ECE358: Computer Networks

Winter 2018

Project 2: Data Link Layers and ARQ Protocols

Date of submission:2018-03-13

Submitted by: YuCheng Liu

Student ID: y698liu

Student name: Liu, YuCheng

Waterloo Email address: y698liu@uwaterloo.ca

Marks received:

Marked by:

Table of Contents

ABP

Question 1: Implement the ABP sender

Before implementing the ABP sender, I have analyzed the system and created an event object class to make to ABP sender easier to understand and implement.

```
class Event:
    def __init__(self,itype,time,error_flag,sequence_number):
        self.type= itype
```

```
        self.time = time
        self.error_flag = error_flag
        self.sequence_number = sequence_number
```

The event class is a generalized class to keep track the packet's sequence number, time and status and trigger actions depending on the type of the events.

In order to create an ABP sender, I have separated the work into different steps.

Step 1: Initialization

I created a list that use the same the idea as a DES, which contains all the events related to each packet being transferred. For example, in the ES list, there will be Time-out event and acknowledge events. Then, I have initialized the counter for the total number of packets that is successfully received to 0, and also the current time counter to 0.

```
    # initialization
    ES = []
    SN = 0
    next_expected_ack =(SN+1)%2
    global current_time
    current_time = 0
    packetLength = H+l
    totalSend = 1

    global totalpacket
    global next_expected_ack_Receiver
    totalpacket= 0
    next_expected_ack_Receiver = 0
    timeoutCounter = 0
```

Step 2: Sending the first packet to the receiver

The total time of sending a packet to the receiver and receive an acknowledgement is made up of the transmission time of the data and the header, the propagation time of τ from the sender to the receiver, the transmission time of the acknowledgement and the propagation time of τ from the receiver to the send. On the other hand, a time-out event is defined to happen Δ seconds after the packet has being transmitted. The current time counter will be updated 4 times for each packet being transmitted, and a time-out event will also be added to the DES when the packet is being send. The sender will analyze the result from the receiver and determine weather it is a no-error acknowledgement or an error acknowledgement, or an packet loss which has the type of "NIL".

```
    # H is header length, l is packet length
    current_time = current_time+packetLength/C
    TimeOutEvent = current_time+timeOut
    ES = addTimeOutEvent(ES,TimeOutEvent,SN)
    result = send(current_time,SN,packetLength,BER,tor)
    current_time = result.time
    if(result.type != 'NIL'):
        ES.append(result)
        ES = mergeSort(ES)
```

Step 3: Dequeue the DES and response to the events

Depending on the type of the dequeued events, the ABP sender will trigger different actions. For a time-out event, the ABP sender will have to resend the packet with the sequence number in the time-out event. At the same time, the current time counter will be synchronized with the time in the time-out event, and get updated when resending the packet. After resending the packet, it will analyze the results and add the resulting event back to the event scheduler.

```
        if(i.type == 'TimeOutEvent'):
            timeoutCounter = timeoutCounter + 1
            current_time = i.time+packetLength/C
            TimeOutEvent = current_time+timeOut
            ES = clearTimeOutEvent(ES)
            ES = addTimeOutEvent(ES,TimeOutEvent,i.sequence_number)
            result = send(current_time,i.sequence_number,packetLength,BER,tor)
            totalSend = totalSend+1
            if(result.type != 'NIL'):
                ES.append(result)
                ES = mergeSort(ES)
```

On the other hand, if the event is an acknowledge event, the ABP sender will first check the error status in the event, and it will also check the sequence number of the acknowledge event. If the error status is 0, which means it has no error, and if the next expected to send counter equals to the sequence number in the acknowledge event, it means that the receiver has correctly received the last frame and is ready for the next one. Then, the ABP sender will update the SN counter depending of the situation and also synchronized the current time counter with the acknowledge event time. After the error checking and synchronization, the ABP sender will send the next packet and analyze the response results and add the resulting event back to the event scheduler.

```
        elif (i.type == 'ACKEvent'):
            if(i.error_flag == 0 and next_expected_ack == i.sequence_number):
                if(SN<1):
                    SN = SN + 1
                else:
                    SN = 0
                next_expected_ack =(SN+1)%2
                ES = clearTimeOutEvent(ES)
                current_time = i.time+packetLength/C
                TimeOutEvent = current_time+timeOut
                ES = addTimeOutEvent(ES,TimeOutEvent,SN)
                result = send(current_time,SN,packetLength,BER,tor)
                totalSend = totalSend+1
                if(result.type != 'NIL'):
                    ES.append(result)
                    ES = mergeSort(ES)
```

Note: In order to get the following simulation results and tables, please use python. On command line type in "python lab2.py", three csv files will be generated according to the lab manual.

i) Consider the scenario where BER = 0. Use your simulator to compute the throughput (bits/sec) as a function of Δ for C = 5Mb/s, and two values of 2τ (10ms and 500ms).
ii) Now, repeat the set of experiments in (i) with BER = 1.0e-5 and BER = 1.0e-4.

Δ/τ	2τ = 10ms			2τ = 500ms		
	BER=0.0	BER=1e-5	BER=1e-4	BER=0.0	BER=1e-5	BER=1e-4
2.5	954441.3	814617.4	229635.2	23877.14	20676.07	5712.162
5	954441.3	736943.9	144878.5	23877.14	17162.15	3201.908
7.5	954441.3	672231.1	101611.5	23877.14	16071.87	2150.809
10	954441.3	612058.2	74386.66	23877.14	13786.16	1736.881
12.5	954441.3	555403.6	67825.96	23877.14	13244.76	1373.431

ABP_NAK

Question 2: Implement the ABP_NAK sender

After implementing the ABP sender, the ABP_NAK sender was implemented with a similar strategy. The main different between the two sender is that when reacting to an acknowledge event, instead of ignoring the event if it has an error or the next expected packet SN does not match, the ABP_NAK sender will take these events as negative acknowledgement(NAK) and act on them by resending the same packet as soon as such events occurs. In the implementation, after checking the event is an acknowledgement event and it is not received correctly, the sender will used the SN in the event, which is the old SN, and resend the packet and also synchronized the current time counter with the acknowledge event time. After the error checking and synchronization, the ABP sender will send the next packet and analyze the response results and add the resulting event back to the event scheduler.

```
        elif (i.type == 'ACKEvent'):
            if(i.error_flag == 0 and next_expected_ack == i.sequence_number):
                if(SN<1):
                    SN = SN + 1
                else:
                    SN = 0
                next_expected_ack =(SN+1)%2
                ES = clearTimeOutEvent(ES)
                current_time = i.time+packetLength/C
                TimeOutEvent = current_time+timeOut
                ES = addTimeOutEvent(ES,TimeOutEvent,SN)
```

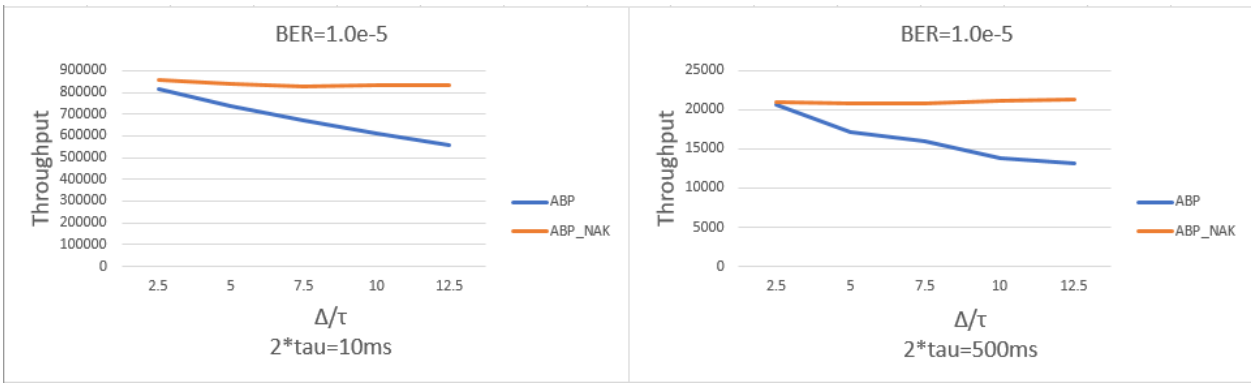
```
            result = sendNACK(current_time,SN,packetLength,BER,tor)

            if(result.type != 'NIL'):
                ES.append(result)
                ES = mergeSort(ES)
        #handling the NAK Case
        else:
            ES = clearTimeOutEvent(ES)
            current_time = i.time+packetLength/C
            TimeOutEvent = current_time+timeOut
            ES = addTimeOutEvent(ES,TimeOutEvent,SN)
            result = sendNACK(current_time,SN,packetLength,BER,tor)
            totalSend = totalSend+1
            if(result.type != 'NIL'):
                ES.append(result)
                ES = mergeSort(ES)
        ES.remove(i)
```

i) Consider the scenario where BER = 0. Use your simulator to compute the throughput (bits/sec) as a function of Δ for C = 5Mb/s, and two values of 2τ (10ms and 500ms).
ii) Now, repeat the set of experiments in (i) with BER = 1.0e-5 and BER = 1.0e-4.

Δ/τ	2τ = 10ms			2τ = 500ms		
	BER=0.0	BER=1e-5	BER=1e-4	BER=0.0	BER=1e-5	BER=1e-4
2.5	954441.3	856769.6	266118.5	23877.14	20981.67	6876.667
5	954441.3	837229.2	253065.7	23877.14	20835.2	6588.205
7.5	954441.3	825641.3	266342.6	23877.14	20817.03	6375.359
10	954441.3	835031.8	257535.1	23877.14	21243.01	6595.164
12.5	954441.3	831394.9	245605.4	23877.14	21261.92	6065.275

a) BER = 0, compare ABP and ABP_NAK

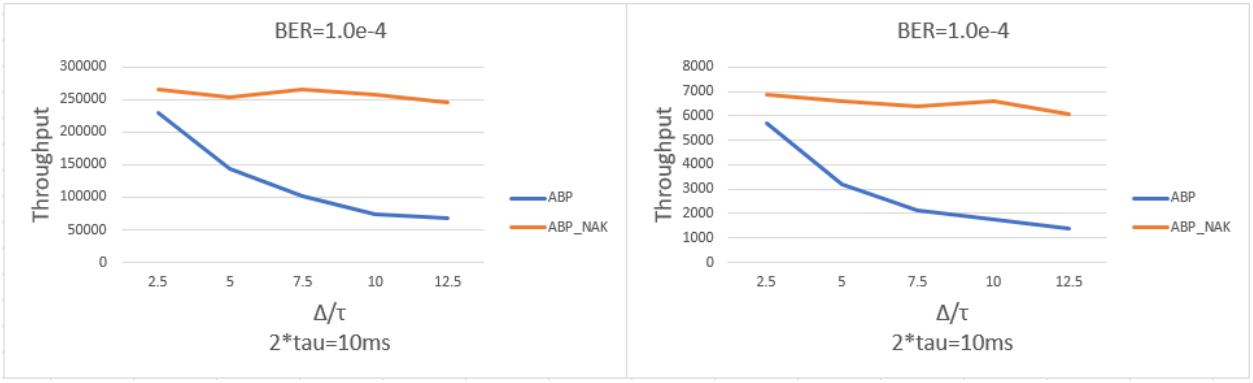
From the graph, we can see that with the BER = 0, the throughput of the ABP sender and the ABP_NAK sender stays the same and equal to each other. This is an expected result, because with BER =0, it means that all the packets will be send without error, so there will not be any timeout event happening. This is also the reason for that when the timeout time changes, the throughput for both sender stays consistent.



From the graph, we can see that with the BER = 1.0e-5, the throughput of the ABP sender decreases significantly as the timeout time increases and the ABP_NAK sender only dropped a little bit and stayed consistent. This is an expected result, because with BER =1.0e-5, it means that all the packets will have a probability of $(1-0.00001)^{12432} = 0.8830$ having no error, and a probability of

$$\binom{12432}{1}\times 0.00001^{1}(1-0.00001)^{12432-1}+\binom{12432}{2}\times 0.00001^{2}(1-0.00001)^{12432-2}+$$
$$\binom{12432}{3}\times 0.00001^{3}(1-0.00001)^{12432-3}+\binom{12432}{4}\times 0.00001^{4}(1-0.00001)^{12432-4} = 0.116$$

having error bits less than 5, and a probability of 1-0.8830 -0.116=0.001 with the packet being lost. Thus, ABP_NAK is saving more time by reacting to the errors and resend the packets intermediately. From analyzing the probability of the ABP and ABP_NAK, we can see that ABP has a probability of 0.116+0.001 = 0.117 encountering a timeout event, and where ABP_NAK only has a probability of 0.001 encountering a timeout event. Thus, as the timeout time increases, the difference become more recognizable and ABP sender's throughput decreases dramatically.



From the graph, we can see that with the BER = 1.0e-4, the throughput of the ABP sender decreases even more significantly as the timeout time increases compare to BER=1.0e-5. However, the ABP_NAK sender remains consistent as the timeout time increases. This is an expected result, we can prove this by calculating the probability again. When BER =1.0e-4, it means that all the

packets will have a probability of $(1-0.0001)^{12432} = 0.2884$ having no error, and a probability of $\binom{12432}{1} \times 0.0001^1 (1-0.0001)^{12432-1} + \binom{12432}{2} \times 0.0001^2 (1-0.0001)^{12432-2} + \binom{12432}{3} \times 0.0001^3 (1-0.0001)^{12432-3} + \binom{12432}{4} \times 0.0001^4 (1-0.0001)^{12432-4} = 0.7026$ having error bits less than 5, and a probability of 1-0.7026-0.2884 =0.009 with the packet being lost. Thus, ABP_NAK is saving more time by reacting to the errors and resend the packets intermediately. From analyzing the probability of the ABP and ABP_NAK, we can see that ABP has a probability of 0.7026+0.009 = 0.7116 encountering a timeout event, and where ABP_NAK only has a probability of 0.009 encountering a timeout event. Thus, since the difference of the two probability is even bigger than BER=1.0e-4, so the results become more obvious。

GBN

Question 3: Implement the GBN sender

When implementing the GBN sender, I was able to build it from changing the ABP sender. In general, the major changes were creating a queue that keep tracks of all the packets being processed. The queue consists of the time, the length and the SN of the packets. With this queue, the GBN sender will be able to send multiple packets depending on the window size, before waiting for an acknowledgement for the oldest packets. From the implementation point of view, a new condition has been added to the sender. After dequeuing an event from the DES, the GBN sender will compare the current time and the time of the event, if there is a gap between the two time and the queue for the packet being send has not reach to the window size limit, the GBN sender will send a new packet to the received to increase the throughput. After sending each packet, the GBN sender will update its current time counter and check the condition again, and when the current time pass the event time or the queue is full, the GBN sender will start to process the next event. Depending on the type of the dequeued events, the GBN sender will trigger different actions. For a time-out event, the GBN sender will have to resend the packet with the sequence number in the time-out event, and at the same it will also resend all the other packets in the queue by resetting the packet in queue counter to zero and move the position counter to the first packet in the queue.

```
    while(totalpacket<1000):
        i = ES[0]
        #print(i.type)
        while(current_time<= i.time and packetInQ < 4):
            current_time = current_time+packetLength/C
            transTime = current_time
            result = sendGBN(transTime,M[packetInQ][0],packetLength,BER,tor)
            packetInQ = packetInQ +1
            if(result.type != 'NIL'):
                ES.append(result)
                ES = mergeSort(ES)
            current_time = transTime
```

```
        #print("current loop number:" + str(test))
        if(i.type == 'TimeOutEvent'):
            timeoutCounter = timeoutCounter + 1
            current_time = i.time+packetLength/C
            TimeOutEvent = current_time+timeOut
            ES = clearTimeOutEvent(ES)
            if(M[0][0] ==i.sequence_number):
                ES= []
                ES = addTimeOutEvent(ES,TimeOutEvent,i.sequence_number)
            M[0][2] = current_time
            for r in range(1,4):
                M[r][2] = M[r-1][2]+packetLength/C

            p_GBN_Receiver = i.sequence_number
            p=i.sequence_number
            result = sendGBN(current_time,i.sequence_number,packetLength,BER,tor)
            packetInQ=1
```

For an acknowledge event, the GBN sender will check if the SN in the event is in the acceptable list of excepted SN. If all the conditions were met, the GBN sender will calculate the slide size of the window, and shift the packets in the queue and refill the new space with new packets.

```
        elif (i.type == 'ACKEvent'):
            #packet send and received correctly
            acceptableRN = [(p+1)%5,(p+2)%5,(p+3)%5,(p+4)%5]
            # print('acceptableRN:' + str(acceptableRN))
            if(i.error_flag == 0 and  i.sequence_number in acceptableRN):
                slideSize = (i.sequence_number-p)%5
                ES = clearTimeOutEvent(ES)
                newQ = shiftAndFill(M,slideSize,i.time,SN,T,L)
                M = newQ[0]
                SN = newQ[1]
                T = newQ[2]
                p = M[0][0]
                packetInQ = 4-slideSize
                current_time = i.time
                TimeOutEvent = M[0][2]+timeOut
                ES = addTimeOutEvent(ES,TimeOutEvent,M[0][0])
                ES = mergeSort(ES)
```

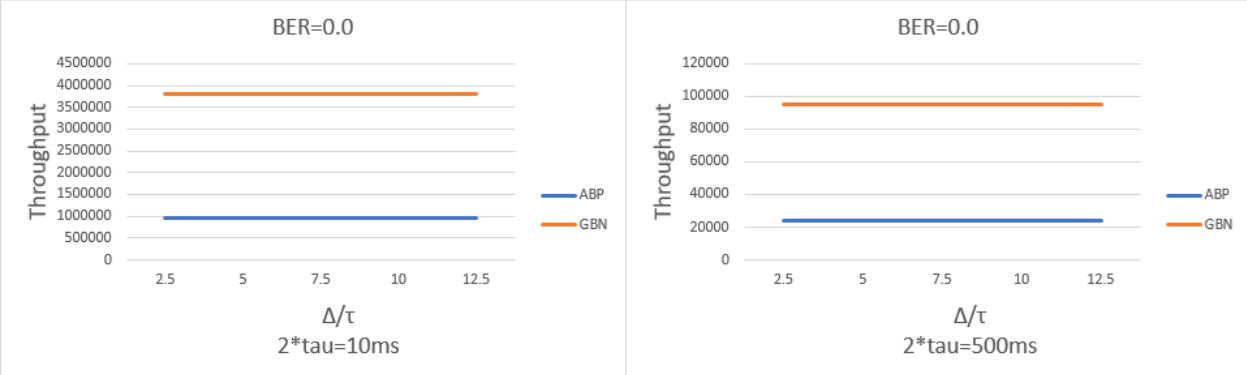
i) Take N = 4 and BER = 0. Use your simulator to compute the throughput (bits/sec) as a function of Δ for C = 5Mb/s, and two values of 2τ (10ms and 500ms).

Compare your results with that of Question 1.i by putting the results for ABP on the same graph.

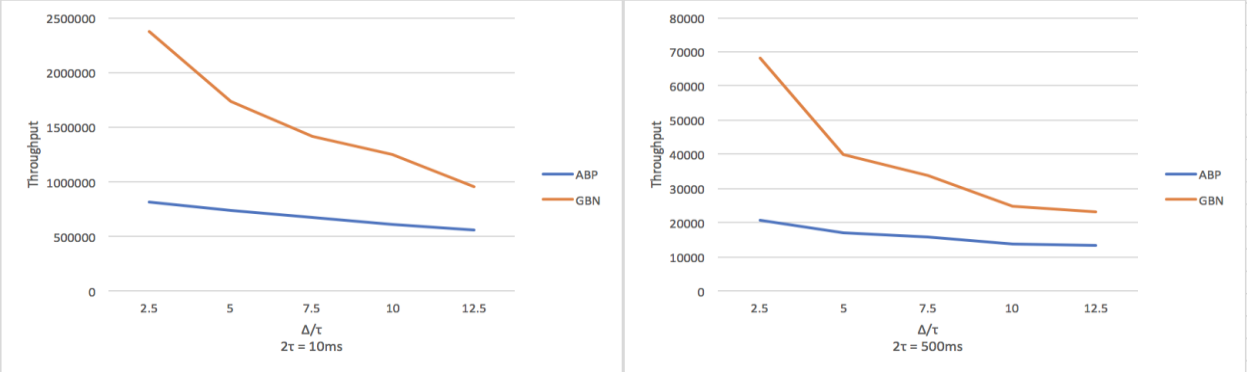
ii) Repeat Question 1.ii for GBN.

Plot your results together with that of results obtained in Question1.ii for ABP. Discuss the results.

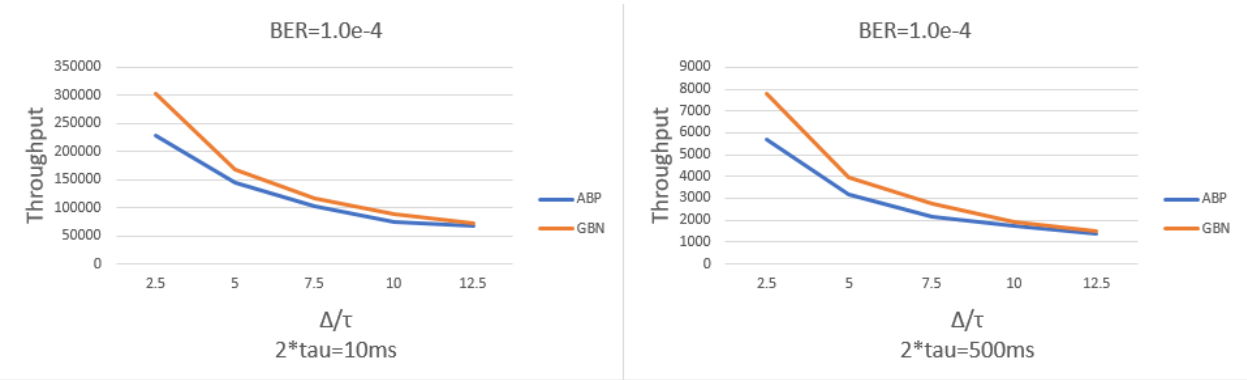
Δ/τ	2τ = 10ms			2τ = 500ms		
	BER=0.0	BER=1e-5	BER=1e-4	BER=0.0	BER=1e-5	BER=1e-4
2.5	3817765	2389200	304115.1	95508.55	69851.5	7770.771
5	3817765	1558002	168762.4	95508.55	38592.41	3935.563
7.5	3817765	1543722	117493.9	95508.55	34015.41	2739.573
10	3817765	1265138	89615.09	95508.55	30279.72	1905.263
12.5	3817765	1004803	73196.32	95508.55	20815.17	1485.8



From the graph, we can see that with the BER = 0, the throughput of the ABP sender and the GBN sender stays the consistent, and GBN's throughput is four times higher compare to ABP's throughput. This is an expected result, because with BER =0, it means that all the packets will be send without error, so there will not be any timeout event happening. Thus, GBN will be able to send a maximum of 4 packets within the time period of ABP sending one packet. We can prove this with some simple calculation. First of all, we know that the take of sending one packet is the transmission time and propagation time, and the transmission time and propagation of the acknowledgement. So, $\frac{54\times8+1500\times8}{5\times10^6}$ = 0.0024s is the transmission time of transmitting a packet to the link, and $\frac{54\times8}{5\times10^6}+2\times0.005$ =0.01s is the time of it takes to receive a response. Thus, within the 0.01s, GBN will be able to transmit 3 more packets, so resulting in the throughput 4 times higher.



From the graph, we can see that with the BER = 1.0e-5, the throughput of the ABP sender decreases as the timeout time increases, this is expected due to the errors in the packet causing timeout event. On the other hand, for GBN sender, when there is a timeout event, there is also a decrease in throughput happening. Since GBN sender does not process with the NAK strategy, the GBN sender needs to wait for a full timeout time before realizing that there is an error or packet loss that happened. Then, after realizing that there is a timeout event, it has to resend all the packets in the queue again, because the receiver did not store the packets that are send after the error packet. The probability of a packet having error stays the same. Thus, there is an 88% of chance the packet has no error, and a 12% of chance the packet need to be resend. This explains that although both ABP sender and GBN are decreasing as the timeout time increases, GBN sender still have a higher throughput overall, because 88% of the whole time, the GBN sender is sending more packets.



From the graph, we can see that with the BER = 1.0e-4, the throughput of the ABP sender decreases as the timeout time increases just like when BER = 1.0e-5, but this time it decreases more at a higher speed. On the other hand, the GBN sender is in the same situation, as timeout time increases, the throughput also decreases at a much higher speed. Both of them are affected by the increase in BER. We can also explain this with probability too. The probability of a packet having error stays the same. Thus, there is an 28% of chance the packet has no error, and a 72% of chance the packet need to be resend. This explains that although GBN is decrease as fast as ABP and almost reaching to the same throughput. GBN and ABP sender are both spending 72% of the total time processing timeouts and wasting time on the timeout time. Thus, the optimization of GBN became less significant as BER and timeout time increases.