ECS 152A/EEC173A (SQ 2020): Project 1

Simulation Analysis of an IEEE 802.X Based Network

Weixiang Wang (wxiwang@ucdavis.edu)

Lingxiao Meng(lmeng@ucdavis.edu)

Abstract of discrete event simulation

Discrete event simulation is a model which marks a discrete sequence of events. At each time stamp we record the event information and the time. The state of the system doesn''t change between two consecutive events. We use some kinds of distributions to generate the event time. We use a Global Event List (GEL) to store the events. WIth this GEL, we can compute some average statistics by computing the area under the curve.

Abstract of project 1

This project simulates the queuing delay and transmission delay in Packet-Switched Networks. We have two main assumptions. First, the arrival time interval and service time follows a negative exponential distribution. Second, the buffer follows FIFO order. Our goal is to determine the average queue length, average server utilization and number of dropped packets. The queuing theory can give results theoretically. We will compare our experiment results with theoretical results.

Overall project logic

a. Data structure

We define an event class in my code. Its data members are time, type, number, queue length and total packet loss. Time is the time when the event happens. Type is a boolean value which is true when the event is an arrival event and is false when it's a departure event. Number is the number of the packet which arrives or departs. Queue length and total packet loss are the length of the queue and the total number of dropped packets when an event takes place.

Then we define a transmitter class to simulate the real-world transmitter. Its data members are current packet, max buffer and buffer. Current packet is the # of the packet which is currently transmitting. Max buffer is the maximum size of the buffer. Buffer is implemented with a queue of integers. Those integers represent packets.

We use a doubly linked list to implement Global Event List. We don't need to sort this list because we make sure the events are inserted in order. We will explain how we achieve this in the following part.

We use a queue to implement the buffer because it's a first-in-first-out container. The first packet which arrives at the buffer is the first to be transmitted.

b. Processing events

The most important thing in this part is to determine whether the next coming event is arrival or departure and its time. To achieve this, we compute and compare the time of the next arrival event and the next departure event separately. We do this iteratively.

If the last event is departure, we only need to update the time of the next departure event (because the time of the next arrival event is computed before). To obtain the time of the next departure event, we should know there are 2 possible cases:

1. The server is busy now. In this case, the time of the next departure event is the time of the last departure event plus the transmission time. The reason is the current packet is being transmitted after the last departure event happens.
2. The server is not busy now. In this case, the time of the next departure event is the time of the next arrival event plus the transmission time. Because there are no packets in the queue or server, we don't have any packets to depart so we would wait until the next packets to arrive.

If the last event is arrival, we only need to update the time of the next arrival event. In this case, to obtain the time of the next arrival event, we only need to add a random number generated by negative exponential distribution to the time of the last arrival event. It's easier than the case above because the arrival time only depends on the time of the last arrival event.

Now we have obtained the arrival time and departure time. We compare them and push the event which happens earlier. In this approach, we maintain the Global Event List in time order. When pushing the event, we also record the queue length and the total number of dropped packets when the event happens.

c. Collecting statistics

With the Global Event List, we can easily collect our statistics.

1. Mean queue length

   Mean queue length is equal to the area under the curve of queue length function divided by the total time. Because the queue length is constant between two consecutive events, the area is equal to

$$\sum_{i=1}^{n-1} queueLength[i] \times (time[i+1] - time[i])$$

   where i is the index of events and n is the number of the events.

   So the mean queue length is equal to

$$\frac{1}{time[n]} \sum_{i=1}^{n-1} queueLength[i] \times (time[i+1] - time[i])$$

2. Mean utilization

   The approach of computing mean utilization is almost the same as computing mean queue length. Here I give the formula directly. The mean utilization is given by

$$\frac{1}{time[n]} \sum_{i=1}^{n-1} sgn(queueLength[i]) \times (time[i+1] - time[i])$$

$$\operatorname{sgn}(x) := \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$

We use this sign function because the server is busy when queue length is greater than 0 and is not busy when queue length is equal to 0.

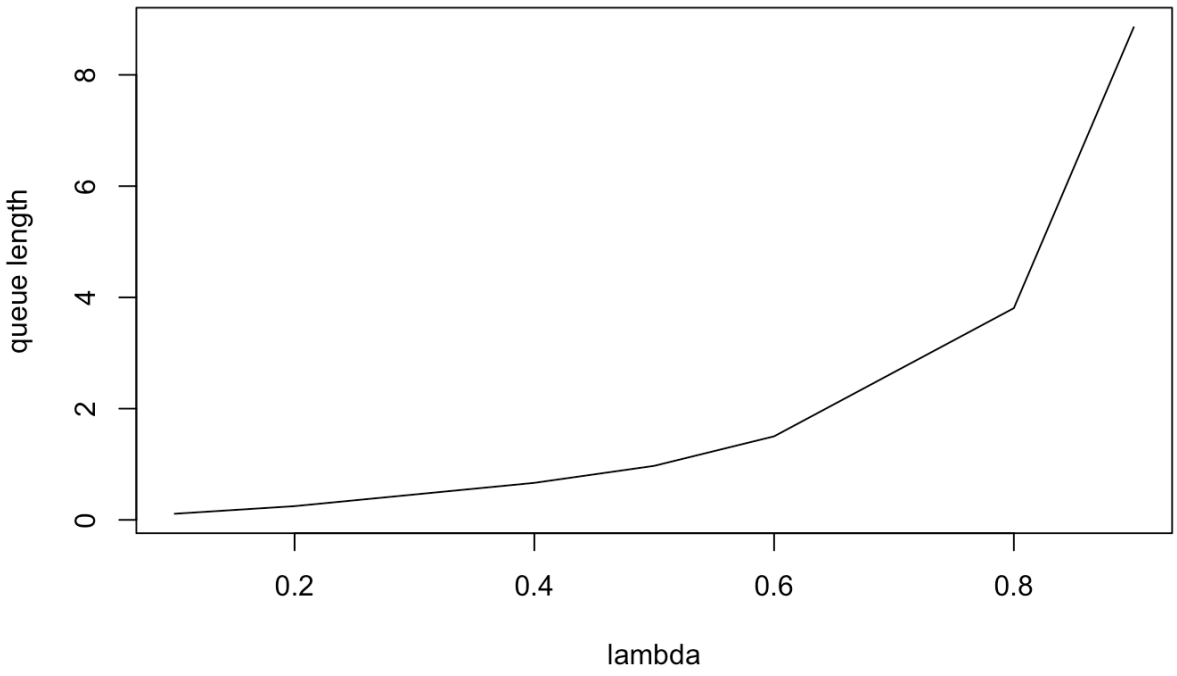
3. Total number of packets dropped

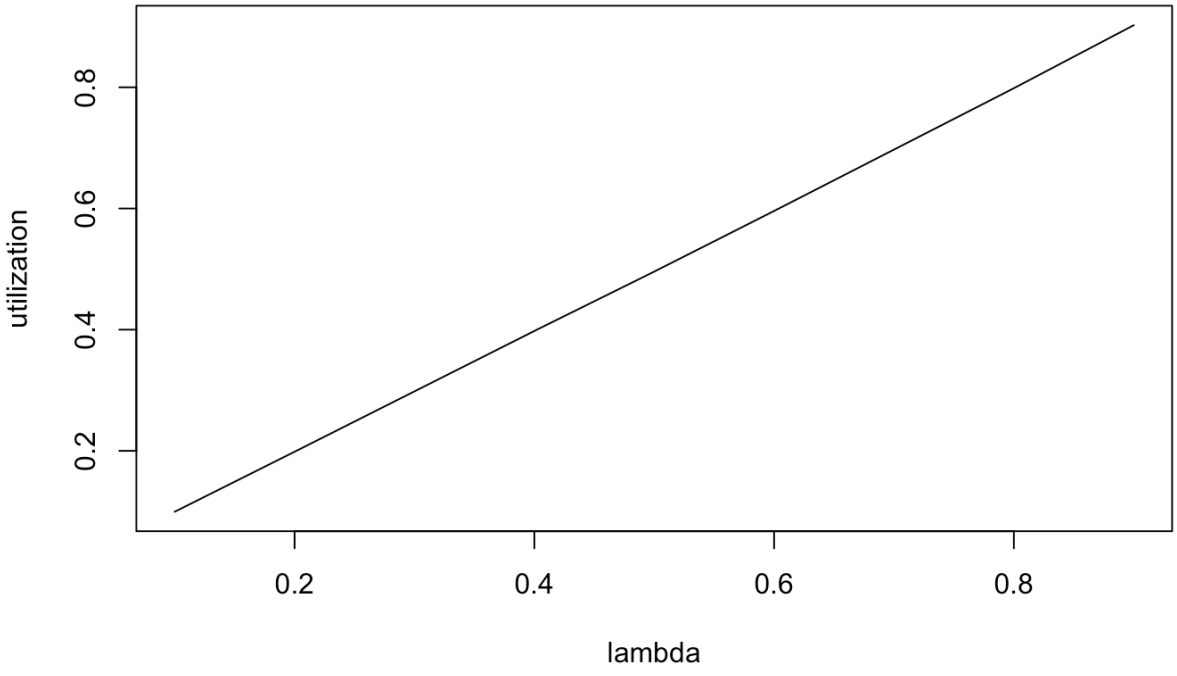
   The total number of packets dropped is equal to the number of packets dropped when the last event occurs. So it's given by

$$totalPacketLoss[n]$$

Results and Analysis

a. Mean queue length and server utilization


queue length
0
2
4
6
8
0.2
0.4
0.6
0.8
lambda


utilization
0.2
0.4
0.6
0.8
0.2
0.4
0.6
0.8
lambda

We can see that the relationship between server utilization and λ(arrival rate) is approximately linear. The relationship between queue length and λ(arrival rate) is non-linear.

b. Comparison between theoretical results and simulation results

In queuing theory, the utilization factor is given by

$$\rho = \frac{\lambda}{\mu}$$

This concept is equivalent to the mean server utilization in our simulation.

In our simulation we fix $\mu = 1$so $\rho = \lambda$. The theoretical mean server utilization is (0.1, 0.2, 0.4, 0.5, 0.6, 0.8, 0.9) for $\lambda = (0.1, 0.2, 0.4, 0.5, 0.6, 0.8, 0.9)$. Our experiment result is $(0.09945754, 0.19851380, 0.39786909, 0.49583434, 0.59594209, 0.79854173, 0.90253429)$. The relative error is less than 0.6%.

In queuing theory, the average number of customers in the system is given by

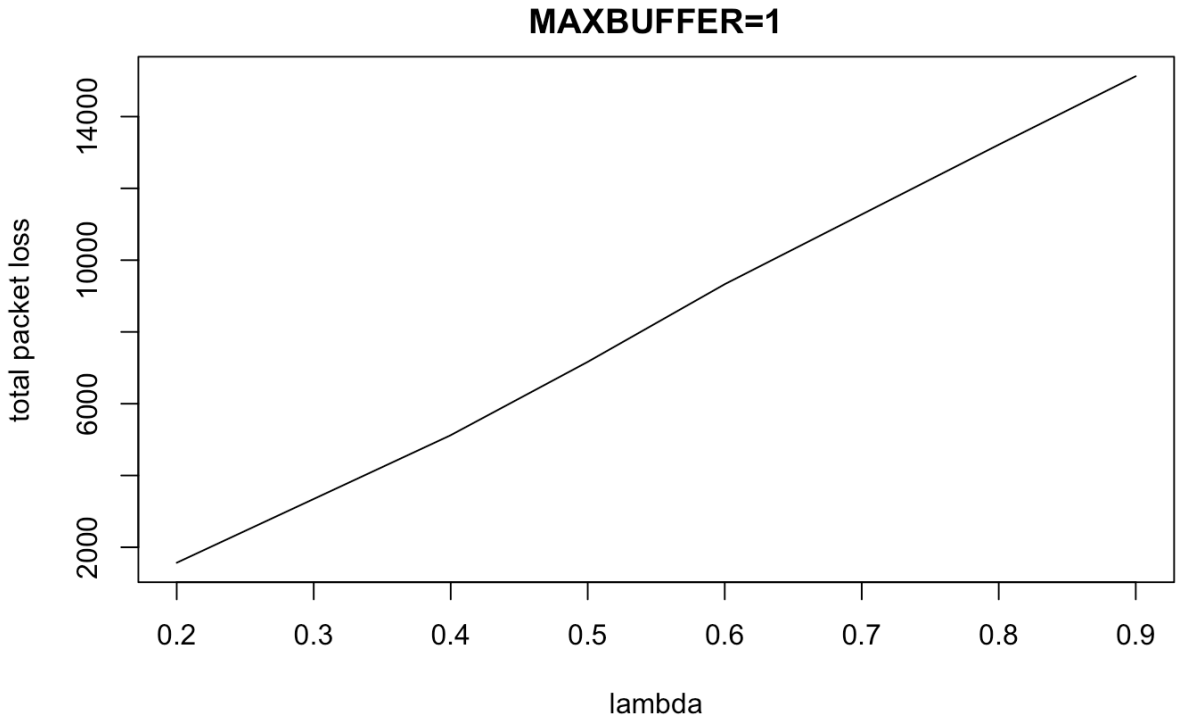
$$N = \frac{\rho}{1-\rho}$$

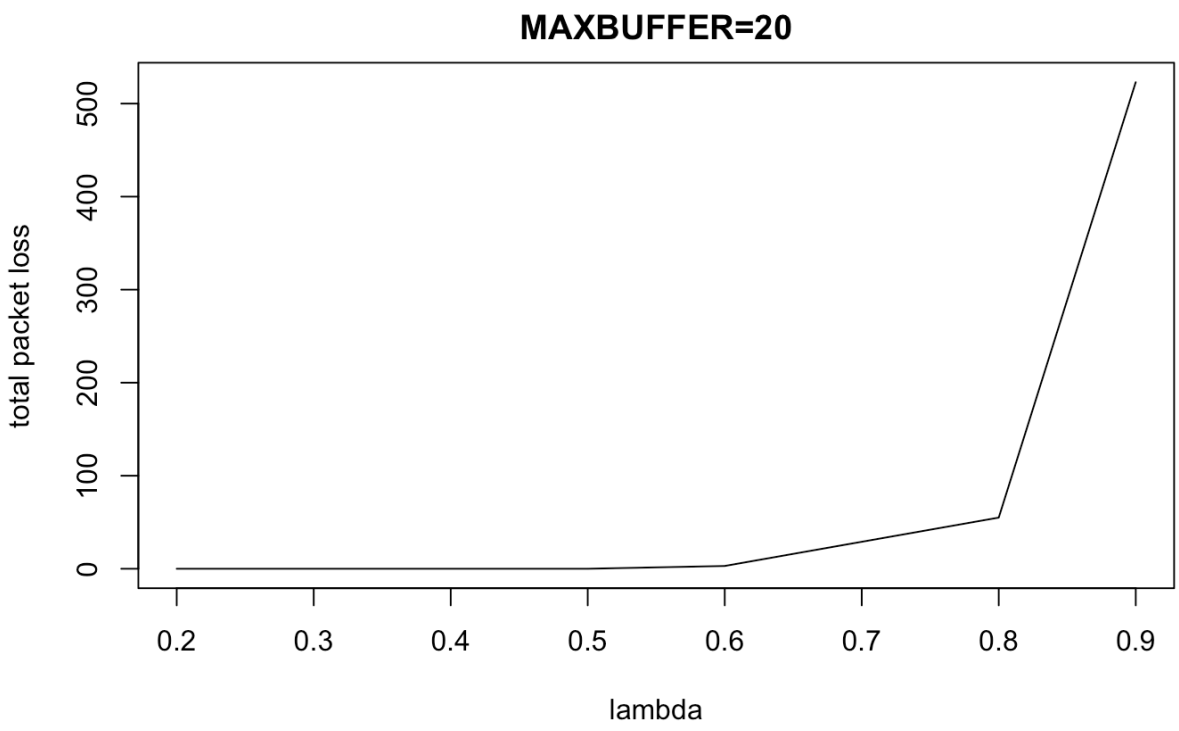
This concept is equivalent to the mean queue length in our simulation.

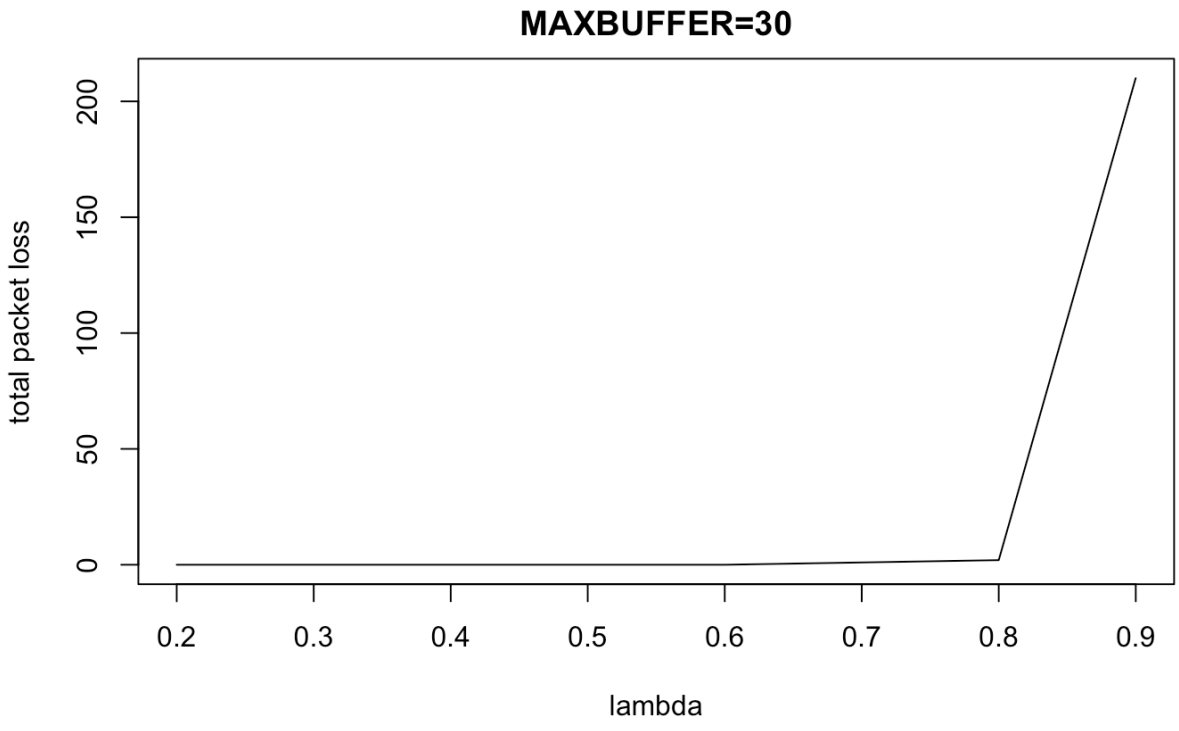
In our simulation, we fix $\mu = 1$so $N = \frac{\lambda}{1-\lambda}$. The theoretical mean queue length is $(0.1111111, 0.2500000, 0.6666667, 1.0000000, 1.5000000, 4.0000000, 9.0000000)$ for $\lambda = (0.1, 0.2, 0.4, 0.5, 0.6, 0.8, 0.9)$. Our experiment result is $(0.1103637, 0.2472941, 0.6658051, 0.9716391, 1.5022231, 3.8070248, 8.8568656)$. The relative error is less than 2.5%.

Therefore, our experiment results are very close to theoretical results.

c. Total number of dropped packets



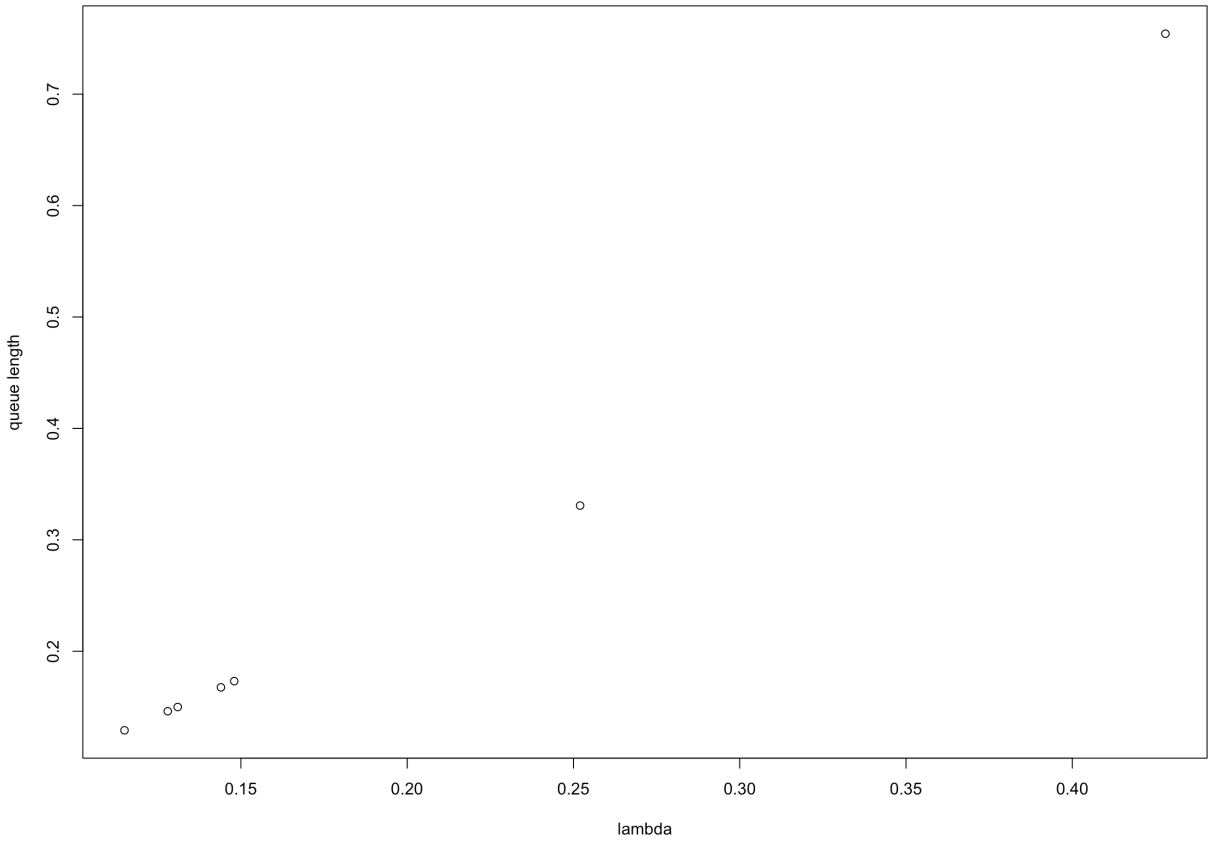

MAXBUFFER=20
total packet loss
0
100
200
300
400
500
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
lambda

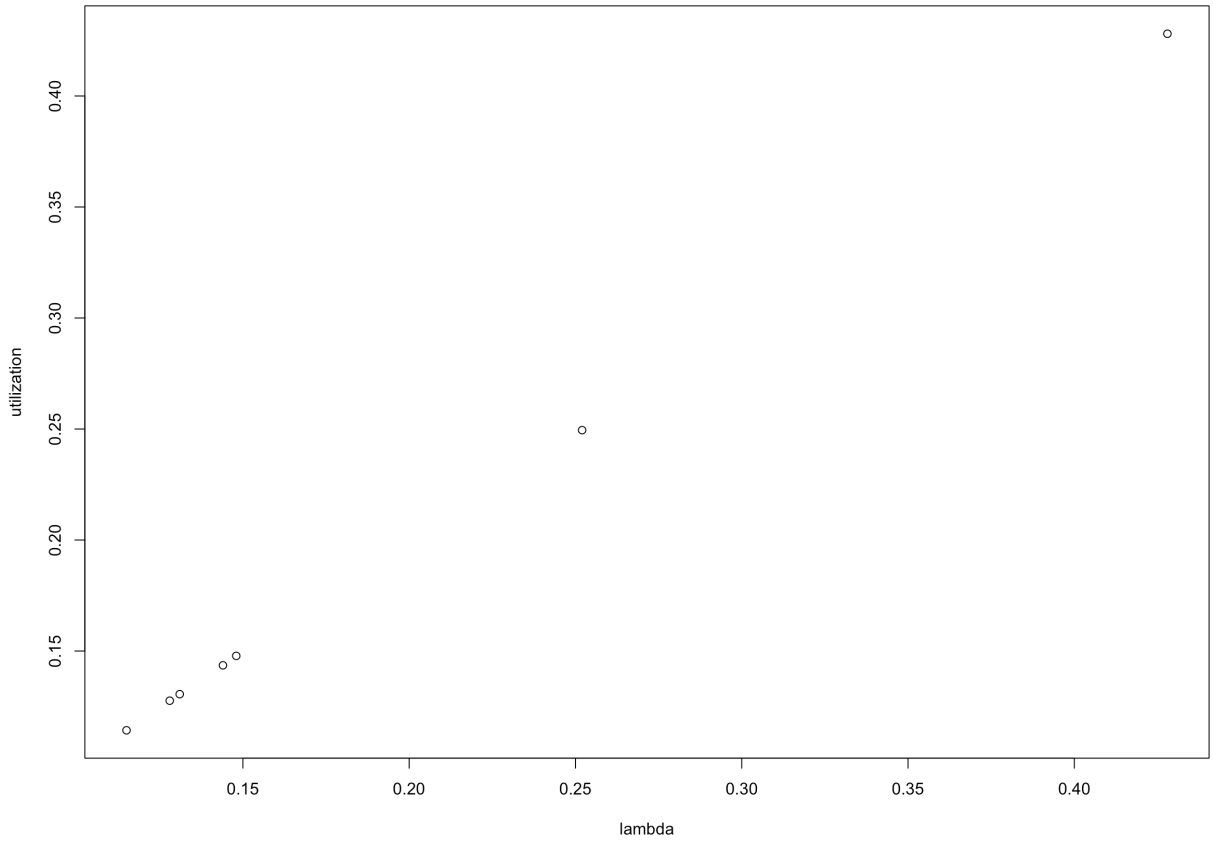

MAXBUFFER=30
total packet loss
0
50
100
150
200
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
lambda

In these plots we can see that when the buffer size is equal to 1, the relationship of the packet loss and the arrival rate is approximately linear. When the buffer size is 20, the packet loss is 0 when the arrival rate is less than or equal to 0.5. When the buffer size is 30, the packet loss is 0 when the arrival rate is less than or equal to 0.6.

d. Extra credit

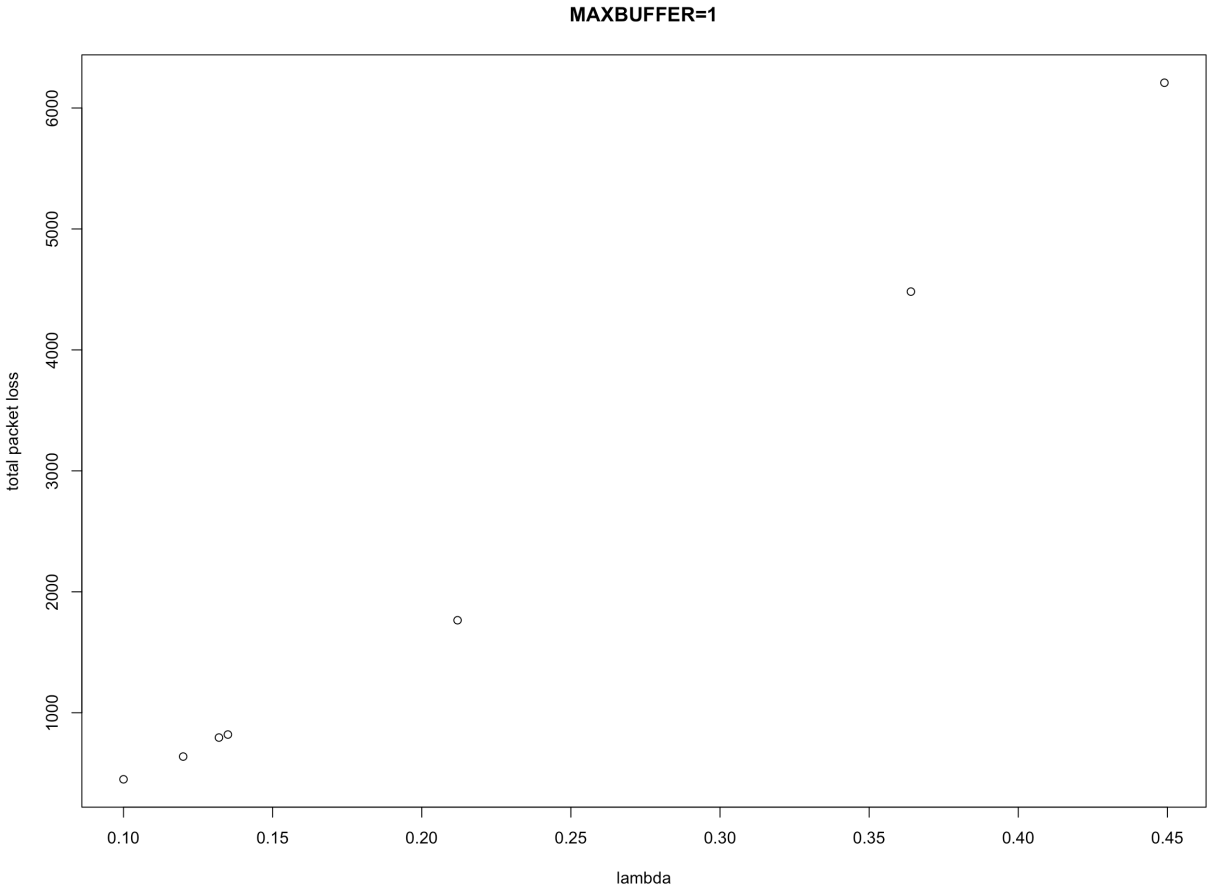
In this part, we generate λ from a Pareto distribution with parameter $x_m = 0.1 \; and \; \alpha = 1.3$. The arrival time and the service time still follows negative exponential distributions with parameter λ and μ. Because Pareto distribution has a low tail probability, most points fall in interval $[0.1,0.2]$. But we can still compare our experiment results with theoretical results between the relationship between mean queue length/server utilization and the arrival rate doesn't change. To make our plots clear, we draw scatter plots in this part.

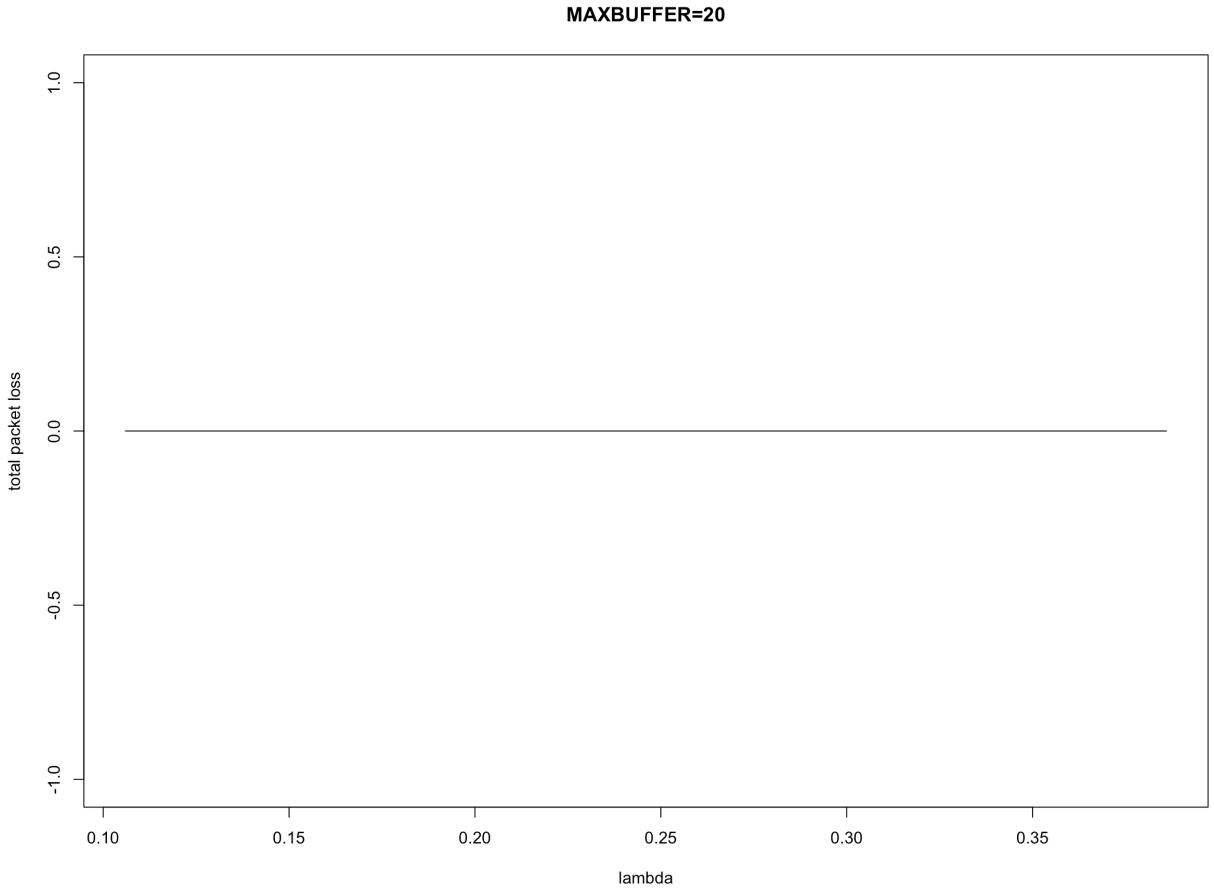

queue length
0.2
0.3
0.4
0.5
0.6
0.7
0.15
0.20
0.25
0.30
0.35
0.40
lambda

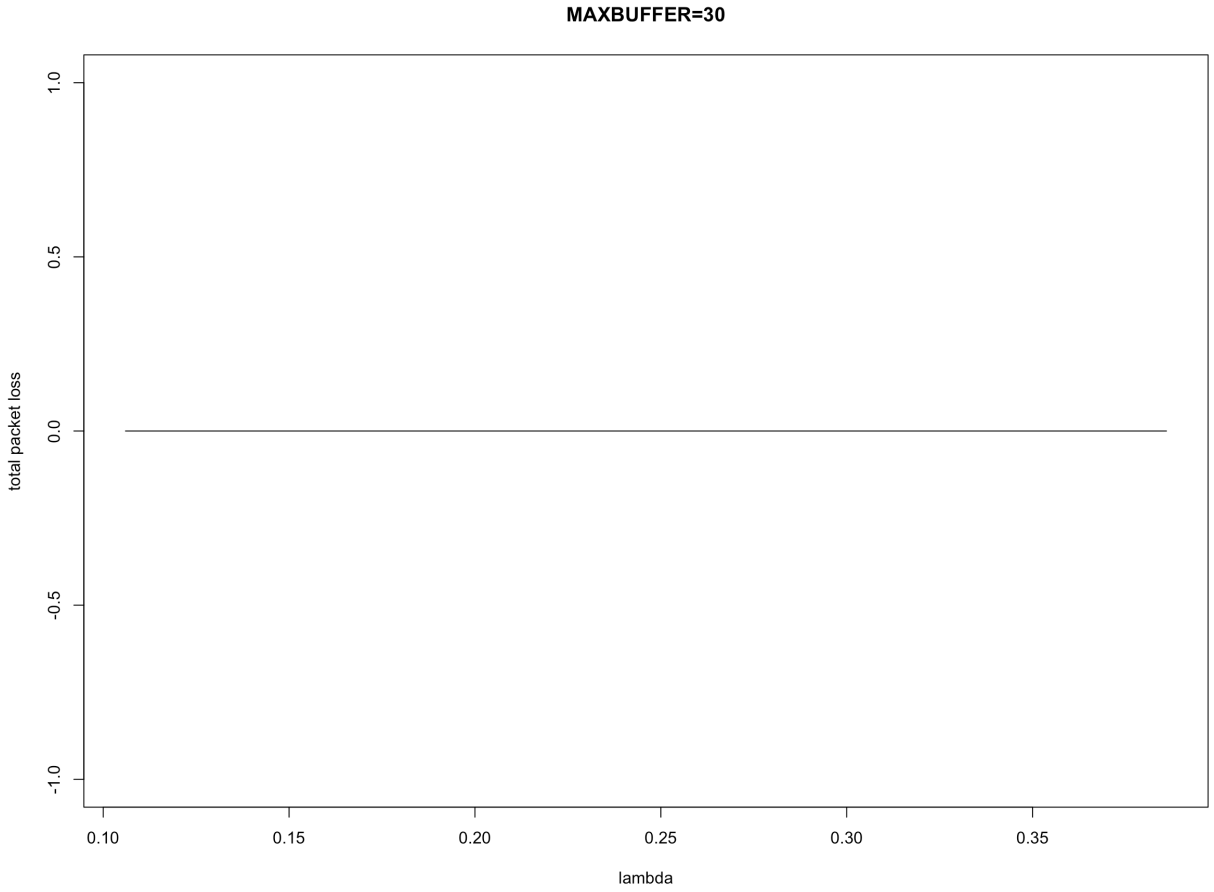


The theoretical values for mean queue length are $(0.1299435, 0.1467890, 0.1507480, 0.1682243, 0.1737089, 0.3368984, 0.7482517)$ while our results are $(0.1290424, 0.1461285, 0.1499424, 0.1675891, 0.1731574, 0.3307207, 0.7542143)$. The relative error is less than 0.94%.

The theoretical values for server utilization are $(0.115, 0.128, 0.131, 0.144, 0.148, 0.252, 0.428)$ while our results are $(0.1142953, 0.1276285, 0.1305723, 0.1435494, 0.1478138, 0.2494856, 0.4280420)$. The relative error is less than 0.53%.


MAXBUFFER=1
total packet loss
1000
2000
3000
4000
5000
6000
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
lambda


MAXBUFFER=20
total packet loss
1.0
0.5
0.0
-0.5
-1.0
0.10
0.15
0.20
0.25
0.30
0.35
lambda



When the max buffer is equal to 20 or 30, there's no dropped packets because the arrival rate is not great enough.

Project Insights and individual contributions

Weixiang Wang(wxiwang@ucdavis.edu)

In this project, I am responsible for the coding part and the extra credit part. The most challenging aspect of my implementation is to maintain the Global Event List. The events in the GEL should be in order. The departure events rely on the arrival events. Finally, I thought of an algorithm that computes the time of the next arrival event and the next departure event separately. Then insert the event that happens first. This algorithm can both maintain the order

of GEL well and match the simulation logic. One of the advantages of my implementation is flexibility. I have an event class and a server class. The algorithm can be put in a function. I can change my code easily if I need to do some other simulations. For example, if we need to simulate the link with multiple servers, I just need to create an array of servers and run the algorithm for each server iteratively.

Lingxiao Meng (lmeng@ucdavis.edu)

In this project, I am working on collecting statistics, analyzing the results and the mathematical part. In project 1, I talk with my partner with the GEL model and understand the query theory. I use R to read data collected from our simulation program and then compute the statistics and draw the plot. Then I compare the theoretical results and simulation results. I also work with my partner to debug the code.