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 3](#_Toc508741753)

[Question 1: Implement the ABP sender 3](#_Toc508741754)

[i) Consider the scenario where BER = 0. Use your simulator to compute the throughput (bits/sec) 3](#_Toc508741755)

[as a function of Δ for C = 5Mb/s, and two values of 2τ (10ms and 500ms). 3](#_Toc508741756)

[M/M/1 Queue 3](#_Toc508741757)

[Question 2: Build your simulator for this queue and explain in words what you have done. 3](#_Toc508741758)

[Step 1(Generating the list of observers): 4](#_Toc508741759)

[Step 2(Generating the list of packets): 4](#_Toc508741760)

[Step 3 (Creating event list for the DES): 5](#_Toc508741761)

[Step 4 (Initialize variables and dequeue the event list): 6](#_Toc508741762)

[Question 3: Assume L=12000 bits, C=1 Mbits/second and give the following figures using the simulator you have programmed. 6](#_Toc508741763)

[1. E[N], the average number of packets in the system as a function of (for 0.25< 0.95, step size 0.1). 6](#_Toc508741764)

[2. PIDLE, the proportion of time the system is idle as a function of ρ, (for 0.25 < 0.95, step size 0.1). 8](#_Toc508741765)

[Question 4: For the same parameters, simulate for ρ = 1.2. What do you observe? 10](#_Toc508741766)

[M/M/1/K Queue 11](#_Toc508741767)

[Question 5: Build a simulator for an M/M/1/K queue. 11](#_Toc508741768)

[Change 1 (Generating a packet list without departure time): 11](#_Toc508741769)

[Change 2(Calculating the departure time during the simulation): 12](#_Toc508741770)

[Change 3(Optimization of event handler speed): 13](#_Toc508741771)

[Question 6: Let L=12000 bits and C=1 Mbits/second. Use your simulator to obtain the following figures 13](#_Toc508741772)

[1. E[N] as a function of ρ (for 0.5 < ρ < 1.5, step size 0.1) for K=5, 10, 40 packets. 13](#_Toc508741773)

[2. PLOSS as a function of (for 0.4 < ρ <10) for K=5, 10, 40 packets. (One curve per value of K on the same figure). 15](#_Toc508741774)

# 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.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ∆/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)

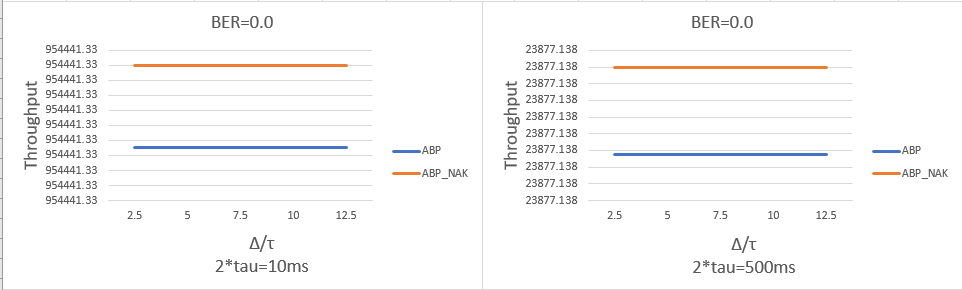
ES.remove(i)

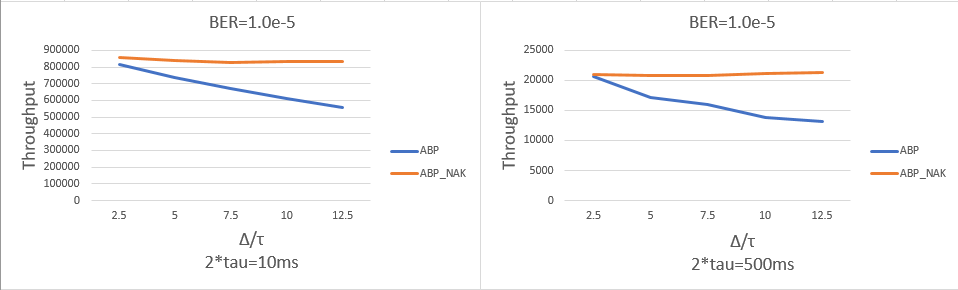
### i) Consider the scenario where BER = 0. Use your simulator to compute the throughput (bits/sec)

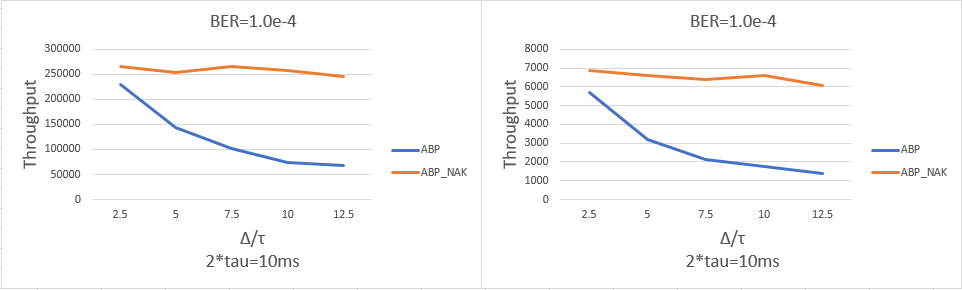
### as a function of Δ for C = 5Mb/s, and two values of 2τ (10ms and 500ms).

### ii) Now, repeat the set of experiments in (i) with BER = 1.0e-5 and BER = 1.0e-4.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ∆/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 |







# GBN

## Question 3: Implement the GBN sender

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.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ∆/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 |

