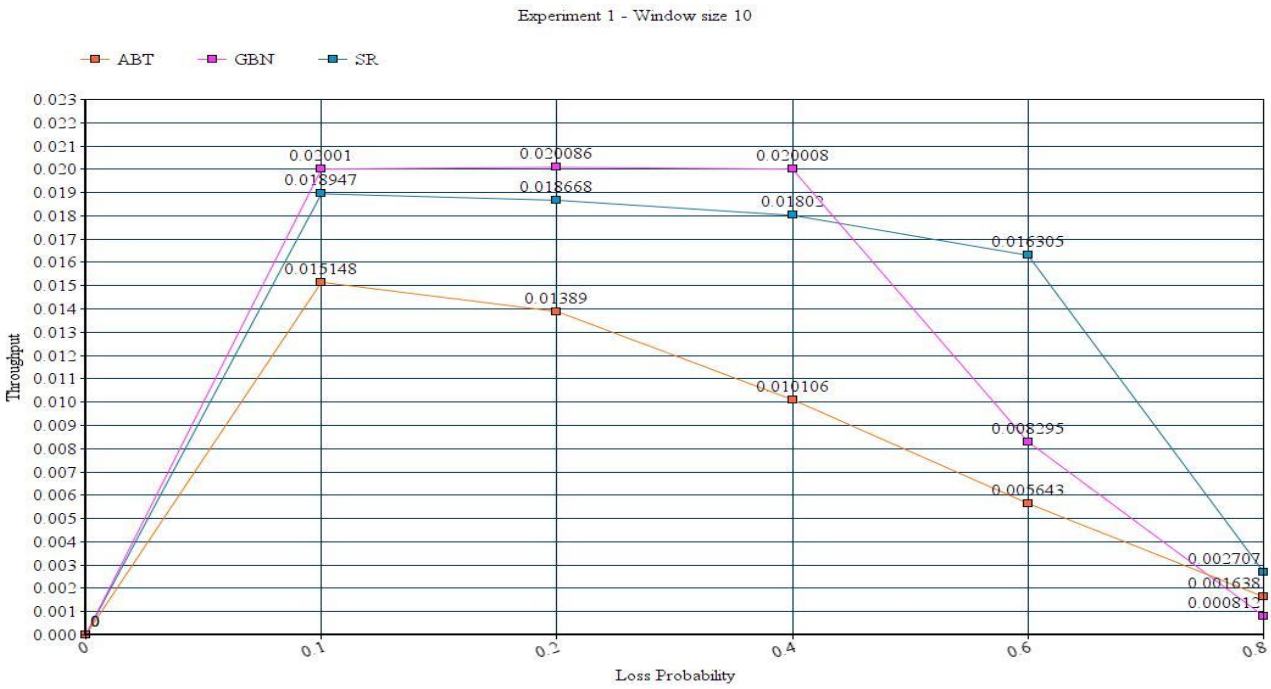
Checksum Calculation:

1. compute_checksum(int seqnum, int acknum, char msg[20]) : This method is responsible for finding the checksum value to be sent from A to B in the packet header. The checksum computation is done as follows:
    - If the message in the packet received at A is NULL, return 0 as a checksum value.
    - Add all the msg bits, in this we simply add the characters from 0 to 19 i.e the first 20 characters in the packet message, the sequence number and the acknowledgement number. The returning checksum is the negation of this sum.
2. Verify_checksum(pkt *packet) : This method is called at receiver B to verify the checksum of the received packet to decide if the packet was corrupted or not.
   Computation logic:

   Similar to compute_checksum, add the first 20 message bits, sequence number, acknowledgement number and the received checksum itself. If the result of this is ~0, the packet was received without any corruption else we say that the packet data was corrupted at layer3.

Experiment 1: Analysis

Window size – 10



Observations:

Here we can see, ABT performs poorly compared to GBN and SR protocols.

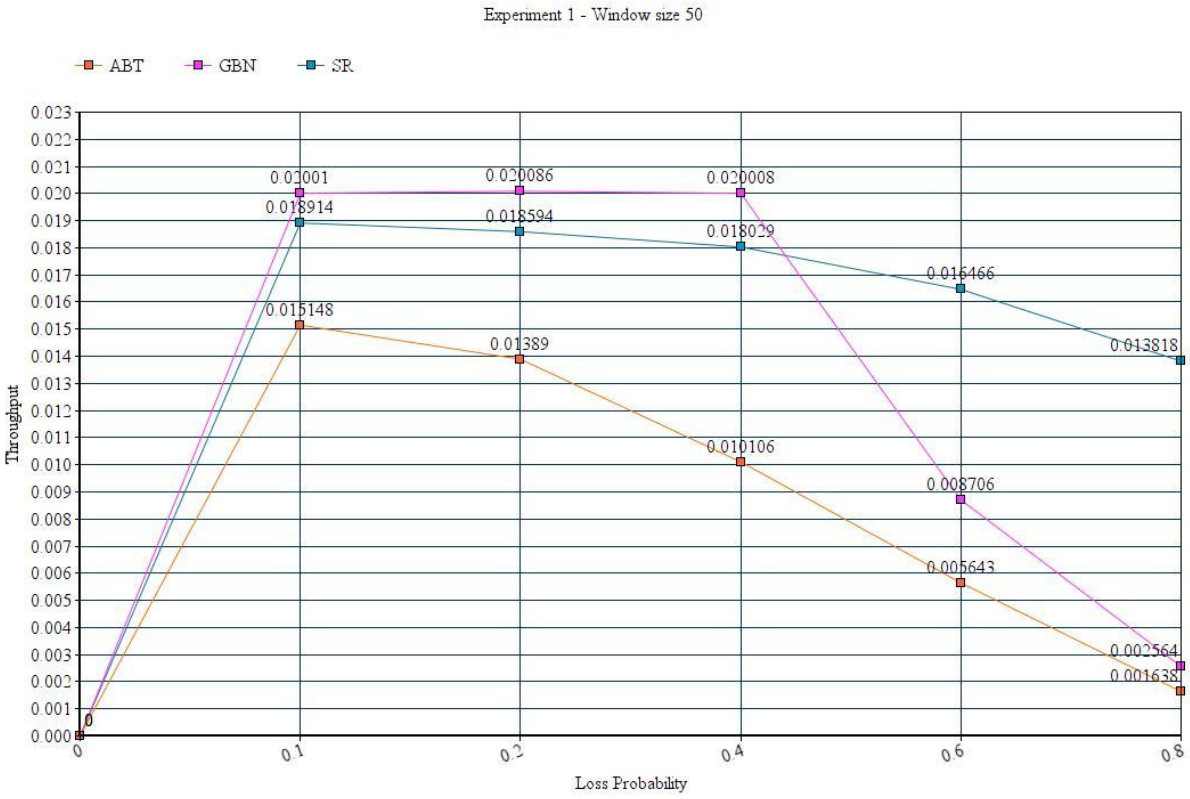
This is primarily because, in ABT until an acknowledgement is received, the next packet won't be transmitted which leads in a lot of packets being dropped. Hence it performs, poorly compared to GBN and SR, as we can see in the graph.

For GBN and SR, we can see that the performance measure of loss probability vs throughput is consistent when the loss probability is 0.1, 0.2 and 0.4 and then it drops by a huge factor in comparison to SR. Thus from these observations, I can say that when the loss was small, GBN performance was slightly better than SR but as the loss probability increased to 0.6 and 0.8; GBN performed poorly. This is because with a higher loss rate, in GBN we have to re-transmit a large number of packet whereas in SR, we just transmit the packet which was lost leading to a better performance.

Expected variations for ABT:

It was expected that as the loss ratio increases, ABT will continue to decline in terms of performance because the number of packets transmitted to the receiver would continue to decrease. This is a major drawback of ABT (RDT3.0) protocol, since it doesn't buffer any packets at the sender but simply drops them until the acknowledgement for the last packet was received.

Window Size - 50



Analysis: GBN vs SR:

It was expected that the performance of GBN for a small window size and low loss probability such as 0.1, 0.2 and 0.4 would remain similar. This is because since the window is small and the loss rate is low, the number of re-transmissions required would be few and hence the performance would continue to be more or less the same. But will increasing loss ratio, the number of packets for re-transmissions would be higher and hence the performance decreases which is the resultant output of the experiment as well, as we can see in the graph above.

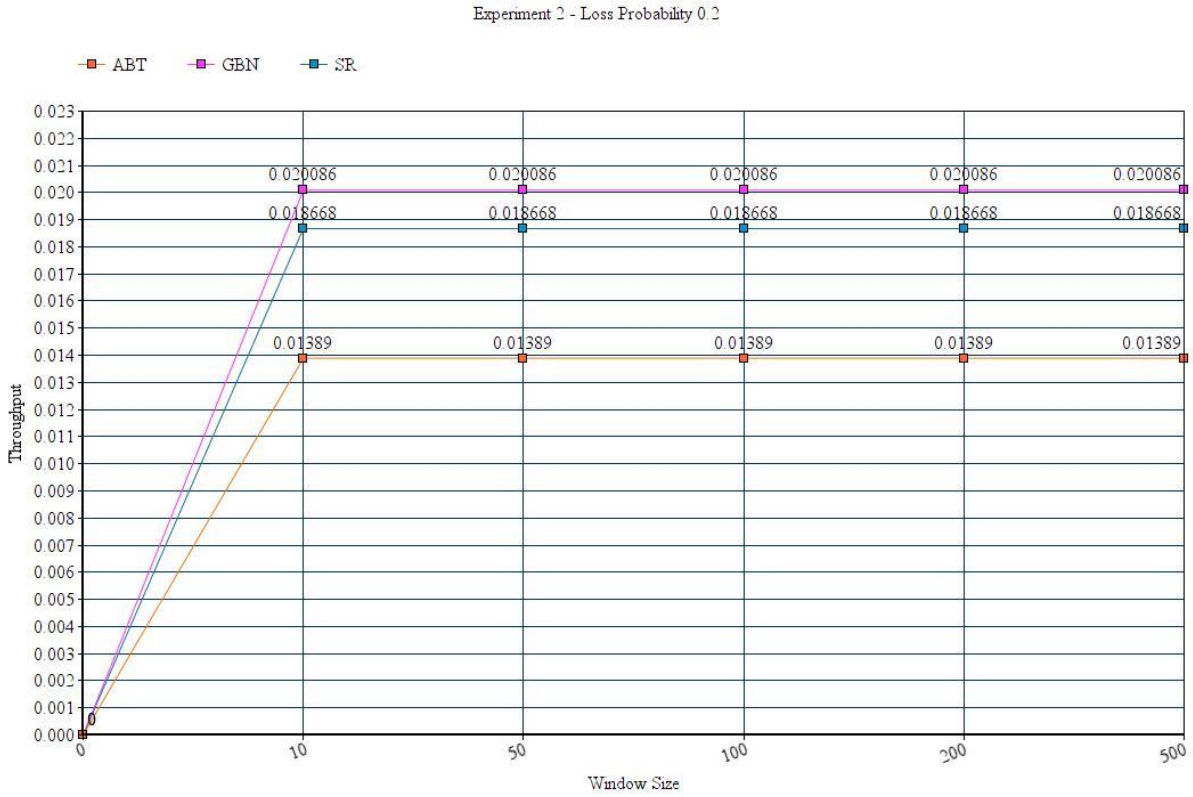
Selective repeat was expected to perform more or less similar to GBN for lower loss rates since GBN would not need a large number of transmissions. But as we increase the loss probability, the performance of SR would be superior to GBN and ABT since SR would not re-transmit all the unacknowledged packets but only the ones it needs to.

From the graph we can we can see that, GBN performed slightly better than SR for lower loss probabilities, which was not expected but the difference in performance isn't much so we can say they both performed more or less similar. But as the loss would increase, SR would outperform GBN which is clearly visible in the graph above, since GBN shows a huge fall in its performance for loss of 0.6 and 0.8 whereas SR still continued to perform as expected.

I don’t completely agree with my measurements for SR for loss of 0.1, 0.2 and 0.4 since it ideally should perform better than or as well as GBN for a small window and fewer losses. But for higher loss probability, SR performs as expected and so do GBN and ABT.

Thus, I can say that GBN and ABT perform as expected for varying probabilities of the packet loss but SR performs a little below expectation for lower loss rates. This could be because, my interrupt timer implementation, isn’t optimized well enough and is dampening the performance by a very small percentage.

Experiment 2: Loss 0.2



Observation:

Throughput remains constant irrespective of the window size.

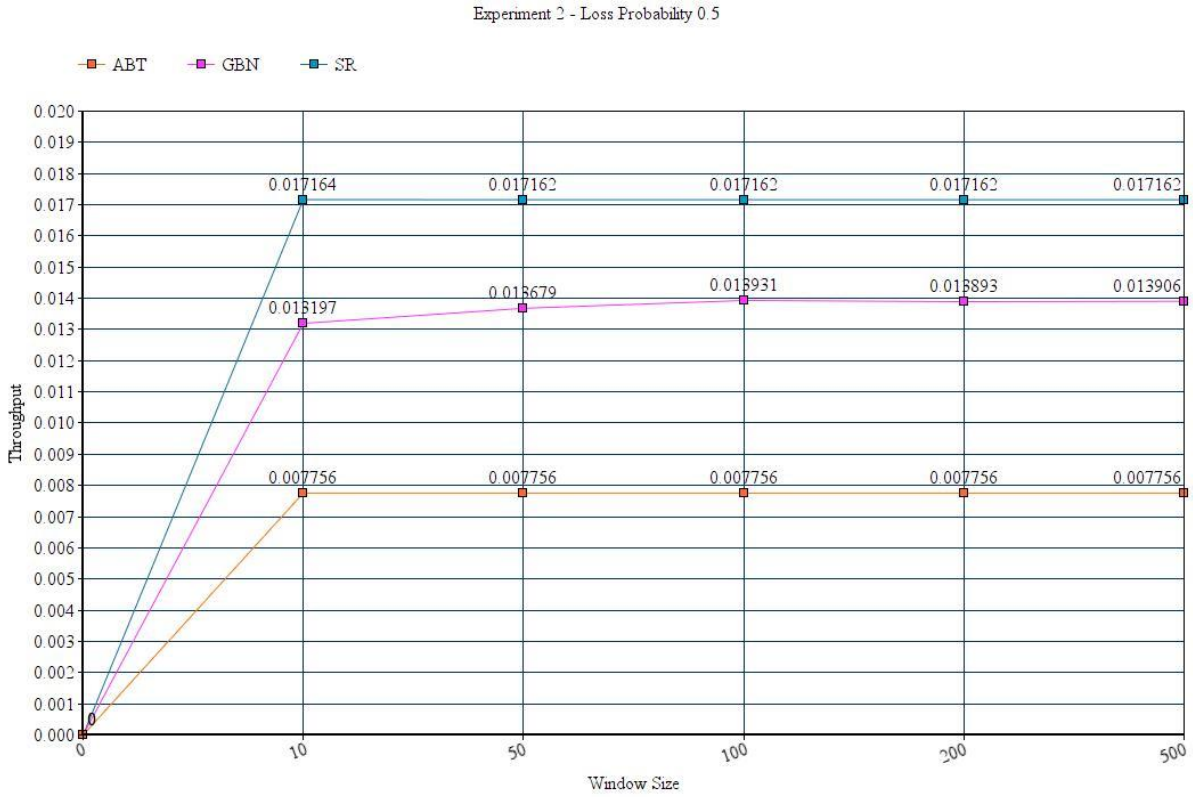
GBN performs slightly better than SR for a channel with small loss probability.

Analysis:

Here we can see that at window size =10 the maximum throughput is achieved and thereafter increasing the window size would have no effect on the throughput.

Thus we can say that, when the network is good, as in this case; we achieve the maximum throughput for a particular window size N and thereafter increasing the window wouldn't affect the throughput.

Experiment 2: Loss 0.5



Observations:

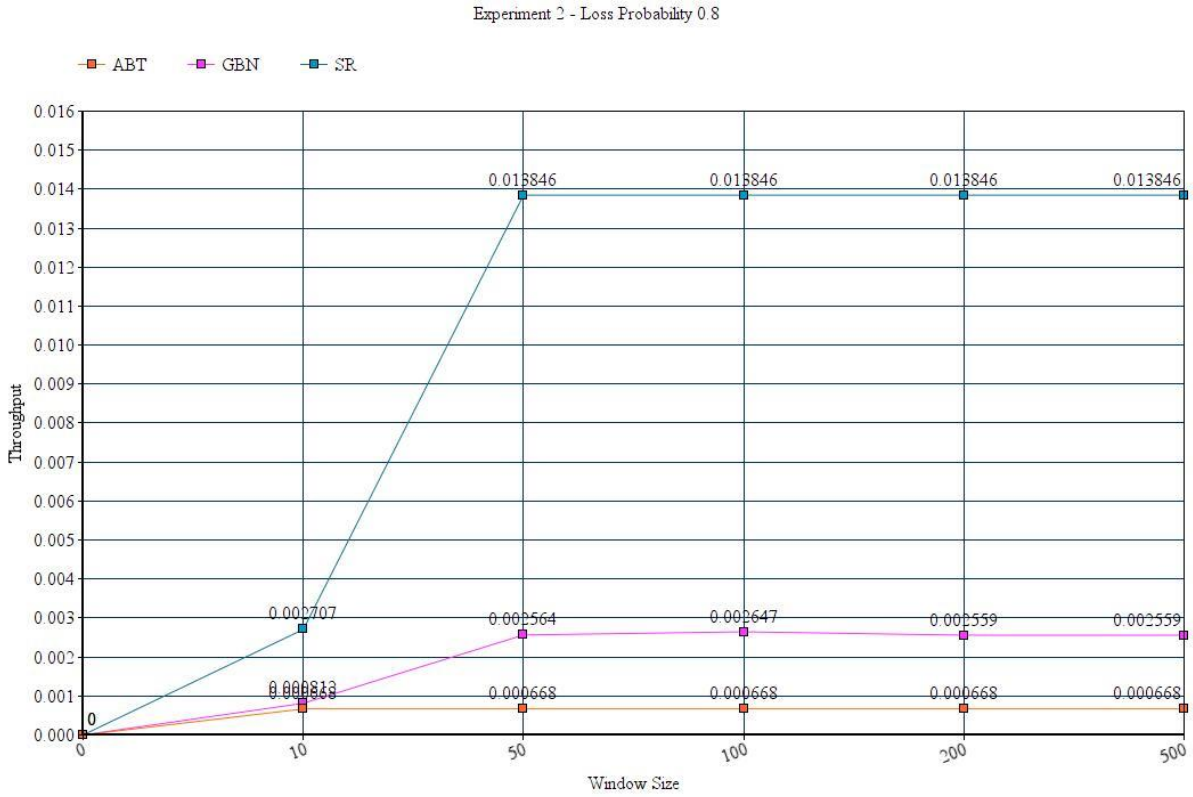
When the network became bad, i.e. the loss probability increased, we need to increase the window size N to get the maximum throughput over the network.

For higher loss rate, SR is the clear winner. GBN would need to transmit a large number of messages every time a packet is lost there by causing poor performance compared to SR which performs best since it would re-transmit only the lost packets.

Analysis

In our implementation, we accept packet only if the window is not full. Thus if the window is small, more packets would be dropped which is why we see lower throughput values for a smaller window size. When the window reaches the maximum needed to achieve the optimal/maximum throughput, it stagnates and thereafter, increasing the window size will have no effect on the throughput.

Experiment 2: Loss 0.8



Observations:

When the loss rate is really high which implies the transmission network is really bad, GBN will perform very poorly, SR on the other hand would give the best performance for a network with high loss probability.

CONCLUSION:

Go back N protocol and Selective Repeat have similar throughputs when the network is good. In such cases, increasing the window size won't affect the throughput value.

When the network is really bad i.e. in cases such as loss probability is 0.5 and above, selective repeat protocol offers the best performance.