Programming Assignment 2

Name: Manpreet Dhanjal

UBIT Name: DHANJAL

Person Number: 50248990

I have read and understood the course academic integrity policy

Introduction

Reliable data transfer is a requirement of most of the present world's networking applications. For this, many protocols have been designed each having their own pros and cons. Basically, a reliable data transfer protocol has to deal with data loss, data corruption and should deliver the data in order. Also, while designing a protocol one should also keep in mind that the layer below the reliable data transfer protocol might be unreliable. In this programming assignment, I have implemented Alternative Bit (ABT) Protocol, Go Back N (GBN) protocol and Selective Repeat (SR) protocol and tried to analyze their performances.

Time Out Schemes

While implementing a transport layer protocol, we have to decide on certain parameters like the window size, range of sequence number, and time out schemes etc. Deciding the time out value is very crucial, because if we take a very large value for timeout, it will result in large delay in retransmission and if we take a very small value for timeout, it will result in too many retransmissions, thus decreasing the over all throughput of the transport layer.

- ### Alternating-Bit-Protocol

The Alternating-Bit protocol transmits just one packet at a time and waits a specific amount of time for the acknowledgement. If the acknowledgment is received within the described time frame, the new packet is sent, otherwise, the old packet is retransmitted and the timer is started again.
Now, at any point of time, there will be only 1 packet in the network and according to the assignment document, a packet sent into the network takes an average of 5 time units to arrive at the other side when there are no other messages in the medium. Hence, the average RTT will be approximately, 10 time units. It is better to wait for a delayed acknowledgement rather than retransmit the same packet again. So, I decided to make the timeout value to be static and greater than 10 time units.
Running experiments in a fixed environment and checking the throughput values at different values of timeout, the optimal value for timeout comes as 13 time units.

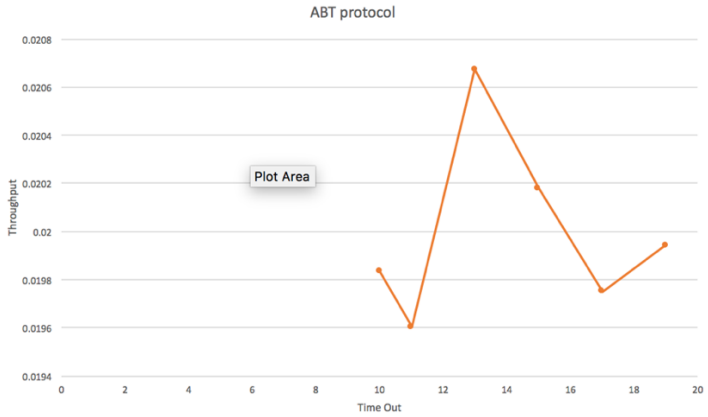

Figure: Throughput vs Timeout
m = 1000, l = 0.2, c = 0.2

- ## Go-Back-N

Go-Back-N protocol implements pipelining of packets and can transmit multiple packets before waiting for acknowledgment. It implements a single timer and retransmits all the unacknowledged packets if the timer times out.

As the size of sliding window for GBN protocol increases, the number of retransmissions in the case of premature timeout will be very large. For example, if 100 packets were transmitted and because of small time out value, the timer expires before the acknowledgement for any of the packet is received, all of the 100 packets will be transmitted again. This reduces the throughput of the protocol. Hence, a timeout value proportional to the number of packets transmitted should be selected.

The average RTT for a packet sent using this protocol would be more that 10 time units since there is high probability that more than 1 packet will be present in the network at the same time. Adaptive time out calculations using SampleRTT would be a little difficult to formulate for GBN protocol because there is a single timer for the unacknowledged packets and we can receive cumulative acknowledgements.

Running experiments in a fixed environment by varying the time out values and checking the throughput, the optimal value of timeout comes as 16 time units.

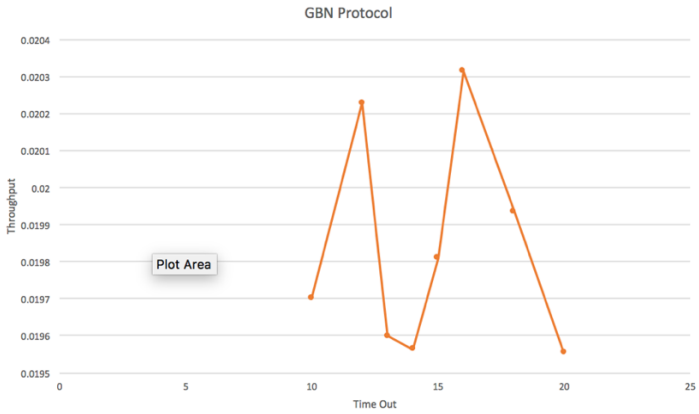

Figure: Throughput vs Timeout
m = 1000, l = 0.2, c = 0.2

So, in the case of normal transmissions, I use 16 time units (GlobalTimeOut) as the timeout values, but in the case of timer interrupts, since multiple transmissions will be there I calculate the timeout value as follows:

Timeout = constant_value * number of packets to be retransmitted
Where Number of retransmissions = Base value – next sequence number

If the calculated timeout value comes out to be less than globaltimeout value, I update the timeout value to the globaltimeout value.

- ## Selective-Repeat

Similar to Go-Back-N protocol, Selective Repeat protocol also implements pipelining of packets and transmits multiple packets. However, it makes use of multiple timers, i.e. 1 timer per unacknowledged packet. If any of the timer times out, only the corresponding packet is retransmitted. Since, separate timers are maintained for each packet, we can improve the performance of the protocol by calculating the timeout adaptively.

By running experiments in a fixed environment and using a static timeout scheme, 20 time units is obtained as the optimal timeout value.

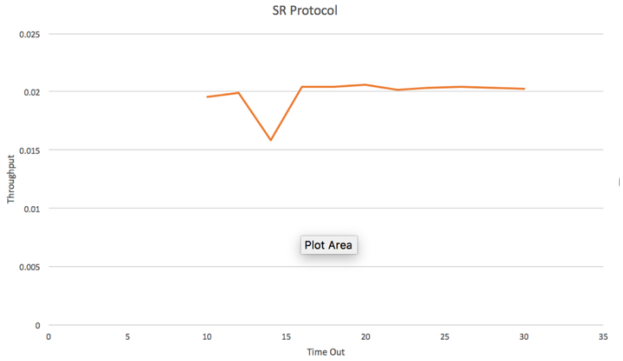

Figure: Throughput vs Timeout
m = 1000, l = 0.2, c = 0.2

To calculate the value of timeout on the run, I take the default timeout value equal to 20 time units, since this value was obtained by running experiments in fixed environment.

There are 2 main scenarios where timeout needs to be calculated:

1. Acknowledgement received: When acknowledgement for a particular packet is received we can calculate the corresponding SampleRTT by taking the time difference between the time the packet was sent and the time the acknowledgement is received.

   currentEstimation = (1 – alpha) * EstimatedTimeOut + alpha * sampleRTT
   EstimatedTimeOut = currentEstimation + margin

   Here, I have taken alpha = 0.8 and I have hardcoded the margin to 5.

2. Timeout Event: When timeout event happens, generally, the old time out value is doubled, but for this project assignment, I have kept the values of timeout to be equal to the old value only.

Following is a comparison between static timeout and adaptive timeout schemes for varying loss and corruption probabilities.

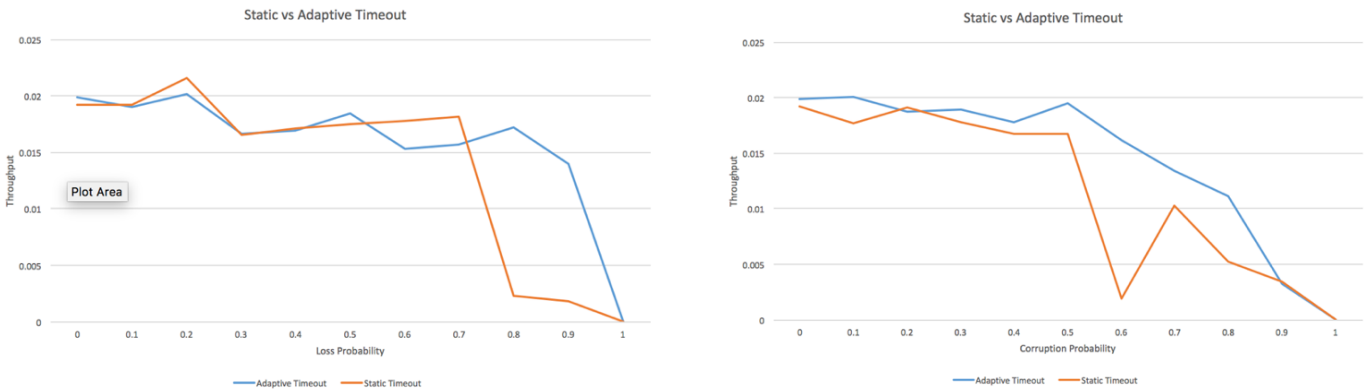

Figure:Static vs Adaptive Timeout
Left- wrt Loss Probability Right- wrt Corruption Probability

By using adaptive time out scheme, the throughput increases. Hence, I have used the adaptive timeout scheme.

Implementation of multiple timers

To implement Selective-Repeat protocol, one timer is needed for each unacknowledged packet. There will always be a constraint on the number of hardware timers we can get, therefore, we can try to get multiple software timers with limited number of hardware timers, which is 1 in the case of this programming assignment.

• Data Structure

In order to obtain multiple timers, I have used circular buffer or timing wheel [2] with minor modifications.
To maintain the status of timer for each unacknowledged packet I created a structure (*timeNode*) containing the time stamp when the packet was sent to layer 3 and sequence number of packet. The structure is as follows:

```
Struct timeNode{
        Float timestamp;
        Int seqnum;
 }
```

The aforementioned Circular Buffer is actually a circular array (or linked list) of *timeNode* of the packets that have been sent to layer 3. The name for the circular buffer is *timerArr*. A timer in this circular buffer is invalid if the packet has already been acknowledged, for this I have created another array, *seqArrA*, which keeps a track of all the unacknowledged packets. The indexes of *seqArrA* represents the sequence number and the value 0 tells that the packet is unacknowledged whereas the value 1 tells that the packet has been acknowledged. The variable *timerStart* always points to the *timeNode* whose timer is currently running and the variable *timerEnd* points to the first available space in the circular buffer. The *timeNode* for any new packet is added at the *timerEnd* index. Since, the *timeNode* is created for each packet as and when it arrives and is inserted into the *timerArr*, the *timerArr* will always be sorted in increasing order of *timestamp* and I make use of this fact to calculate the new time out value every time an acknowledgment is received or timeout occurs. The size of both *seqArrA* and *timerArr* should be equal to the range of sequence number used. For my implementation I have used adaptive timeout scheme to calculate the timeout for a single packet, and it is stored in variable called *globalTimeOut*.

• Implementation

The *timeout* value is calculated by subtracting the difference between the current simulation time and the *timestamp* of the *timeNode* under consideration from the adaptively calculated timeout for each packet (*globalTimeOut*). Following is the formula:

$$timeOut = globalTimeOut - (get\_sim\_time() - timeNode.timeStamp)$$

Consider, we have 2 unacknowledged packets, so the *timerArr* should look as the diagram below:

SeqNum: 1 TimeStamp: 19:20	SeqNum: 2 TimeStamp: 19:25	

If acknowledgement for SeqNum 1 is received at 19:30. Then the current timer is stopped and new timeout is calculated as:

$$Timeout = 25 - (19:30 - 19:25) = 25 - 5 = 20$$

Thus the timer is again started with timeout value of 20 time units.

The implementation part of the timer can be divided into 3 main event scenarios:

Event 1: Input from layer 5

- Whenever a new message needs to be sent, a packet is created and is saved in a buffer, *sentBuf,* for retransmission.
- A *timeNode* is created for it with the *timestamp* as the current simulation time and *seqnum* as the sequence number of the new packet. This *timeNode* is then inserted into the circular buffer, *timerArr*.
- If the timer is not already running, that is, current packet is the only unacknowledged packet, calculate the time out for the current packet and start the timer.

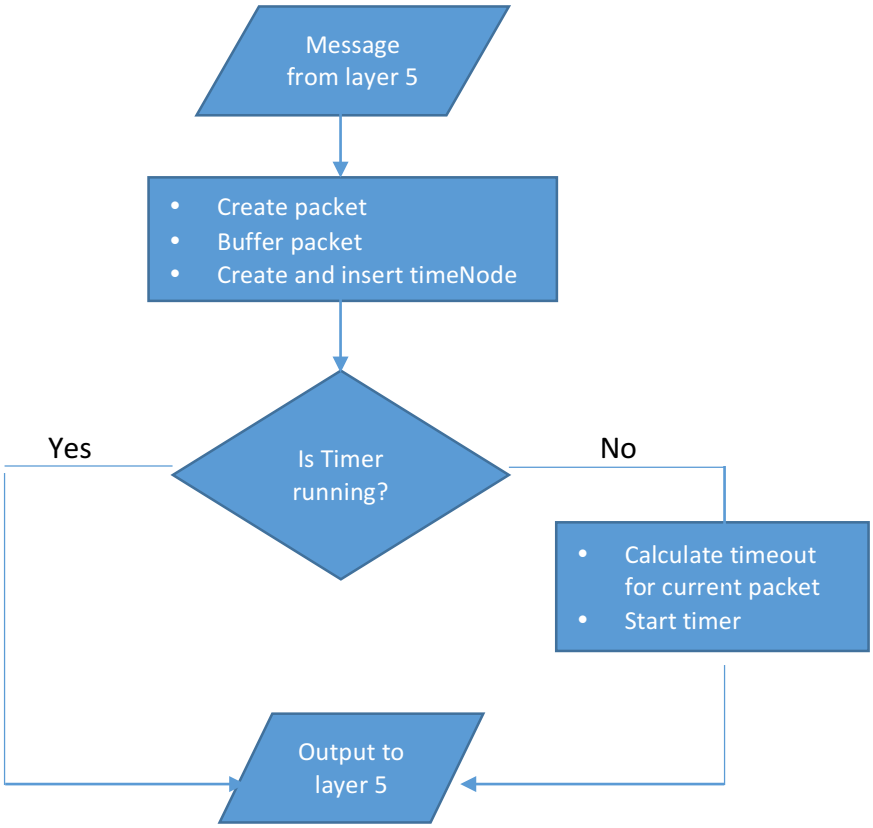

Figure: Timer flowchart when input from layer 5 received

Event 2: Non duplicate and uncorrupted acknowledgement

- Stop the current timer.
- If the acknowledgement is corrupted just wait for the timeout event.
- Otherwise, get the *timeNode* for the oldest unacknowledged packet and calculate the new timeout value for which the timer should run.
- Start the timer.

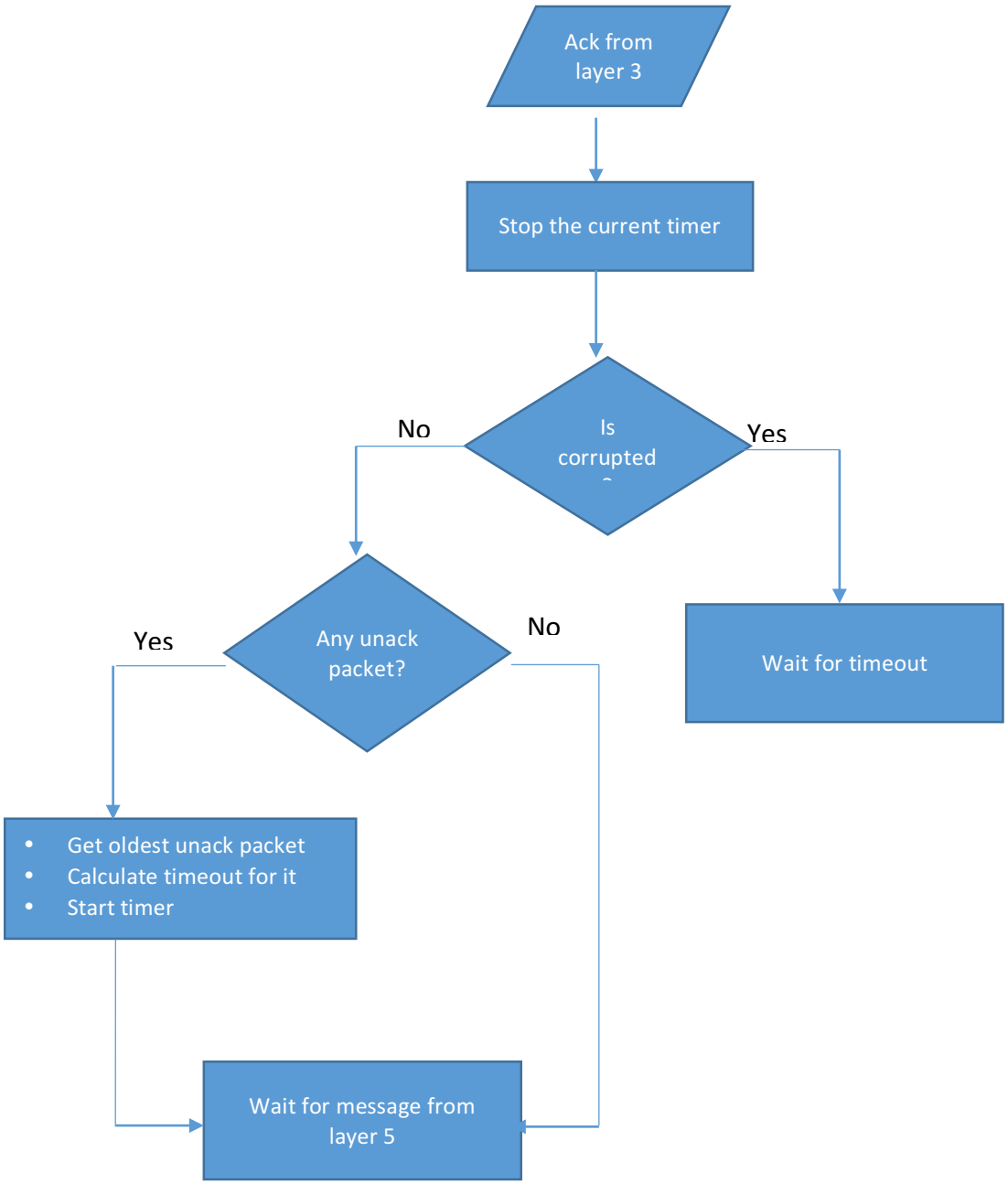

Figure: Timer flowchart when non corrupt and non duplicate acknowledgment received

Event 3: Time out event

- Get the *timeNode* of the packet whose timer just timed out, update the timestamp of the *timeNode* to the current simulation time, since this is the time at which we are retransmitting the packet.
- Insert this node into the *timerArr* at the end of the circular buffer (*timerEnd* index). This packet will now be the latest unacknowledged packet.
- Get the *timeNode* for the oldest unacknowledged packet and calculate the new timeout value for which the timer should run.

- Start the timer.

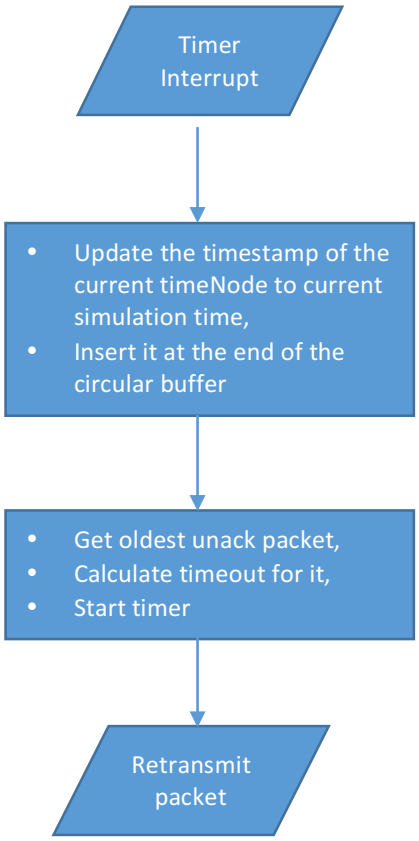

Figure: Timer flowchart for timeout event

Performance Comparison

- ### Experiment 1

Window size = 10

Following are the graphs of (Maximum, Minimum, Average) Throughput vs Loss Probability for window size 10.

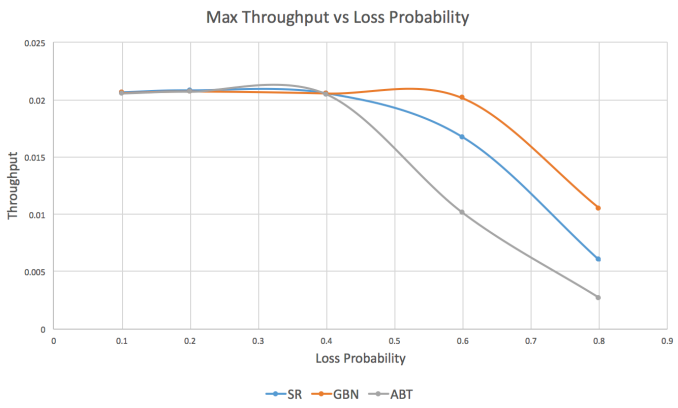

Figure: Max Throughput vs Loss Probability

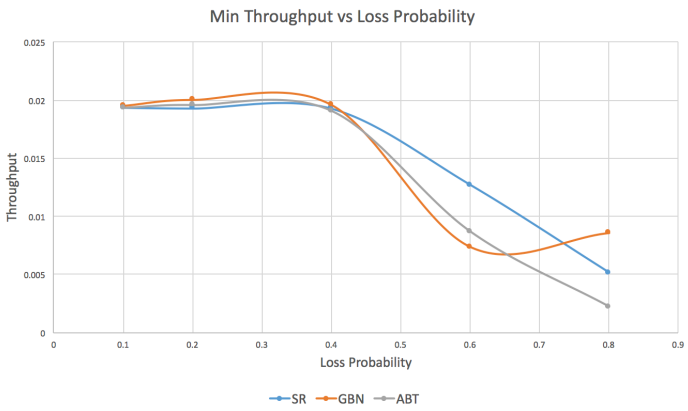

Figure: Min Throughput vs Loss Probability

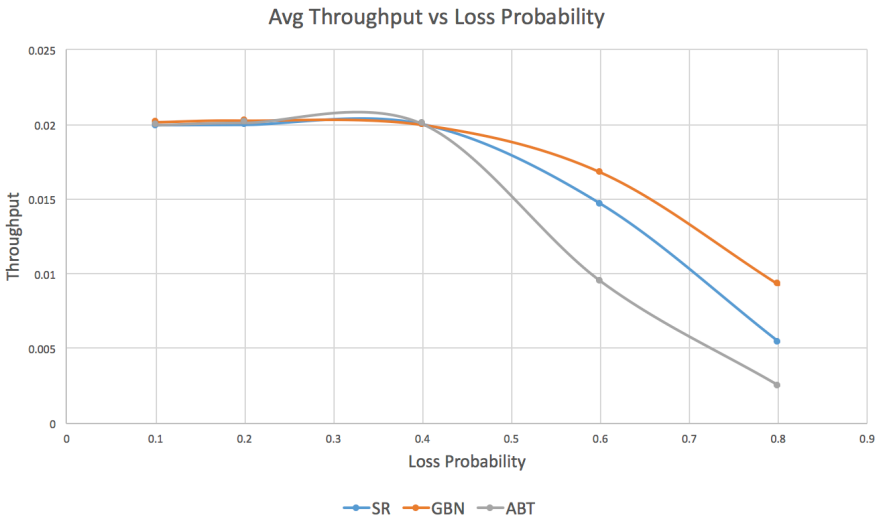

Figure: Avg Throughput vs Loss Probabiltity

Observations:

- The average throughput for all the three protocols decreases with increase in loss probability (approximately). However, the throughput doesn't vary linearly.
- All the three protocols have similar performance till 0.4 loss and after that GBN performs better as compared to other two protocols i.e. the performance of GBN is the better than SR and ABT protocol for higher losses.

Analysis:

With increase in loss probability, the throughput for all of the protocols is expected to decrease and this is what is observed from the graphs. Selective Repeat protocol is expected to perform better than GBN and ABT protocol because of the less number of retransmissions required as compared to GBN and less waiting time compared to ABT. But this is not observed from the graphs.

The reason for GBN to perform better than SR protocol might be that since at high loss probability the number of packets lost will be very high, it will be better to retransmit all the packets within the window at once, as long as the window is size is not very large, rather than waiting for timeouts for individual packets. In this way, the receiver will be able to receive the retransmitted data sooner.

Window Size = 50

Following are the graphs of (Maximum, Minimum, Average) Throughput vs Loss Probability for window size 50.

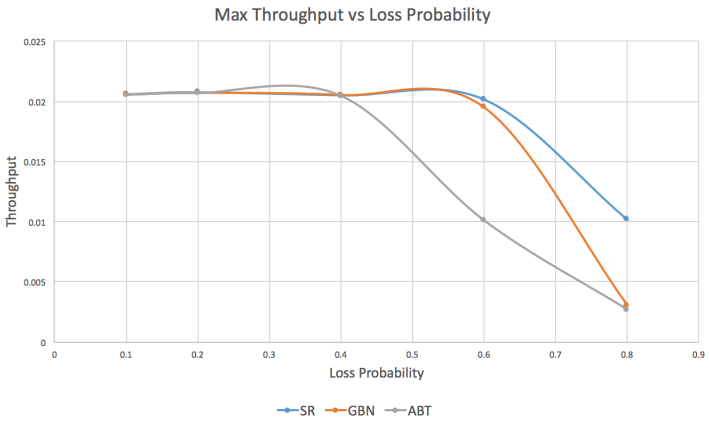

Figure: Max Throughput vs Loss Probability

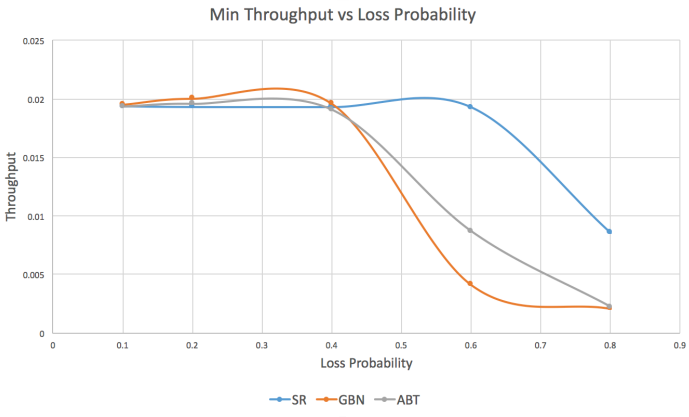

Figure: Min Throughput vs Loss Probability

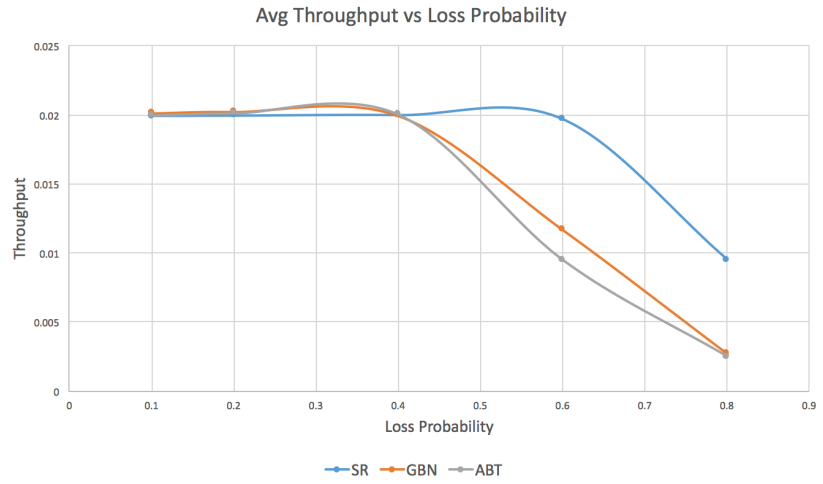

Figure: Avg Throughput vs Loss Probability

Observations:

- The throughput is inversely proportional to the loss probability i.e. it decreases with increase in loss probability.
- Selective Repeat performs better than both ABT and GBN protocol.
- The average throughputs of ABT and GBN are comparable.

Analysis:

The throughput is expected to decrease with increase in loss probability. This is because, with increase in loss, more retransmissions will occur and thus more time would be required to transmit whole data correctly, thus reducing the throughput. This is also observed from the graphs displayed above.

ABT protocol waits for acknowledgement for each packet before sending the next packet, thus the time required to transmit data in the case of losses will be large. In GBN protocol, the receiver discards all out of order packets thus requiring retransmission of a lot of data even if some part of it was delivered correctly. Hence, this results in less throughput for GBN. Therefore, SR protocol will perform better than both GBN and ABT protocol, because of the pipelining and less number of retransmissions. And this is observed from the graphs displayed as well.

• Experiment 2

Loss Probability: 0.2

Following are the graphs of (Maximum, Minimum, Average) Throughput vs Window Size for 0.2 Loss Probability.

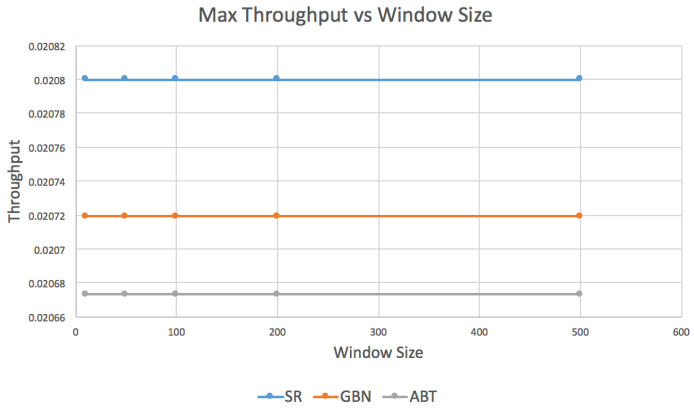

Figure: Max Throughput vs Window Size

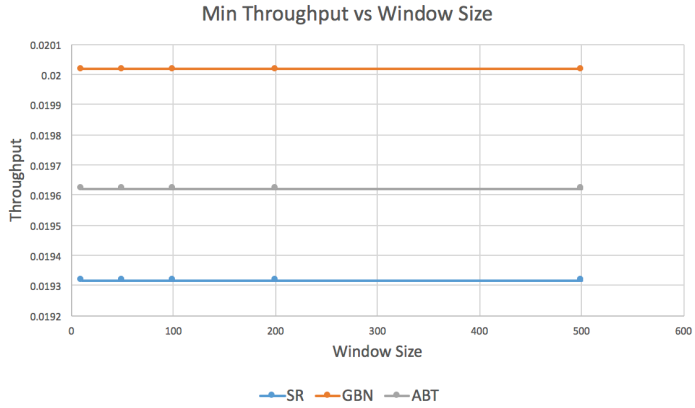

Figure: Min Throughput vs Window Size

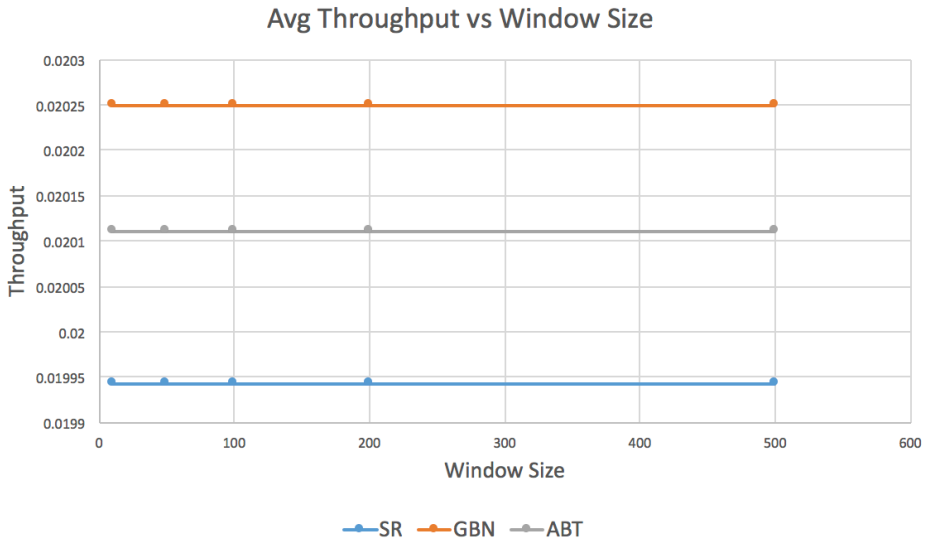

Figure: Avg Throughput vs Window Size

Observations:

- The throughput remains constant with the increase in window size at low loss.
- The average throughputs of all three protocols have comparable values.
- Maximum throughput is reached using SR protocol.

Analysis:

The size of window defines the number of packets that can be sent into a network simultaneously. For loss probability of 0.2, the packet loss will be less. Hence, the performance should increase or at most remain constant with increase in window size. The same is observed from the graphs.

Loss Probability: 0.5

Following are the graphs of (Maximum, Minimum, Average) Throughput vs Window Size for 0.5 Loss Probability.

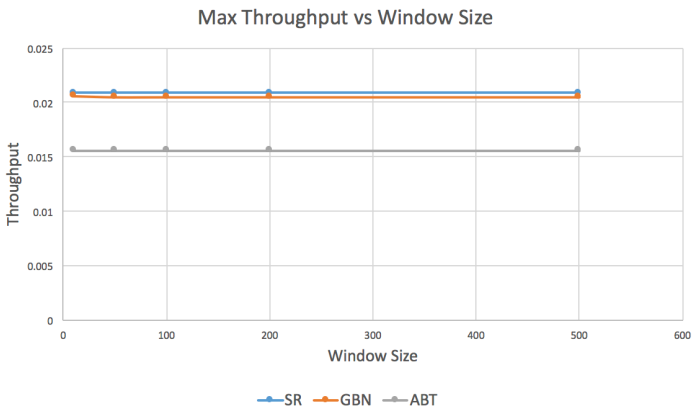

Figure: Max Throughput vs Window Size

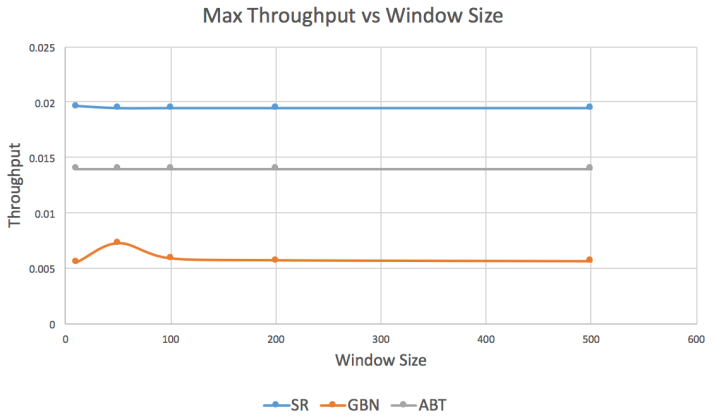

Figure: Min Throughput vs Window Size

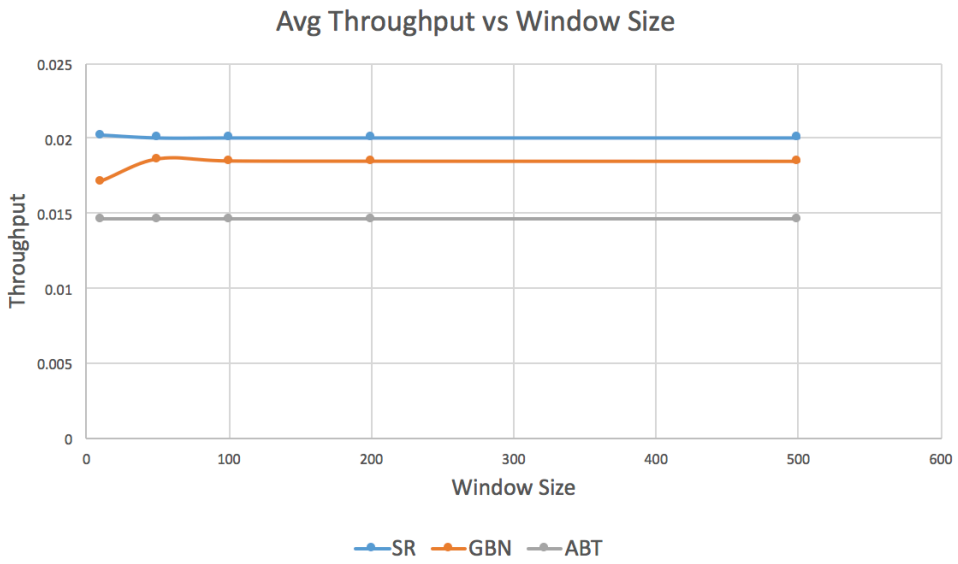

Figure: Avg Throughput vs Window Size

Observations:

- The average throughput of all three protocols remains almost constant.
- There is minor increase in throughput of GBN when we go from window size 10 to window size 50.
- Average throughput of ABT is the least.

Analysis:

At high losses, SR and GBN are supposed to perform better than ABT, this is because of the pipelining of packets. The same can be observed from the graphs.

Loss Probability: 0.8

Following are the graphs of (Maximum, Minimum, Average) Throughput vs Window Size for 0.8 Loss Probability.

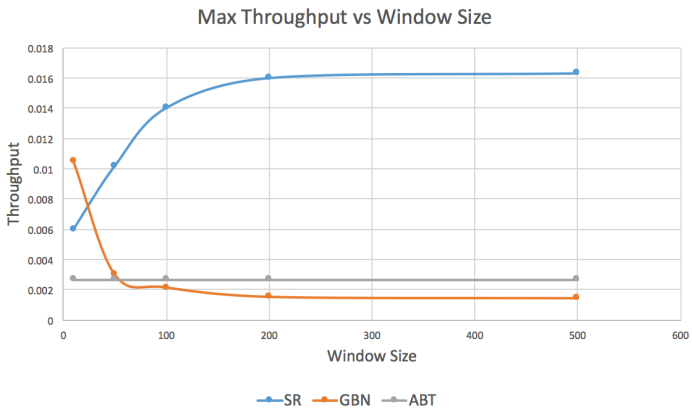

Figure: Max Throughput vs Window Size

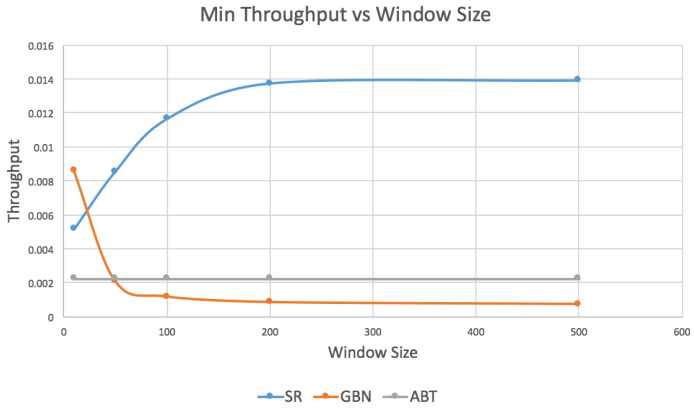

Figure: Min Throughput vs Window Size

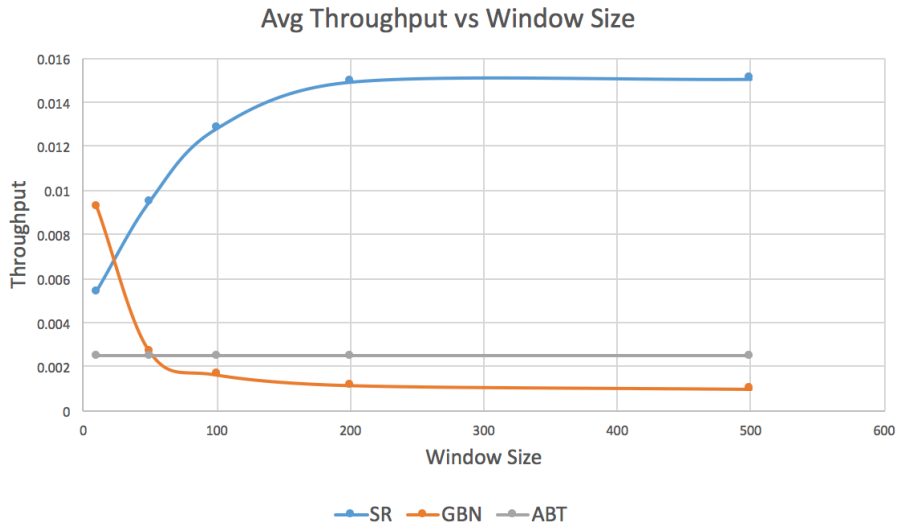

Figure: Avg Throughput vs Window Size

Observations:

- The average throughput of SR protocol increases with increase in the window size whereas, the average throughput of GBN decreases with increase in window size.
- Overall, the performance of SR is the best among all three protocols.
- ABT performs the worst in the case of high loss environment.

Analysis:

The SR protocol performs better with increase in window size. This is because, with increase in window size the number of packets sent into the simultaneously increases. Hence, the receiver will be able to receive more data in less time. In the case of loss, only the lost packet will be retransmitted, this results in efficient use of the bandwidth of the network. Opposite to this, in GBN, for a single packet loss, all the packets following the lost packet are retransmitted, this results in misuse of the bandwidth of the network and more time is needed to receive the data in order. Hence, the performance of GBN protocol reduces with increase in window size.

Conclusion

For this programming assignment, we observed various characteristics of the three protocols named Alternating Bit Protocol, Go Back N protocol and Selective Repeat Protocol.

The Alternating Bit Protocol performs good in the case of no loss environment, but its performance can be improved by pipelining the packets. This is done in both Go Back N and Selective Repeat protocols. However, Go Back N works with cumulative acknowledgments and to maintain the in order delivery of packets, it retransmits all the unacknowledged packets when timeout event happens. The performance of GBN protocol is good when the window size is small and the performance degrades as the size increases.

Selective Repeat protocol is an improvement over the Go Back N protocol as instead of discarding the out of order packets, it buffers them if they lie within the receive window. And in lossy environment, only the lost packet is retransmitted. Hence, the performance of SR protocol increases with increase in window size.

References

[1] Computer Networking – A top down approach by Kurose and Ross.

[2] G. Varghese and A. Lauck, “Hashed and hierarchical timing wheels: Data structures for the efficient implementation of a timer facility”.