CSE 589 Programming Assignment 2 Report

"I have read and understood the course academic integrity policy"

Abbreviations:
ABT - Alternating BiT protocol
GBN - Go Back N protocol
SR - Selective Repeat protocol
RTT - Round Trip Time

1. Brief description of the timeout scheme you used in your protocol implementation and why you chose that scheme.

Initially, I tried an adaptive timeout based on RTT as in TCP. In case of timeout, I doubled the timeout and retransmitted the packets. But I found only few packets reaching B. So, I decided to use a static timeout calculated using the formula,

Timeout = Average RTT + Constant delay

where, *Average RTT* = twice the average one way delay (2*5 = 10 time units)
Constant delay = additional constant time delay calculated empirically

I tried with 0.1*RTT(1 time units), 1*RTT(10 time units) and 1.5*RTT(15 time units) as additional constant delay values. For ABT, an additional delay of 1 time unit worked well. For GBN and SR, when I chose a delay of 1 time unit, there were too many retransmissions and it took a lot of time for the simulation to complete(20 hours for SR!!!). With 10 time units, it improved. 15 time units turned out to be pretty good. Beyond this, number of packets reaching B started decreasing. So, finalized timeout values for the three protocols are as follows,

Protocol	Constant delay(Time units)	Timeout(Time units)
ABT	1	11
GBN	15	25
SR	15	25

2. Brief description (including references to the corresponding variables and data structures in your code) of how you implemented multiple software timers in SR using a single hardware timer.

When a packet is transmitted, a new TIMER instance will be created.

```
typedef struct timer
{
        int pkt_id;  // sequence number of the packet
        float start_time; // Time at which the packet is transmitted
        float end_time; // Time at which the timer will expire(start_time + timeout)
} TIMER;
```

It will be added to the TIMER_LIST which is a linked list that contains the TIMER instances of all the packets in flight.

```
typedef struct timer_node
{
        TIMER timer; // TIMER instance
        struct timer_node* next; // Pointer to the next TIMER_NODE node
} TIMER_LIST, TIMER_NODE;
```

I will keep track of the currently running TIMER in a global variable. This is the actual timer running on the hardware timer.

Working of the timer:

- When a new timer needs to be scheduled, if no timer is running currently, it will be scheduled instantly.
- If there is a timer already running, it will just be added to the TIMER_LIST.
- Once a timer expires, the node corresponding to the expired timer will be removed from the TIMER_LIST(usually, head of the list).
- If there are any pending timers to be scheduled, the timer at the head of the list will be scheduled with new timeout calculated as follows,

  new_timeout = current_time - timer.start_time
  timer.start_time = current_time
  timer.end_time = timer.start_time + new_timeout
- The expired packet will be rescheduled and the timer will be added to the list.
- In case of ACK, if the ACK is for the packet for which the timer is currently running, it will be stopped and next timer will be scheduled as explained before.
- If the ACK is for the packet for which the timer is yet to be scheduled, timer will just be removed from the list.

Sample scenario:

At time t = 1000, timer for packet P1(start_time = 1000) is scheduled with a timeout of 25 time units(end_time = 1025). At time t = 1010, we transmit another packet P2 and since there is a timer running, we add the timer for P2(start_time = 1010, end_time = 1035) to the TIMER_LIST. At time t = 1020, another packet P3 is transmitted (start_time = 1020, end_time = 1045).

At time t = 1025, timer for P1 expires and we schedule P2 with new timeout(start_time = 1025, end_time = 1035, time_out = 10). P1 is retransmitted and rescheduled again(start_time = 1025, end_time = 1050). So, the current timer is for P2.

At time t = 1030, ACK for P2 is received. Since the current timer is for P2, we stop it and schedule the next timer P3 with new timeout(start_time = 1030, end_time = 1045, time_out = 15).

At time t = 1035, ACK for P1 is received. Since timer is yet to be scheduled for P1, it is enough that we remove the timer from the list. Note that the current timer is for P3.

At time t = 1040, ACK for P3 is received. The current timer is for P3 and so it is stopped. Since there are no more timers to schedule, we do nothing.

Performance Comparison

Experiment 1

In this experiment, we study the effect of loss (loss probabilities = {0.1, 0.2, 0.4, 0.6, 0.8}) on throughput for various window sizes {10, 50} for each of the three protocols - ABT, GBN and SR.

Figure 1: Loss probability Vs Throughput for window size = 10

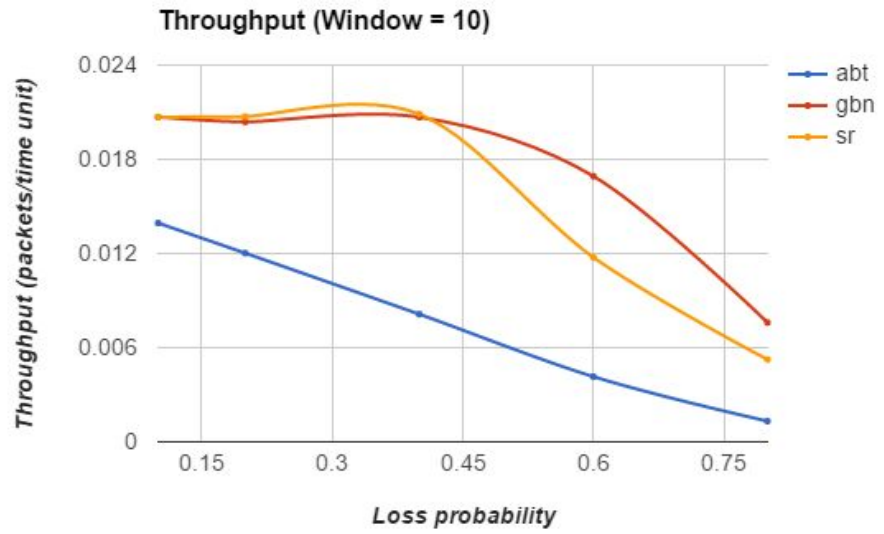
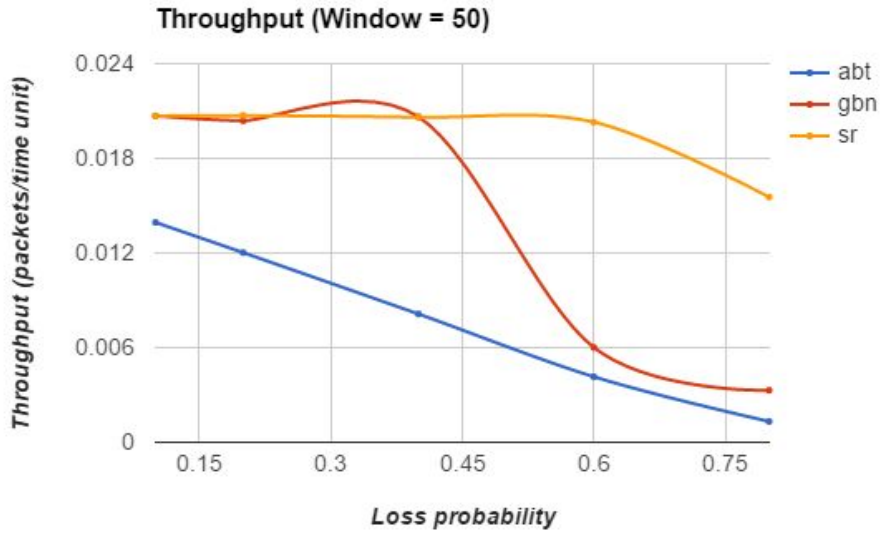


Figure 2: Loss probability Vs Throughput for window size = 50



From the graphs(Figures 1 and 2), we observe that the throughput decreases as loss increases for all the three protocols. This is expected because the packets are lost often in the

channel which, in turn, result in lot of retransmissions after timeout and hence throughput decreases. For ABT, this effect is more pronounced because the protocol design allows only one packet to be transmitted at a time. For GBN, it is less severe compared to ABT because it transmits a window of packets. But still, on timeout, it retransmits the entire window of packets which is inefficient. SR is the most efficient protocol of the three because it transmits a window of packets (as GBN) but retransmits only the packet for which timeout occurred.

In Figure 1, we observe that the throughput of GBN is greater than that of SR in case of heavy packet lossy. It may be because of smaller window size. But in Figure 2, with window size = 50, we can see that SR gives a higher throughput than GBN when the medium is very lossy.

Experiment 2

In this experiment, we study the effect of window size {10, 50, 100, 200, 500} on throughput for various loss probabilities {0.2, 0.5, 0.8} for each of the three protocols - ABT, GBN and SR.

Figure 3: Window size Vs Throughput for loss probability = 0.2

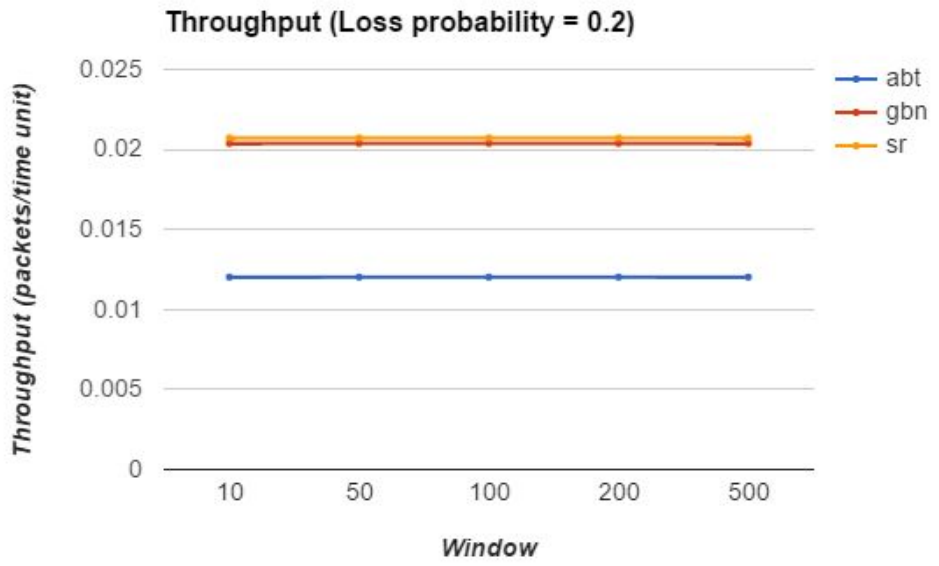


Figure 4: Window size Vs Throughput for loss probability = 0.5

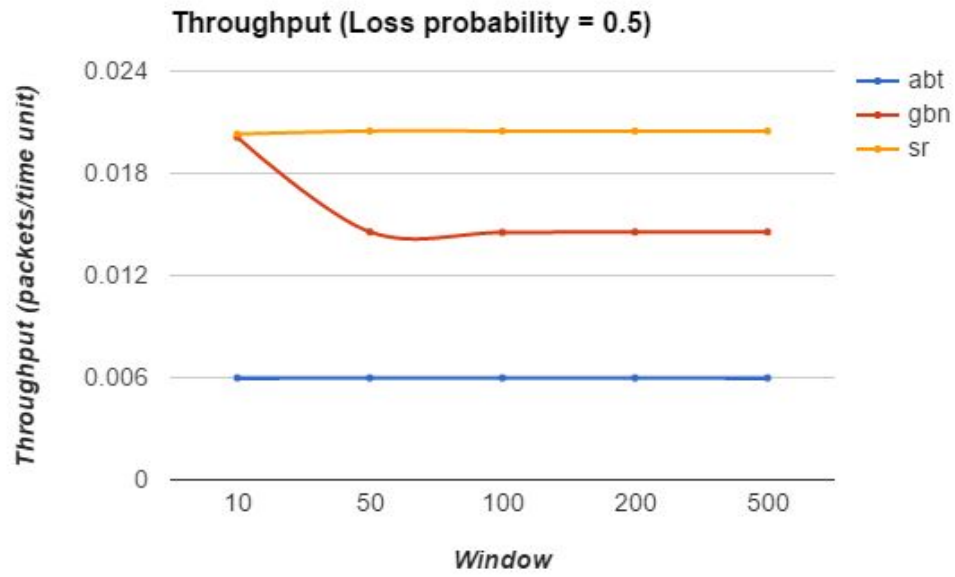
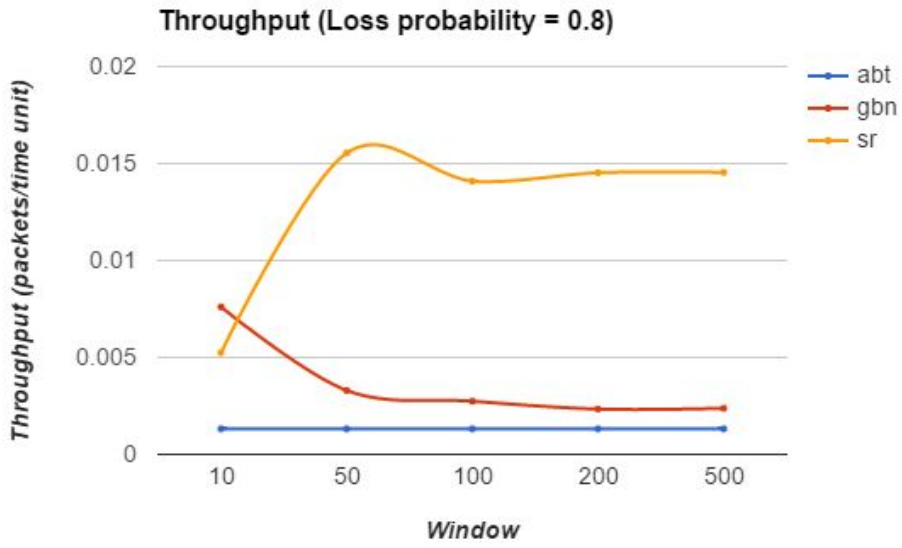


Figure 5: Window size Vs Throughput for loss probability = 0.8



From the graphs(Figures 3, 4 and 5), we observe that throughput of ABT is not affected by window size since it does not have the concept of window. From Figure 3, we infer that even GBN and SR does not have any impact on throughput for various window sizes for very low loss and corruption. But, with higher loss probabilities, GBN and SR has significant difference in throughputs for varied window sizes.

From Figures 4 and 5, we observe that, for GBN, for higher loss probabilities, throughput decreases with increasing window size. This is because, if there is a heavy packet loss and if the window size is large, we have to retransmit large number of packets in case of a timeout. This will cause lot of congestion in the network and decrease the throughput drastically. But, for SR, throughput improves when we increase the window size. Since SR retransmits only one packet on timeout, there is no significant congestion due to retransmission. Also, with large windows, we can transmit many packets. Hence SR has a better throughput for large window sizes and heavy packet losses.

Conclusion

Based on the experiments, among the three protocols - ABT, GBN, SR, I conclude that SR gives better throughput even in case of heavy packet loss and corruption.