IMPLEMENTING RELIABLE TRANSFER PROTOCOLS

PROGRAMMING ASSIGNMENT 2 - CSE 589

ANALYSIS

By

VIVEKANANDH VEL RATHINAM

I have read and understood the course academic integrity policy located under this link:
http://www.cse.buffalo.edu/faculty/dimitrio/courses/cse4589_f14/index.html#integrity

Description of the timeout scheme used in my implementation:

1) Alternating bit protocol:

I start the timer and set the pending acknowledgement flag to 1 when I send a packet from A to B. The Pending Acknowledgement flag toggles between 0 and 1. When A receives a packet from the upper layer, it checks if the Acknowledgement is received and then proceeds to send the packet to layer 3, otherwise it discards the packet.

When an acknowledgement is received, the Pending Acknowledgement flag is set to 0, indicating that A can send a new packet.

When a timer interrupts, A restarts its timer and resends the last sent packet to B

2) Go Back N protocol:

The timeout scheme is implemented as exactly the same in the textbook. I start the timer for the packet with the smallest sequence number(base). If I receive an acknowledgement for all the packets that are sent, the timer is stopped or else the timer is restarted for the packet the smallest sequence number(base).

When the timer interrupts, the timer is restarted again and all the packets starting from the base to the next sequence number are retransmitted.

3) Selective Repeat protocol:

The timeout scheme used for Selective Repeat is taken from the paper "Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility". I have used a straight forward scheme which interrupts the sender for each time unit, where we can reduce the time remaining value for each of the packets in the sliding window and perform the EXPIRY PROCESSING for the packets where the time remaining is 0. More details are given in the next section.

Implementation of multiple software timers in Selective Repeat using a single hardware timer:

I have used a straightforward approach in implementing the multiple software timers.

A user defined structure “packet_with_time” is created for storing the retransmission time along with the packet in the sliding window. Also a new field “ack_received” is included along with the packet to mark the packet if the acknowledgement has been received.

```
struct packet_with_time{
      float retranmsit_time;
      int ack_received;
      struct pkt packet;
};
```

The sliding window is implemented as a **circular array** using a pointer to the structure “packet_with_time” with a size of WINSIZE(size of the sliding window)

```
struct packet_with_time* buffer_for_window = null;

void buffer_init(){

buffer_for_window  =  (struct  packet_with_time*)realloc(buffer_for_window,
WINSIZE * sizeof(struct packet_with_time));

}
```

I have used two variables **head_of_buffer** and **tail_of_buffer** for keeping track of the base and the next sequence number in the circular array.

```
int head_of_buffer, tail_of_buffer;
```

I start the timer with an increment of 1 `starttimer(0, 1)` during the A_init() method as well as in the A_timerinterrupt() method. Therefore, for each time unit the timer is interrupted and the A_timerinterrupt() method is invoked.

When A sends a packet to B, it stores the packet in the sliding window with retransmission time equal to **time + TIMEOUT.** The **time** is a global variable which includes the current time in time units.

In each invocation of A_timerinterrupt(), A checks whether the retransmission time of the packet is less than or equal to global variable **time.** If so, it is time for retransmission of the packet and resends the packet again**.**

```
/* called when A's timer goes off */
void A_timerinterrupt()
{
   //Declaring local variables
   int y = 0;

   //Retransmitting the unacked packets
   for(y = base; y < nextSeqNo; y++){
        if(buffer_for_window[y % WINSIZE].ack_received == 0 &&
           buffer_for_window[y % WINSIZE].retranmsit_time <= time){

           printf("A:Retransmitting packet with Seq No: %d\n",y);

           //Setting the new retrasmit time for the packet
           buffer_for_window[y % WINSIZE].retranmsit_time = time + TIMEOUT;

           //Sending the message to layer 3
           tolayer3(0, buffer_for_window[y % WINSIZE].packet);
        }
    }

    //Starting the timer again
    starttimer(0, 1);
}
```

After retransmitting the packet, A again resets the retransmission time to be equal to time + TIMEOUT.

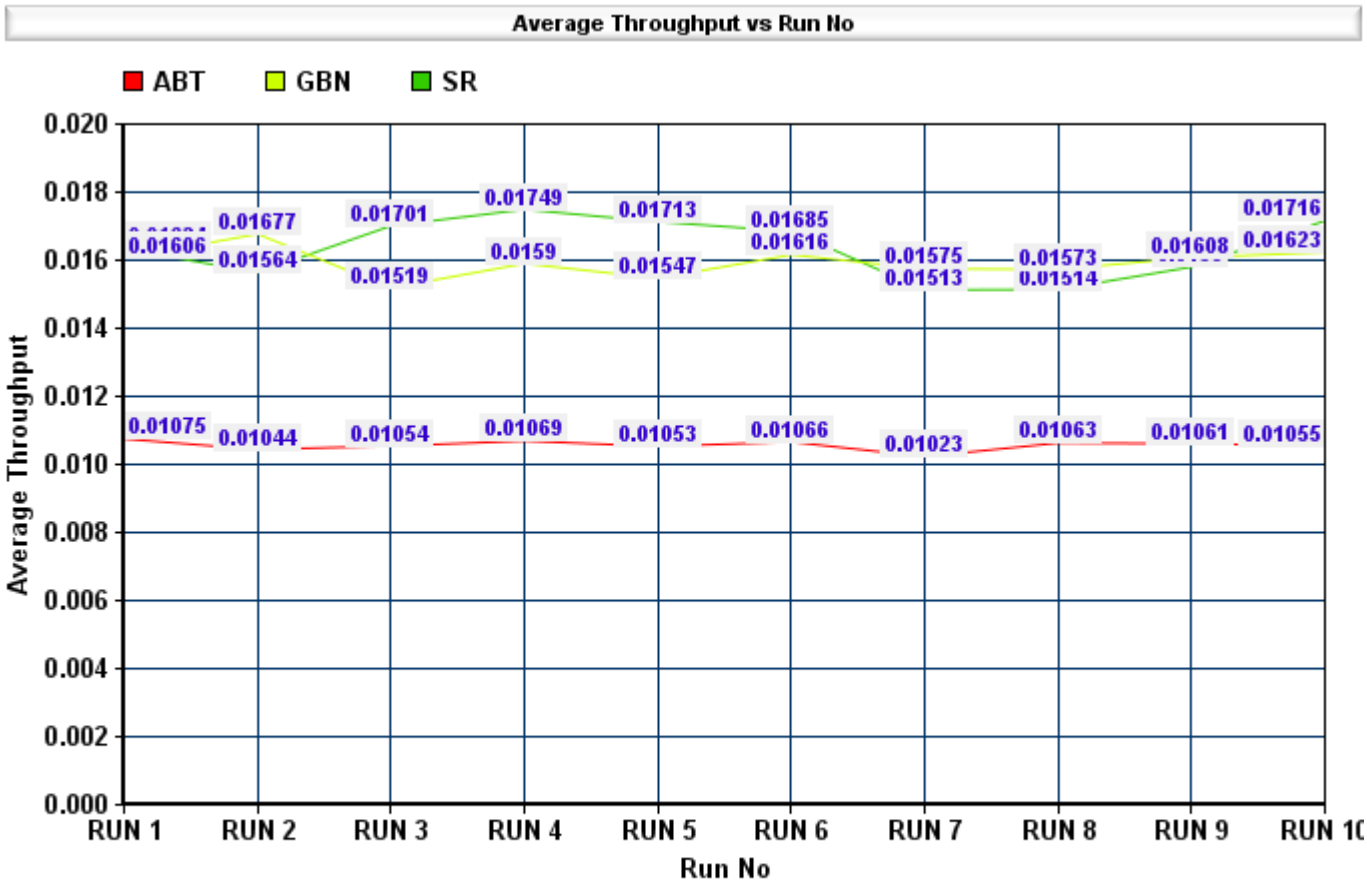
When an acknowledgement is received in the A_input() method, the **base** is updated to be equal to the next smallest unacknowledged packet in the sliding window or the next sequence number of the sender. The messages in the buffer are transmitted as the window has been moved

```
//Updating the lowest unacked packet number
if(base == packet.acknum){
   for(z = base+1; z <= nextSeqNo; z++){
     if(buffer_for_window[z % WINSIZE].ack_received == 0 || z == nextSeqNo)
     {
        printf("A:Updating base from %d to %d\n",base,z);
        base = z;
        A_sendMessagesFromBufferToLayer3();//Sending messages to layer 3
        break;
     }
   }
}
```

Experiment 1 – Observations

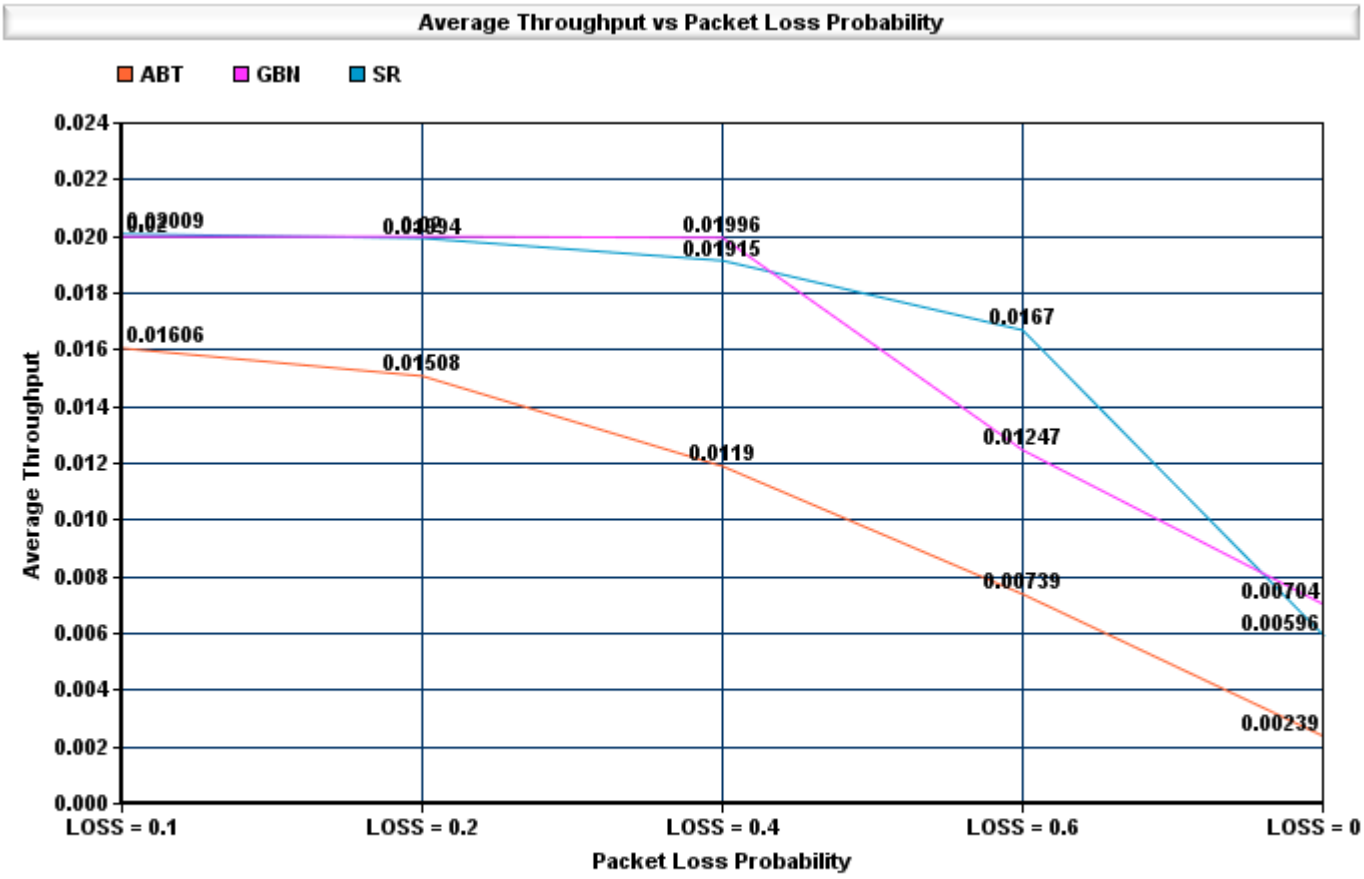
A) Average Throughput vs Run Number

Experiments were conducted using different seeds 1234, 1111, 2222, 3333, 4444, 5555, 6666, 7777, 8888 and 9999. For the Alternating Bit Protocol, the throughput remains in the range of 0.010 to 0.012. For the other two protocols, the throughput seems to vary for different seed numbers. It clearly indicates that the average throughputs are different and does not have a direct relationship with seed numbers as the program creates randomness in the network based on the seed numbers.



B) Average Throughput vs Packet Loss Probability

The below graph indicates that as the packet loss probability increases, the average throughput decreases regardless of the protocol used. It also suggests us that the GBN protocol and the Selective Repeat consistently performs well than the Alternating Bit protocol regardless of the packet loss probability.



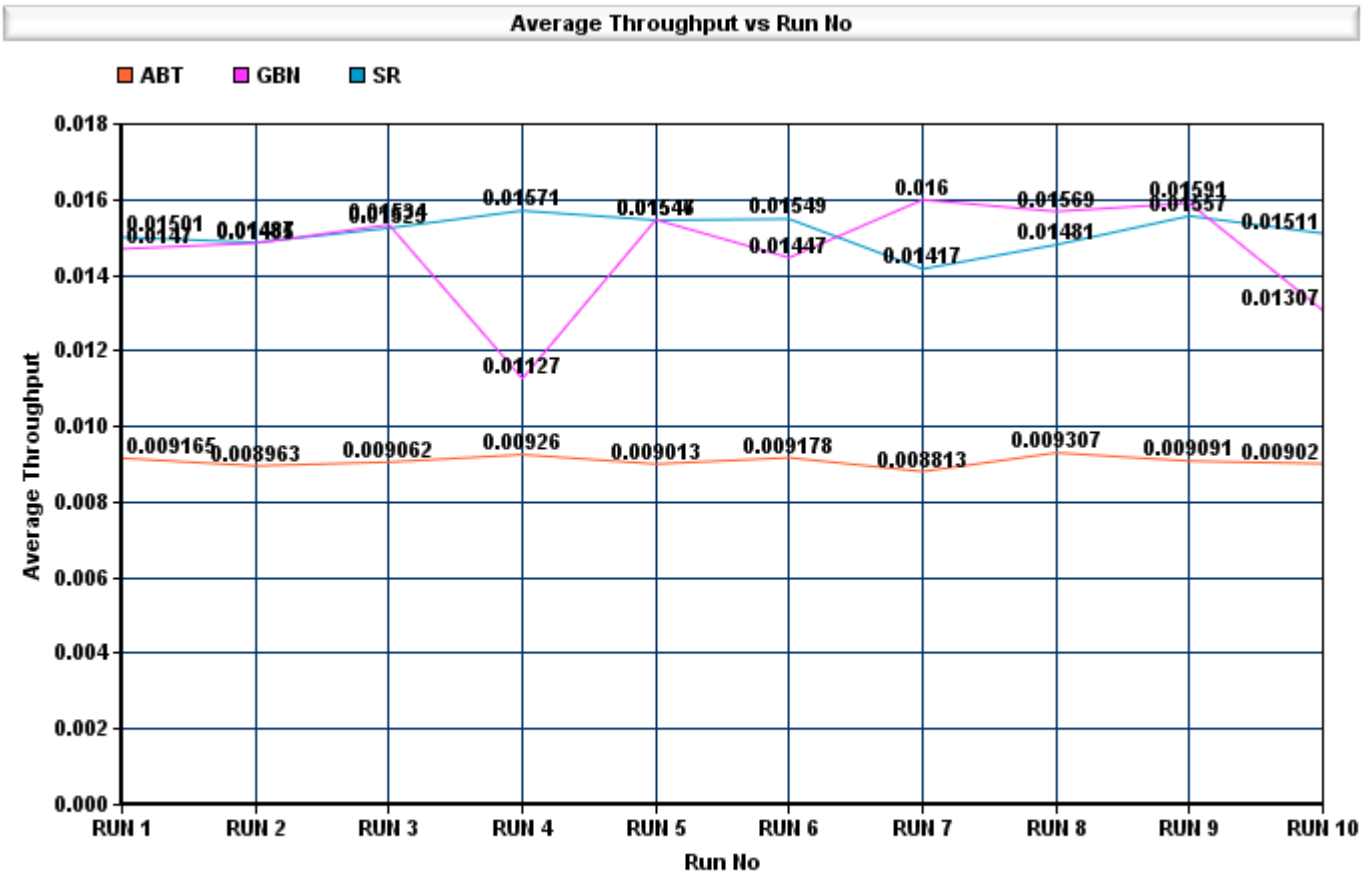
Experiment 2 – Observations

Experiments were conducted for window sizes 10, 50, 100, 200 and 500 with loss probabilities of 0.2, 0.5 and 0.8.

Note: The experiments using window sizes 200 and 500 with packet loss probabilities of 0.5 and 0.8 took a long time to run

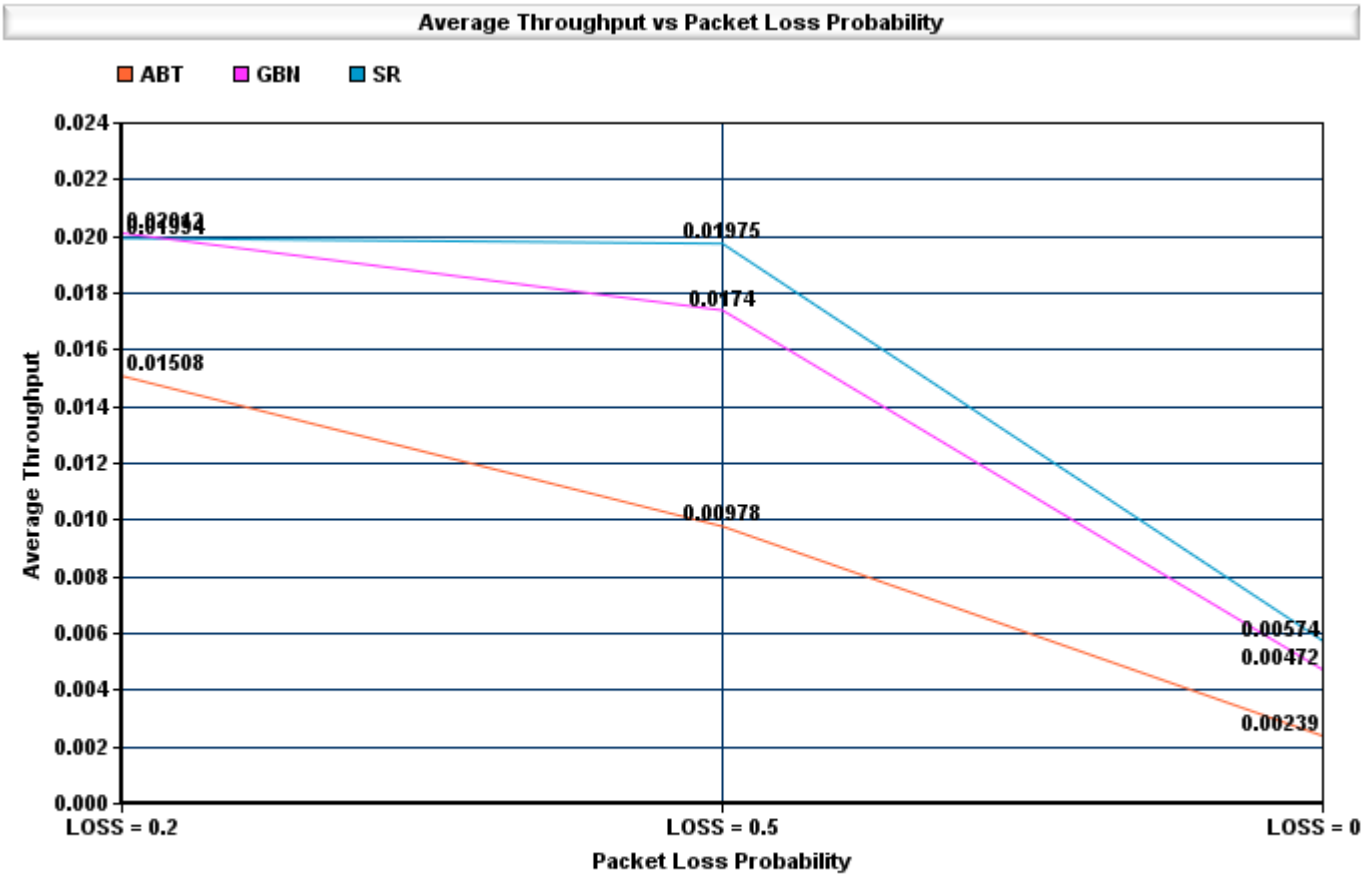
A) Average Throughput vs Run Number

The Go Back N protocol gives a lower throughput of 0.01127 for the seed value of 3333 and for every other runs, the throughput remains in the range 0.014 to 0.016. There is also the same level of randomness as indicated in the first experiment.



B) Average Throughput vs Packet Loss Probability

Same results are observed as in experiment 1 for different loss probabilities, where the performance of the GBN protocol and the Selective Repeat protocol is higher than the Alternating Bit protocol regardless of the packet loss probability.



C) Average Throughput vs Window Size

The average throughput was measured for different sizes of the sliding window (10, 50, 100, 200, 500) for the GBN and the Selective Repeat protocols. The performance of Selective Repeat remains higher than Go Back N protocol regardless of different window sizes.

Note: The experiment could not be run for the window size 500 and loss probability 0.8 for the Go Back N protocol, which suggests a minor flaw in my code. This lack of data which would have given an average throughput lesser than 0.01326 could not be accounted, and the resulting calculation suggested a higher throughput of 0.01794