Reliable Transport Protocols

Bilin Shi(50291716) & Wenqi Li(50290834)
Department of Computer Science and Engineering, University at Buffalo
{bilinshi,wli3533}@buffalo.edu

November 2018

Contents

Academic Integrity

I have read and understood the course academic integrity policy.

Multiple logical timer on single hardware clock

Data structure

```
//store the packet and the time it was sent by sender
struct pkt_time{
  struct pkt packet;
  float start_time;
};

//The window buffer of the sender.
//Stores each packet that has been sent but not received acknowledge packet and its sendin
vector<pkt_time> A_windows;

int N //The window's size

struct SenderInfo {
    int next_seq; //The sequence number of the next sent packet
    int next_ack; //The ack number of the next sent packet
    //Record the sequence number of the first packet sent in the window buffer
    int base;
    //The sequence number of the largest sequence number acknowledged packet
    int last_rec_seq;
    struct pkt next_packet; //The next sent packet
} A;
```

Implementation

In the SR protocol, we have added a *pkt_time* struct to help us implement a single clock to simulate multiple clocks. There are two variables in this struct, one is the packet itself, and the other is its sending time(We use *get_sim_time* to get the packet's sending time). For each packet that has already been sent, we not only store it in the window buffer of the sender, but also store its sending time in the window buffer. We maintain the window buffer *A_windows* as a priority queue, where the sending time of each packet is marked as priority. When timeout occurs, it means the first packet in window buffer didn't receive the acknowledge packet and it need to be resent, so we update the sending time of the first packet and re-order the window buffer. Now the next timer interrupt will occur at *the sending time of the first packet in window buffer + TIME_OUT*, so we reset the timer with time gap *the sending time of the first packet in window buffer + TIME_OUT - current time*. That is how we implement multiple logical timer using only one hardware timer.

Timeout scheme

We set the same $TIME\_OUT = 30.0$ for all three protocols, the time-out mechanism is shown as follow.

- **Alternating Bit Protocol**: For ABT protocol, there is only one packet waiting for response acknowledge packet at one time, so the timer is set to this packet, when timeout occurs, ABT protocol only retransmit this packet to the receiver.

- **Go Back N Protocol**: For GBN protocol, we have a window buffer in size N, this window buffer stores the last N packets we sent to the receiver and the first packet in the window buffer hasn't received the matched acknowledge packet. We set the timer to the first packet in the window buffer ,and when timeout occurs, GBN protocol retransmit all packets in the window buffer to the receiver.
- **Selective Repeat Protocol**: For the SR protocol, we have a window buffer of size N, which stores the last N packets we sent to the receiver and their transmission time. The first packet in the window buffer was not received. Matching confirmation packet. Our goal is to resend each timeout packet without sending packets that have not timed out. We set the timer to the first packet in the window buffer. When timeout occurs, the SR protocol resends the first packet in the window buffer and restores the packet to the end of the window buffer. And reset the timer to the remaining time in the window for the second packet that did not receive an acknowledgment.

Experiments

Increasing loss probability on window sizes

For all the experiments, we won't change the rate of corruption because in our implementation, if one packet is corrupted, we won't do anything for the corrupted packet, which is just like a lost packet. From the source file *simulator.cpp*, we have the defination of **Throughput** as :

$$Throughput = \frac{\#\ of\ received\ packets\ in\ application\ layer}{total\ running\ time}$$

As suggested the test conducted with the parameters shown in Table 1.

Table 1: Performance of 3 protocols on increasing loss probability

Protocol	Message	Corruption	Loss	Window
Alternating Bit	1000	0.2	$\{0.1, 0.2, 0.4, 0.6, 0.8\}$	$\{10, 50\}$
Go Back N	1000	0.2	$\{0.1, 0.2, 0.4, 0.6, 0.8\}$	$\{10, 50\}$
Selective Repeat	1000	0.2	$\{0.1, 0.2, 0.4, 0.6, 0.8\}$	$\{10, 50\}$

As loss probability increasing, the throughput of three protocols with $WindowSize = 10$ is shown in Figure 1, and the throughput of three protocols with $WindowSize = 50$ is shown in Figure 2.

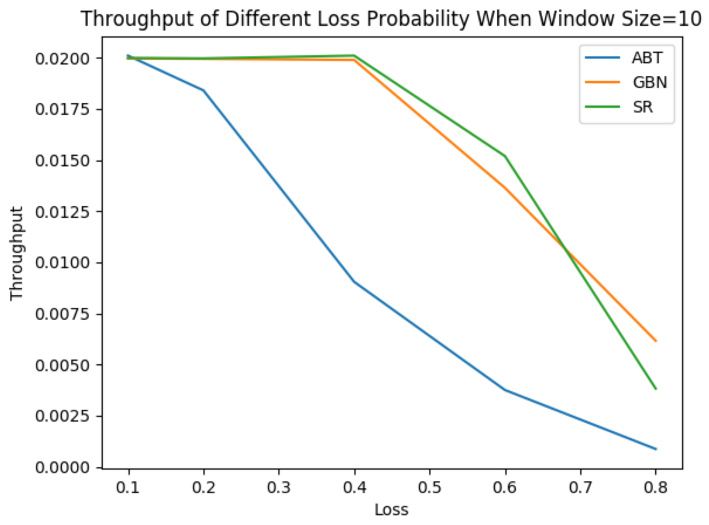

Figure 1: Performance of Different Loss Probability When Window Size=10

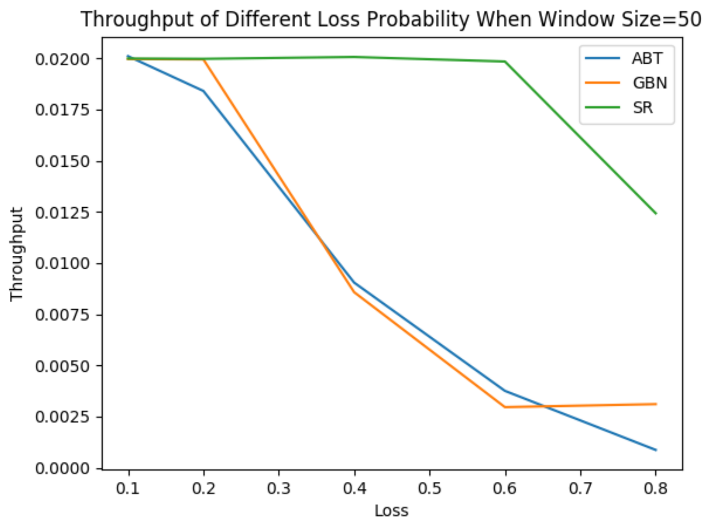

Figure 2: Performance of Different Loss Probability When Window Size=50

As the level of loss increases, the experimental results show that the throughput of the three protocols is decreasing. Compared with ABT and GBN, SR maintains a more stable throughput in the face of increased loss levels. This may be because it selectively performs double buffering and packet transmission on the receiving end and the transmitting end, avoiding unnecessary packet transmission. The GBN needs to resend all window size messages after a timeout, requiring a large amount of communication resources to resend the lost packets. The ABT stop-and-wait mechanism is set so that whenever a packet is lost, it needs to stop and wait for a retransmission. This also wastes communication resources.
When the window size is 50, both GBN and ABT show a significant drop as the loss level exceeds 0.2. According to the transmission method of GBN, when the window size becomes larger, retransmission will

become more frequent, which leads to a decrease in throughput. At the same time, because the window size does not affect the performance of the ABT, the performance of the ABT is similar to the case when the window is 10. Figure 3 and Figure 4 show the total amount of transmitted packets at the transport layer, which is match our expectation. Here we define M as:

$$M = \frac{\#\ of\ transmitted\ packets\ in\ transport\ layer}{\#\ of\ transmitted\ packets\ in\ application\ layer}$$

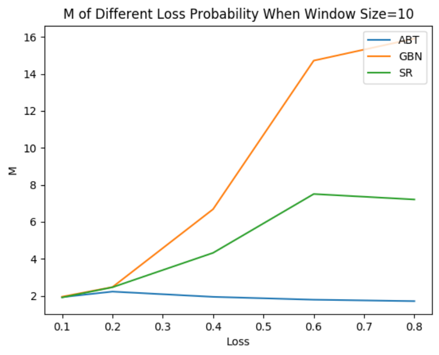

Figure 3: M of Different Loss, Window Size=10

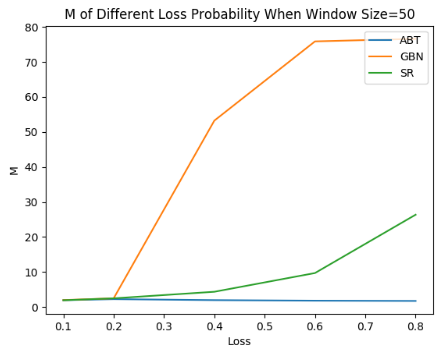

Figure 4: M of Different Loss, Window Size=50

Increasing window sizes on loss probability

As suggested the test conducted with the parameters shown in Table 2.

Table 2: Performance of 3 protocols on increasing window size

Protocol	Message	Corruption	Loss	Window
Alternating Bit	1000	0.2	$\{0.2, 0.5, 0.8\}$	$\{10, 50, 100, 200, 500\}$
Go Back N	1000	0.2	$\{0.2, 0.5, 0.8\}$	$\{10, 50, 100, 200, 500\}$
Selective Repeat	1000	0.2	$\{0.2, 0.5, 0.8\}$	$\{10, 50, 100, 200, 500\}$

The throughput of three protocols in different window size under $Loss = 0.2$ is shown in Figure 5, under $Loss = 0.5$ is shown in Figure 6, and under $Loss = 0.8$ is shown in Figure 7.

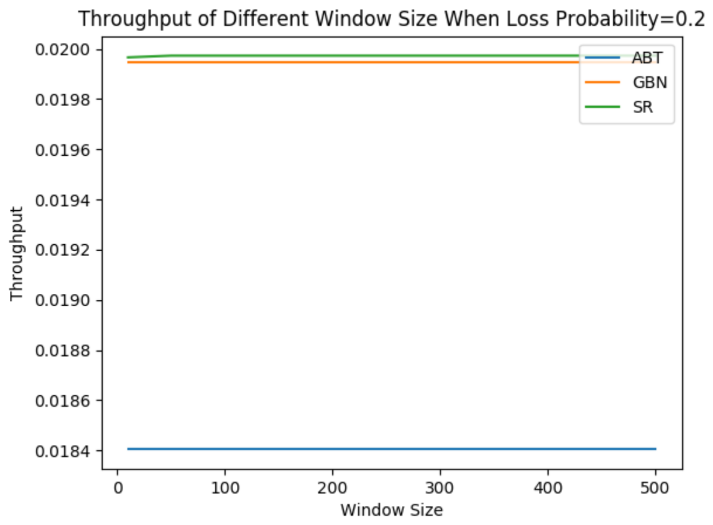

Figure 5: Performance of Different Window Size When Loss Probability=0.2

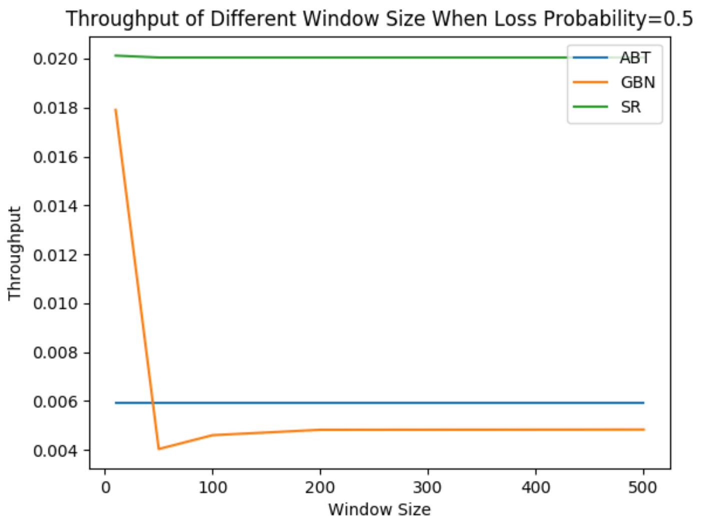

Figure 6: Performance of Different Window Size When Loss Probability=0.5

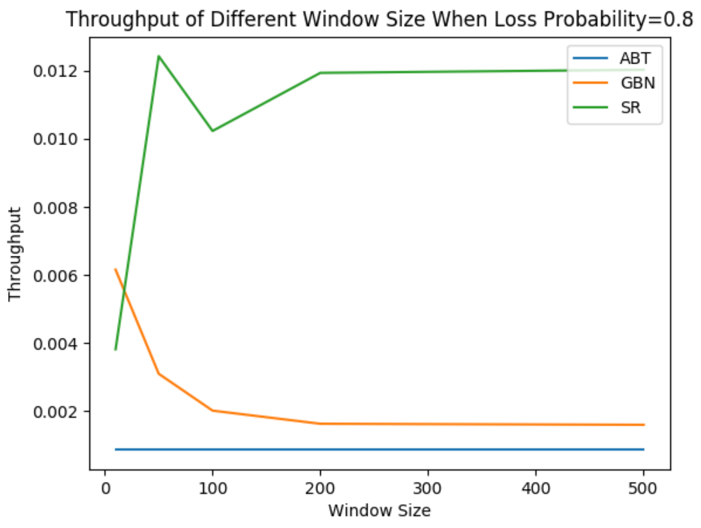

Figure 7: Performance of Different Window Size When Loss Probability=0.8

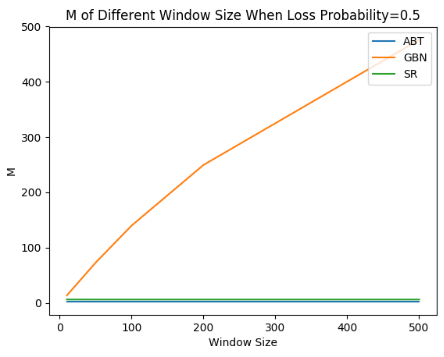

Figure 8: M of Different Window Size, Loss=0.5

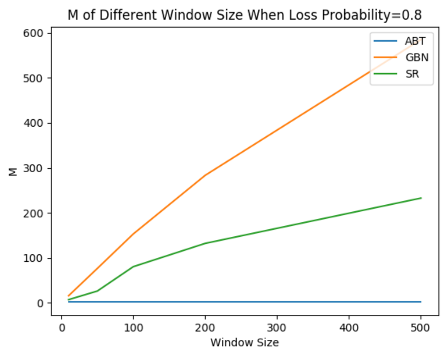

Figure 9: M of Different Window Size, Loss=0.8

As can be seen from the figures, the throughput rate of ABT has been kept at a constant state, which is consistent with our expectation, because the throughput rate of ABT is not affected by the change of window size. As the window size increases, SR and GBN show the same stability performance. We can see from Figure 7, although the throughput of the selective repeat protocol is relatively low when the window size is set small, the SR throughput becomes better and more stable as the window size increases. This may be due to an increase in the transmission frequency of the packet, which reduces the frequency of timeouts. As the window size increases, the performance of the GBN deteriorates in Figure 6 and Figure 7. This may be because when a packet is lost, the GBN needs to retransmit more packets than the SR, which again leads to a higher risk of packet loss. And we can see in Figure 8 and Figure 9, the M of GBN is much higher than SR, which confirms our conjecture.