CSE 589: PA2 Report

Minfan Wang

#50168833

Academic statement: I (We) have read and understood the course academic integrity policy located under this link: http://www.cse.buffalo.edu/faculty/dimitrio/courses/cse4589_s16/index.html - integrity.

➢ Part 1: Description of timeout scheme

In my code, all the timeout values are set as fixed float numbers.

1. **ABT:** Timeout is set as set as 10 time units. ABT is a kind of stop and wait mechanism, the next packet has to wait for the ACK of the first transferred one, so that we have to make as many retransmissions as possible before the next message input from application layer, so that kind of guaranteed transmission can be realized. As we know that RTT (Round Trip Time) for a packet is about 5 time units, consider valid retransmission efficiency and guaranteed transmission in, I set the timeout for timer as 10 time units.
   Timeout: If A can receive the ACK during timeout period, timer will be stopped, so that if timeout happens, it means packet or ACK corrupt, we need to retransfer corresponding packet and start timer again.

2. **GBN:** Time out is set as 15 time units. Timer for GBN is set for packet which has the send base – the last unacknowledged send packet, similar to ABT, we also want to manage guaranteed transmission, if packet lost, we can manage retransmission as soon as possible. However, difference exits because of the mechanism, GBN has to retransmission packets in the whole transmission window, so that if timeout is very small, invalid retransmission can be made which will cause great bandwidth waste and long transmission time, so that I timeout for GBN is higher than ABT.
   Timeout: Start timer again and retransmit all the packets in the send window *[send_banse, next_seq_num -1].*

3. **SR:** Time out is set as 10 time units. I set the system timer as trigger to invoke the count of simulate timer (interrupt). In order to update the simulate timers fast, time out for system timer has to be small so that it will works more efficiently. Simulate time out is 10 time units because in SR, all packets have a timer, it works similar to ABT, we have to retransmit as fast as possible but not that much invalid retransmission. Since we only need to retransmit one packet if timeout or packet loss, the timeout can be set smaller than GBN.
   Timeout: Counter value is 0, retransmit packet corresponding to the timer ID.
   Details descript below.

➢ Part 2: Description of multiple timers in SR implementation

First of all, if we call system timer with *starttimer(AorB, TIMEOUT)* function, every time when time out(*TIMEOUT* equals to zero), timer interrupt will be called. Since only one system timer can work at one time, we can set the timeout very small and restart it when time out so that the system timer can work as trigger, we can utilize the trigger to implement simulate timers.

I define a timer structure to represent simulate timers as below:

```
#define TRUE 1      //flag value for start the timer
#define FALSE 0     //flag value for stop the simulate timers
#define MAX_TIMER_NUM 500
struct timer
{
   int inuse; //flag for call the timer in use
   float counter; //timer out value, every time when timeintterup called, check the value to do corresponding response
   int number; //corresponding to the send_base, every packet has a timer
} timers[MAX_TIMER_NUM];
```

Timers[ID] represent timer with *ID,* and in this program, *ID* is connected with the next transport sequence number, *MAX_TIMER_NUM* is corresponding to the window size, because the largest window size we will use is 500, *MAX_TIMER_NUM* is 500. But the number of timers been used is corresponding to the present window size. And when the packet with send base has been acknowledged, the corresponding timer will be stopped and wait to be use in further packets. Timer for *next_seq_num* packet can be get by *timer_num = next_seq_num%N.*

Before the program runs, we need to initialize the timers, all timers shut down.

```
void timers_init(){
   struct   timer *t;
   for(t=timers;t<&timers[MAX_TIMER_NUM];t++){
      t->inuse = FALSE;
      t->counter = 0;
   }
}
```

Code below is how to start a timer. When do *A_init(),* I start the system timer to keep check and update the data of each simulate timer.

```
timer_num = next_seq_num%N; //find corresponding timer ID
timers[timer_num].inuse = TRUE; //set start flag as TRUE
timers[timer_num].counter = COUNTER; //set time out, COUNTER equals to 10
```

Every time when system time out, interrupt will check the state and update information, if simulate time out, retransmission.

```
void A_timerinterrupt()
{
   struct timer *t;
   for(t=timers;t<&timers[MAX_TIMER_NUM];t++){//loop though every timer and check the value
```

```
        if(t->inuse){
            if(t->counter!=0){//unexpired
              t->counter--; //counter works like timer out, call 10 times interrupt will cause a started timer time out
            }
            if(t->counter <= 0){//expired
                // do retransmission and start corresponding timer
                    t->counter = COUNTER; //restart the corresponding timer
                    t->inuse = TRUE;
            }
        }
    }
    starttimer(0,TIMEOUT);//restart system timer to maintain the trigger operation
}
```

➢ Part 3: DATA & ANALYSIS

Throughput: In general terms, throughput is the rate of production or the rate at which something can be processed.

When used in the context of communication networks, such as Ethernet or packet radio, throughput or network throughput is the rate of successful message delivery over a communication channel. The data these messages belong to may be delivered over a physical or logical link, or it can pass through a certainnetwork node. Throughput is usually measured in bits per second (bit/s or bps), and sometimes in data packets per second (p/s or pps) or data packets per time slot.

- **Experiment 1**

  With loss probabilities: {0.1, 0.2, 0.4, 0.6, 0.8}, compare the 3 protocols' throughputs at the application layer of receiver B. Use 2 window sizes: {10, 50} for the Go-Back-N version and the Selective-Repeat Version.

- Window size: 10; X-axis: Loss probability; Y-axis: Throughput (ABT, GBN and SR) in one graph/plot.

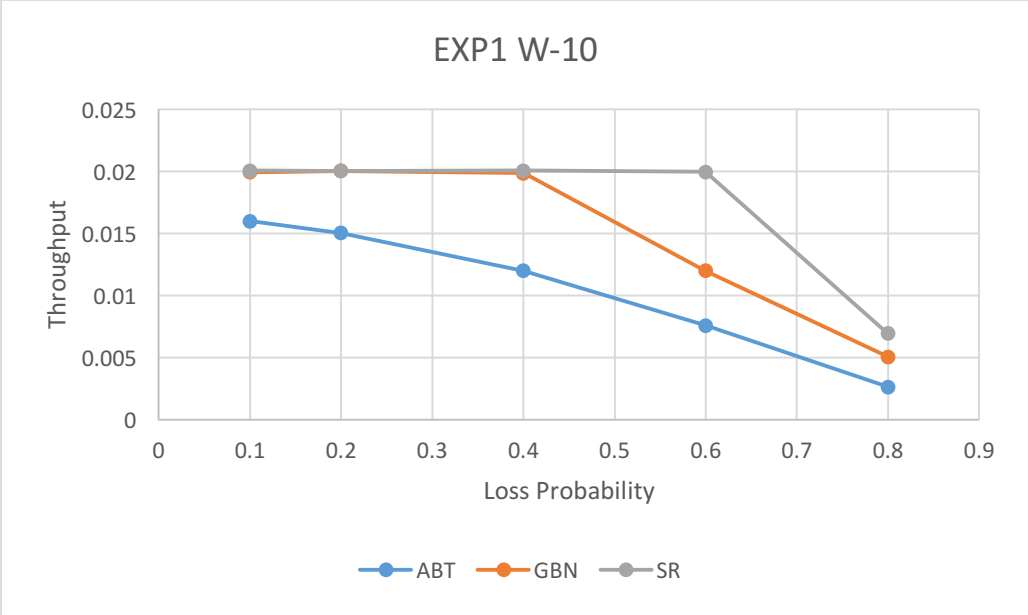

Figure 1 – Expected output of window 10

Expectation: Figure – 1 is the output in theory of how these three protocols work. Since GBN and SR are kinds of guaranteed protocol, if the loss is not that high, for example, less than 0.8 or 0.6, almost all the packets should be received by the receiver. And as loss rise, only few packets get lost. But ABT won't have well performance when loss is great, because only when a packet has been correctly transited the sequence number can move on, under the circumstance of great loss, retransmission is very high so that A will drop many packets in the transfer layer. SR only retransmit unacknowledged packet so that only when loss is super high the throughput will corrupt. But in case of GBN, it need to retransmit all the packets in the transmission window, so that when loss is above 0.6, GBN won't have enough time to manage all the retransmission before simulator time out.

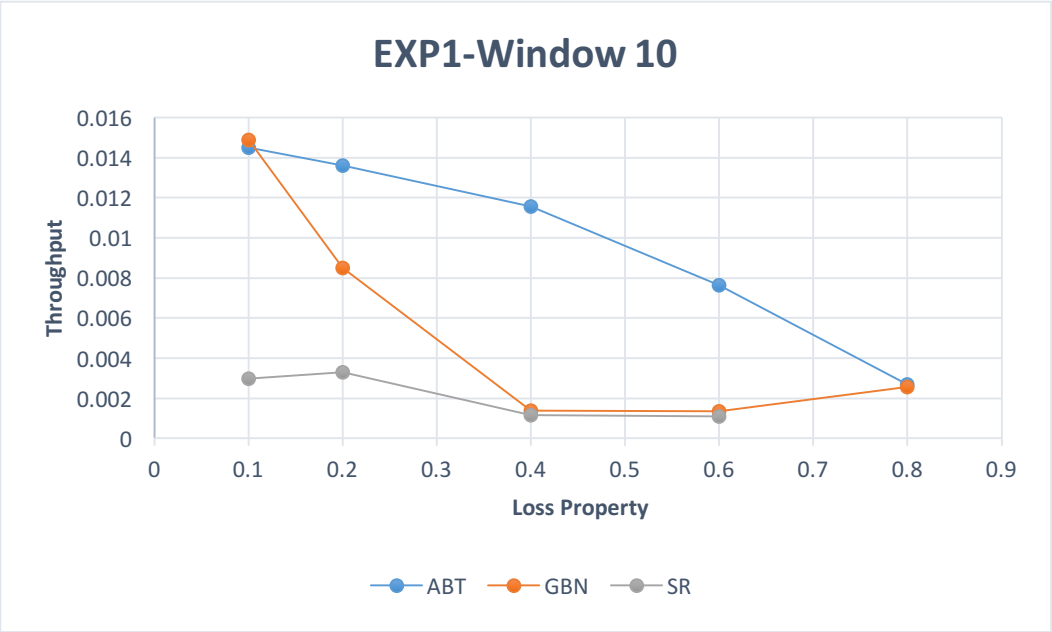

Figure 2 – Experiment1(Window size 10)

Actual output id my experiment is shown as figure – 2, ABT has the highest throughput value, and for all the three protocols, as loss property grows, throughput will fall. And when loss is 0.8, I cannot get the value of the output of SR, but we can know the trend.

Result does not correspond with the expectation. The whole trend of the throughput change with loss property is right but take the working mechanism in to account, SR should have better performance than GBN than ABT, but things works in contrast here. When debug my code， I found when message quantity is not that high, SR do show great performance, not only in operating time but also the throughput. So that I tried again with 200 messages and window size of 10.

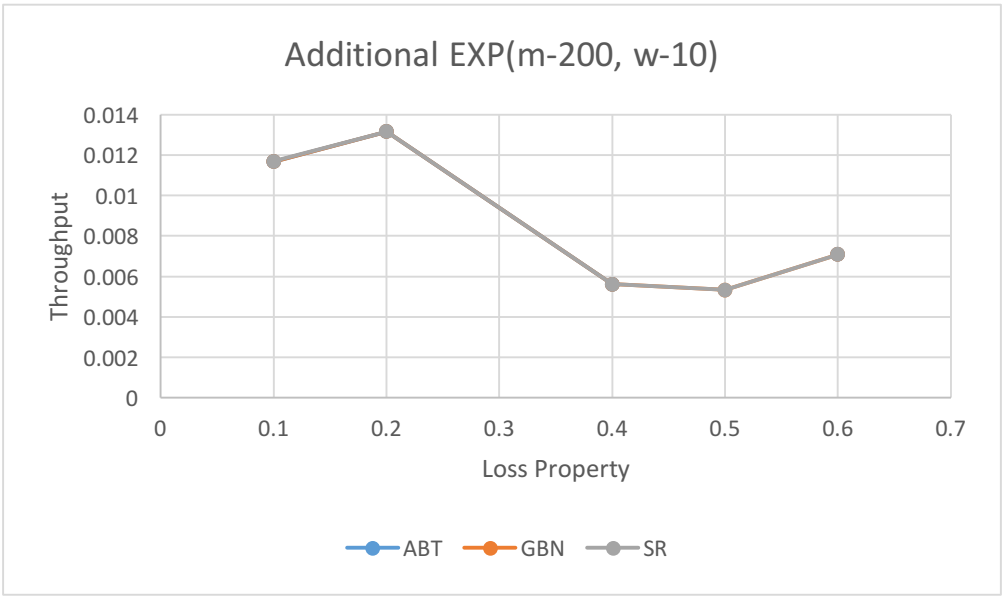

Figure 3 – Extra Experiment

However, the result is still not that well, and all the three protocols almost overlap together. I assume that it may be caused by the amount of messages we need to send.
I checked the data in detail and found something with GBN and SR. Details are as follows:

Run	Messages	Loss	Corruption	Time_bw_m	Application_	Transport_A	Transport_B	Application_	Total_time	Throughput
1	1000	0.1	0.2	50	1000	14800	5225	671	49376.957	0.013589
2	1000	0.1	0.2	50	1000	2856	2588	999	50646.2852	0.019725
3	1000	0.1	0.2	50	1000	30984	8674	114	49057.8672	0.002324
4	1000	0.1	0.2	50	1000	2630	2370	999	49292.9609	0.020267
5	1000	0.1	0.2	50	1000	21197	6583	460	51946.0938	0.008855
6	1000	0.1	0.2	50	1000	29644	8306	162	49895.2891	0.003247
7	1000	0.1	0.2	50	1000	2842	2547	999	49652.832	0.02012
8	1000	0.1	0.2	50	1000	2868	2580	999	49721.1367	0.020092
9	1000	0.1	0.2	50	1000	2940	2651	999	48788.1875	0.020476
10	1000	0.1	0.2	50	1000	2575	2329	999	49862.3516	0.020035

Figure 4 – Data for GBN with window 10

For GBN with low loss and low corruption, only part of the packets can be totally delivered to side of B. If it happens to get the "bad seed", great amount of packets will be retransmitted but the retransmissions were not valid, as we can observe from the data that large amount of retransmission cannot lead to high throughput but low packets can be correctly received.

Run	Messages	Loss	Corruption	Time_bw_m	Application_	Transport_A	Transport_B	Application_	Total_time	Throughput
1	1000	0.1	0.2	50	1000	44362	8414	86	49188.3789	0.001748
2	1000	0.1	0.2	50	1000	34365	7384	284	49945	0.005686
3	1000	0.1	0.2	50	1000	24300	7305	269	49711	0.005411
4	1000	0.1	0.2	50	1000	44855	8673	79	49897	0.001583
5	1000	0.1	0.2	50	1000	38450	8583	38	48033.6602	0.000791
6	1000	0.1	0.2	50	1000	33614	7543	293	51338	0.005707
7	1000	0.1	0.2	50	1000	41266	8985	50	50754.9492	0.000985
8	1000	0.1	0.2	50	1000	33996	7499	267	51481.5586	0.005186
9	1000	0.1	0.2	50	1000	33916	9026	17	49941	0.00034
10	1000	0.1	0.2	50	1000	38836	8697	119	51372	0.002316

Figure 5 – Data for SR with window 10

Run	Messages	Loss	Corruption	Time_bw_m	Application_	Transport_A	Transport_B	Application_	Total_time	Throughput
1	200	0.1	0.2	50	200	24169	1302	86	10399	0.00827
2	200	0.1	0.2	50	200	649	587	198	10048.8242	0.019704
3	200	0.1	0.2	50	200	621	549	199	10065	0.019771
4	200	0.1	0.2	50	200	24974	1447	76	10026	0.00758
5	200	0.1	0.2	50	200	34640	1599	38	9664	0.003932
6	200	0.1	0.2	50	200	552	496	198	10181	0.019448
7	200	0.1	0.2	50	200	30925	1515	50	9906	0.005047
8	200	0.1	0.2	50	200	513	470	199	10455	0.019034
9	200	0.1	0.2	50	200	41481	1799	17	10017	0.001697
10	200	0.1	0.2	50	200	16509	1121	123	9920	0.012399

Figure 6 – Data for SR with 200 messages

In case of SR, when messages amount is low (200), I experienced the same problem with GBN, great amount of invalid retransmissions. When message amount of very high, most of the retransmissions are invalid.
Conclusion: Basic logic of my implementation is right but the retransmission algorithm is ineffective, which caused the result that the simulator has stopped generating messages but the retransmission still

transmission, but since the simulator has stopped, the left transmission cannot be transmitted.

- Window size: 50; X-axis: Loss probability; Y-axis: Throughput (ABT, GBN and SR) in one graph/plot. Expectation: Even though window size will be raised into 50, but the number is not that high, so that the output won't make a big difference. The expected output should look like figure – 7. And the actual output of my code is displayed in figure – 8.

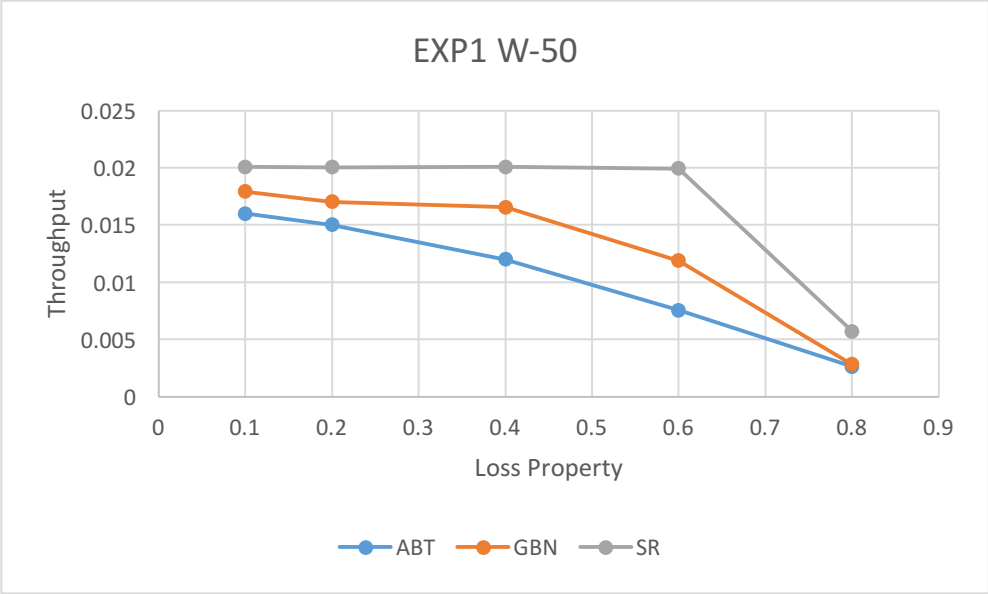

Figure 7 – Output of window size 50

Figure 8 – EXP1 of window size 50

Time for get the result is very long, and always broken pipe when doing the experiment. I can only get part of the data but as we can observe that result is the same as the experiment with widow size 50. And I think the reason is still the same.

- **Experiment 2**

  With window sizes: {10, 50, 100, 200, 500} for GBN and SR, compare the 3 protocols' throughputs at the application layer of receiver B. Use 3 loss probabilities: {0.2, 0.5, 0.8} for all 3 protocols.

  Expectation: Window size is defined in GBN and SR, they utilize pipeline to transmit multiple packets at the same time. So that under the same condition, GBN and SR will show better performance than ABT. And with low loss, no big difference exists between GBN and SR, but since GBN has to retransmit all the packets in transmit window, when loss is very high and window size is very large, GBN will perform not as good as SR.

  About the actual data, I always meet pipeline broken, cannot get all the output so that cannot compare the property of three protocols.

  Expected Graphs

- Loss probability: 0.2; X-axis: Window size; Y-axis: Throughput (ABT, GBN and SR) in one graph/plot.

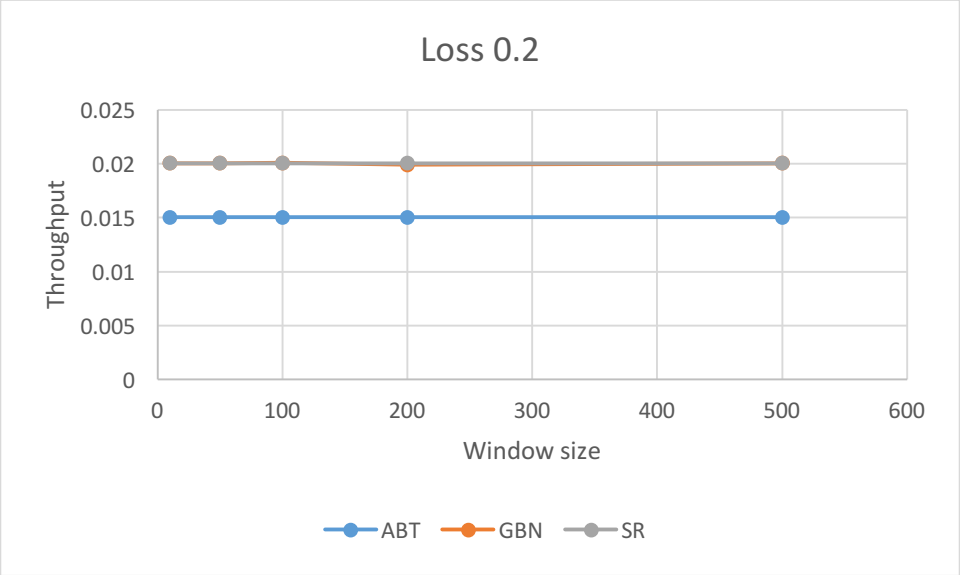

Figure 9 – Theoretical output

Since the loss property is very low, no big difference should exist between GBN and SR. And GBN and SR perform better than ABT.

- Loss probability: 0.5; X-axis: Window size; Y-axis: Throughput (ABT, GBN and SR) in one graph/plot.

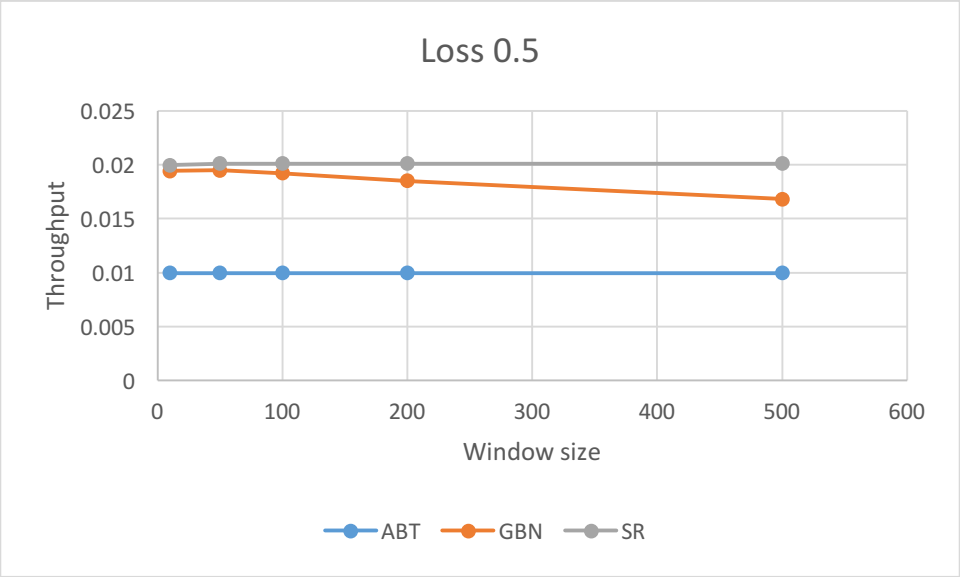

Figure 10 – Theoretical output

When loss is raised to 0.5, as window size increase, retransmissions in GBN will be great so that the throughput will decrease as window size grows. But not that bad.

- Loss probability: 0.8; X-axis: Window size; Y-axis: Throughput (ABT, GBN and SR) in one graph/plot.

Figure 11 – Theoretical output

When the loss is 0.8, all the three protocol will come across great transmission corruption, and in case of GBN, if the window size is very large, the packets for transmission every time is very large, so that great amount of invalid retransmission will be managed, so that it won't perform better than ABT.

➢ Conclusion and Modification of my code

Conclusion: SR has the best performance than GBN than ABT, GBN will cause great bandwidth waste.

Modification of my code:
In SR and GBN, set the content of the buffer as packets, which means when layer 5 deliver messages, layer 4 make the messages into packets and buffer it, so that we don't have to waste the time to make packets every time to layer 3. Because in GBN, we have to transmit great amount of packets.