ECS152A Project 1: Discrete event simulation of Queue

Yuliang Dong (yuldong@ucdavis.edu)
Jui-yang Cheng (zrycheng@ucdavis.edu)
Sampson Ezieme (siezieme@ucdavis.edu)

Description of discrete event simulation

Discrete event simulation is a method to simulate systems like a queue. Unlike synchronous simulation, discrete event simulation is based on the occurrence time of events instead of real time. The simulation time jump from one event to next event.

Description of Project 1

This project is an implementation of discrete event simulation of a switch queue. We are able to simulate different situation by specifying different arrival rate of packets and service rate of packets.

Overall Project Logic

Data Structure

- Event: Event is a data structure to store an event. It includes timestamp and service time.
- Global Event List (GEL) : Global Event list is a list to store every events generated in the simulation process. The data structure class is a subclass of LinkedList provided by Java.util package. Upon insertion, the newly added event will be inserted in a position to maintain a ascending order.
- Packet Queue: Packet Queue is a queue to store each packet waiting for processing. The data structure class is a subclass of LinkedList provided by Java.util package. Besides the basic function of a queue, we add instance variable to record max queue size and implemente an 'add' method to drop packet if the queue has max queue size elements.

Function

- Randdom Generator: This function is to generate a random variable following an negative exponential distribution with a given rate.

Overall logic

The overall logic for the simulation is from assignment specification.

- Initialization: In initialization stage, global event list, time, and packet queue will be initialzed. Also, the first arrival event will be generated with specified mu and lambda value. Moreover, variables to collect statistics are declared and initialized as well.
- Event processing for Arrival event: This is the stage to process events. When an arrival event is processed, new arrival event will be put into the global event list. Then it will be processed if there is no packet processing. A departure event will be put into the global event list. If the processor is busy, then the packet will be inserted to a queue.
- Event processing for Departure event: Departure event will be processed in this stage. If there are packets in queue, the first packet in the queue will be extracted and be put into global event list.

Statistics Collection

- dropCount: This variable will be incremented by 1 when the queue size reaches the max buffer length.
- Utilization rate and BusyTime: BusyTime variable will be increament by service time of an arrival event when the arrival event is being processed. Eventually, utilization rate will be BusyTime divided by Time.
- Mean Queue length, queueArea and lastQueueChangeTime: The calculation of mean queue length is through calculating the area under the curve. When the queue change, we will calculate the area by (current time - lastQueueChangeTime) * length.

Test Approach

we did both unit test and system test in this project.

- Unit test: we wrote three tests to check if all three data structures are working correctly.
- System test: we compare the output statistic with theoretical value to see if our simulation is working correctly.

Result & Analysis:

To test the correctness of our program, we followed the instruction and test the program with following data sets:

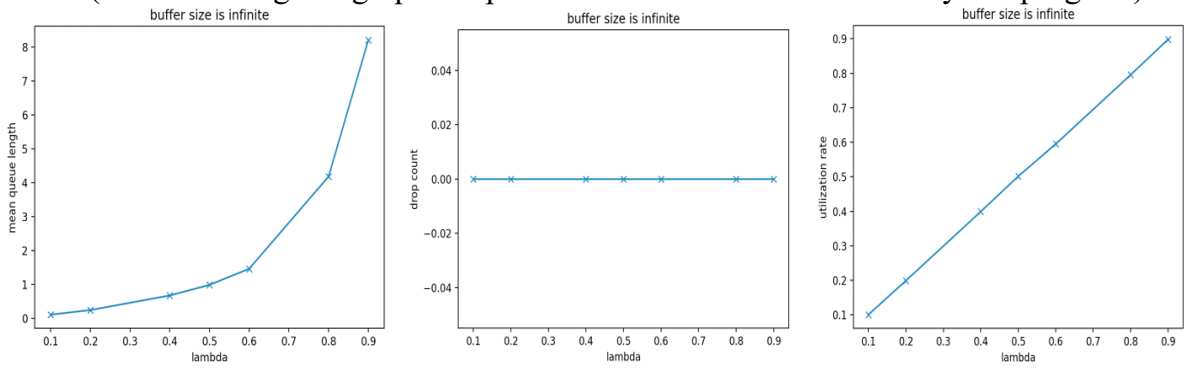
1)
Assume that μ = 1 packet/second. Plot the queue-length and the server utilization as a function of λ for λ = 0.1, 0.2, 0.4, 0.5, 0.6, 0.80, 0.90 packets/second when the buffer size is infinite.

After inputting data into our program, we received the following results:

lambda	u	maxbuffer		dropcount	mean queue	utilization rate
0.1	1	-1		0	0.11142	0.100774
0.2	1	-1		0	0.245315	0.199309
0.4	1	-1		0	0.67566	0.399479
0.5	1	-1		0	0.98804	0.501785
0.6	1	-1		0	1.463251	0.5952
0.8	1	-1		0	4.18326	0.795634
0.9	1	-1		0	8.207232	0.898451

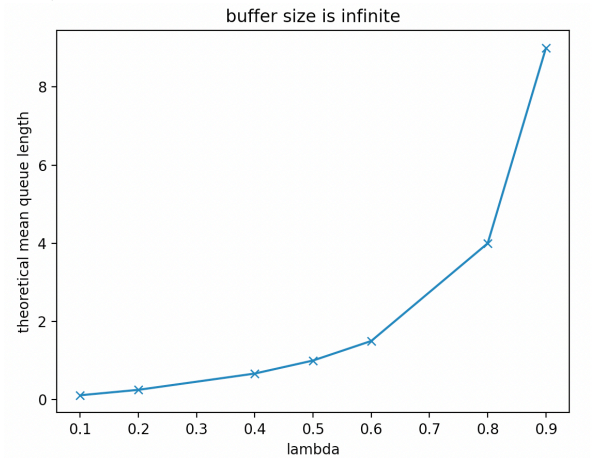
As you can see, with the buffer size set to infinite (we represent infinity with -1 in our program) and a fixed μ = 1 packet/second, the number of packets dropped is constantly zero as expected, and both mean queue length and utilization rate increases as λ increases.

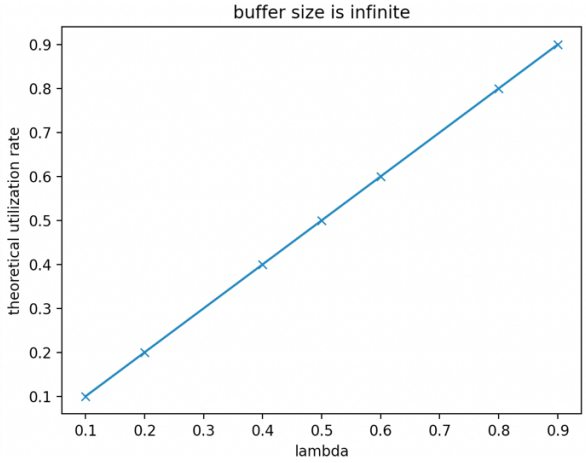
(The following is a graphic representation of the data returned by our program)



In order to further test the correctness of the results return by our program, we also compare it with the theoretical value:
We first compute the Utilization factor $p = \frac{\lambda}{\mu}$ and gets values with is essentially identical to the values of λ, and we also get the average utilization with the formula $p = \frac{\lambda}{\mu}$

(The follow is the graphic result for theoretical values of mean queue length and utilization rate.)





As the above pictures show, our results are the same as the theoretical values, which proven the correctness of our program.

2)

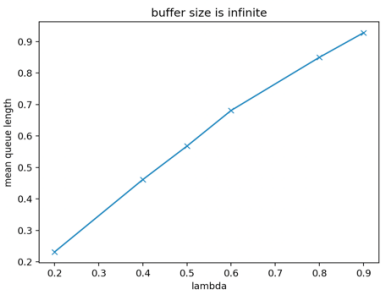
Assume that μ = 1 packet/second. Plot the total number of dropped packets as a function of λ for λ = 0.2, 0.4, 0.5, 0.6, 0.8, 0.9 packets/second for MAXBUFFER = 1.

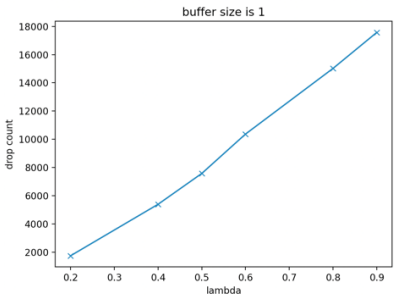
After inputting data into our program, we received the following results:

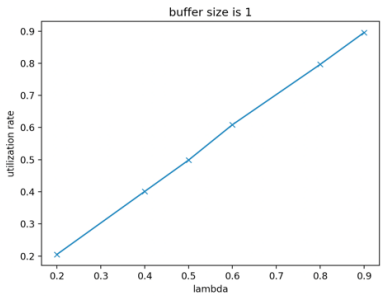
lambda	u	maxbuffer		dropcount	mean queue	utilization rate
0.2	1	1		1737	0.231674	0.204953
0.4	1	1		5387	0.461571	0.400925
0.5	1	1		7575	0.567896	0.49853
0.6	1	1		10369	0.681055	0.608637
0.8	1	1		15024	0.84986	0.796405
0.9	1	1		17575	0.927791	0.895581

As you can see, with the buffer size set to 1 and a fixed μ = 1 packet/second, the number of packets dropped, mean queue length and utilization rate increases as λ increases.

(The following is a graphic representation of the data returned by our program)



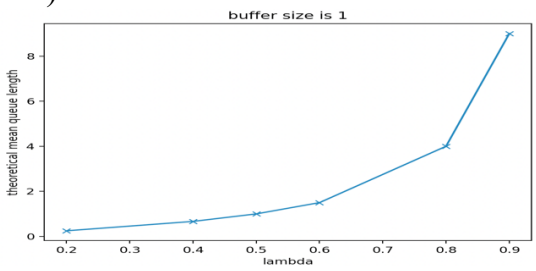


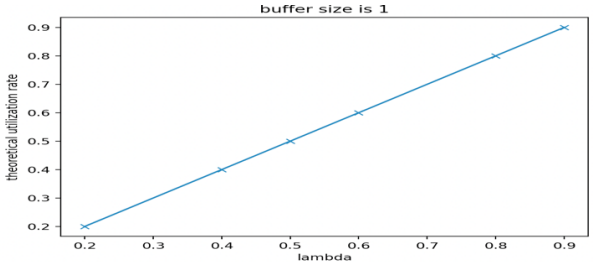


In order to further test the correctness of the results return by our program, we also compare it with the theoretical value:

We first compute the Utilization factor $p = \frac{\lambda}{\mu}$ and gets values with is essentially identical to the values of λ, and we also get the average utilization with the formula $p = \frac{\lambda}{\mu}$

(The follow is the graphic result for theoretical values of mean queue length and utilization rate.)





As the above pictures show, our result of utilization is the same as the theoretical value. However, also as above pictures show, the theoretical mean queue length is different from our result. This is because the theoretical value for mean queue length does not take the maximum buffer size into consideration, so the theoretical mean queue length is not realistic and is only for reference.

3)

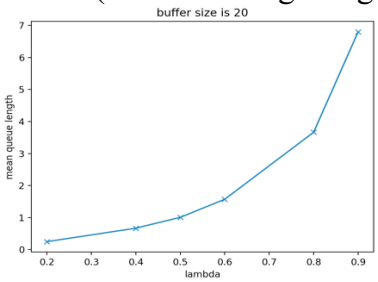
Assume that μ = 1 packet/second. Plot the total number of dropped packets as a function of λ for λ = 0.2, 0.4, 0.5, 0.6, 0.8, 0.9 packets/second for MAXBUFFER = 20.

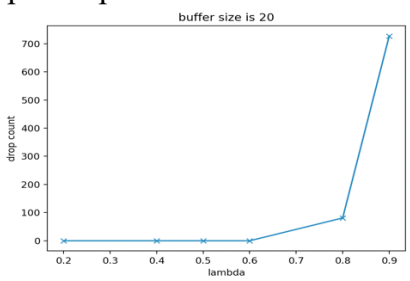
After inputting data into our program, we received the following results:

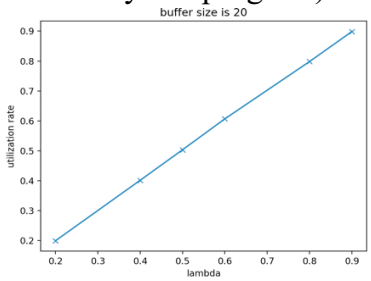
lambda	u	maxbuffer		dropcount	mean queue	utilization rate
0.2	1	20		0	0.24822	0.199073
0.4	1	20		0	0.668407	0.40126
0.5	1	20		0	1.00688	0.503206
0.6	1	20		0	1.57174	0.606748
0.8	1	20		81	3.663989	0.798178
0.9	1	20		727	6.800433	0.898606

As you can see, with the buffer size set to 1 and a fixed μ = 1 packet/second, the number of packets dropped, mean queue length and utilization rate increases as λ increases as always.

(The following is a graphic representation of the data returned by our program)



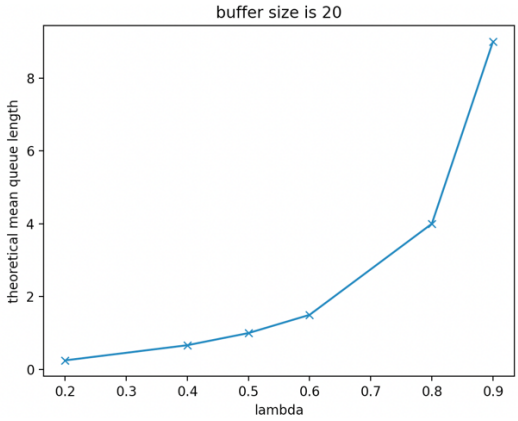


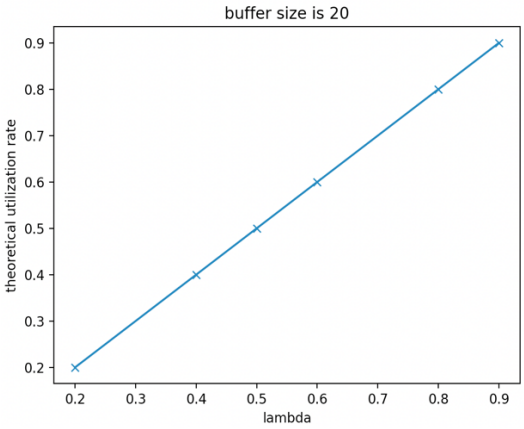


In order to further test the correctness of the results return by our program, we also compare it with the theoretical value:

We first compute the Utilization factor $p = \frac{\lambda}{\mu}$ and gets values with is essentially identical to the values of λ, and we also get the average utilization with the formula $p = \frac{\lambda}{\mu}$

(The follow is the graphic result for theoretical values of mean queue length and utilization rate.)





As the above pictures show, our results are the same as the theoretical values, which proven the correctness of our program.

4)

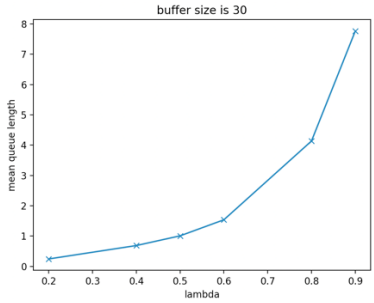
Assume that μ = 1 packet/second. Plot the total number of dropped packets as a function of λ for λ = 0.2, 0.4, 0.5, 0.6, 0.8, 0.9 packets/second for MAXBUFFER = 30.

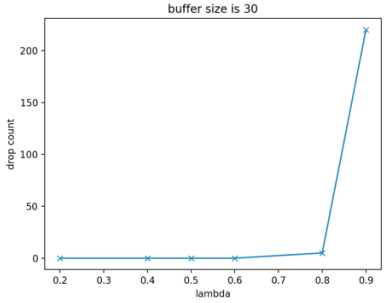
After inputting data into our program, we received the following results:

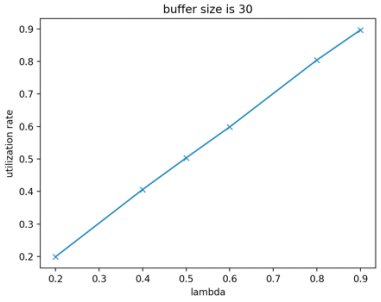
lambda	u	maxbuffer		dropcount	mean queue	utilization rate
0.2	1	30		0	0.24536	0.199022
0.4	1	30		0	0.68784	0.406016
0.5	1	30		0	1.00826	0.503396
0.6	1	30		0	1.538089	0.598665
0.8	1	30		5	4.137769	0.804323
0.9	1	30		220	7.768577	0.896989

As you can see, with the buffer size set to 1 and a fixed μ = 1 packet/second, the number of packets dropped, mean queue length and utilization rate increases as λ increases as always.

(The following is a graphic representation of the data returned by our program)

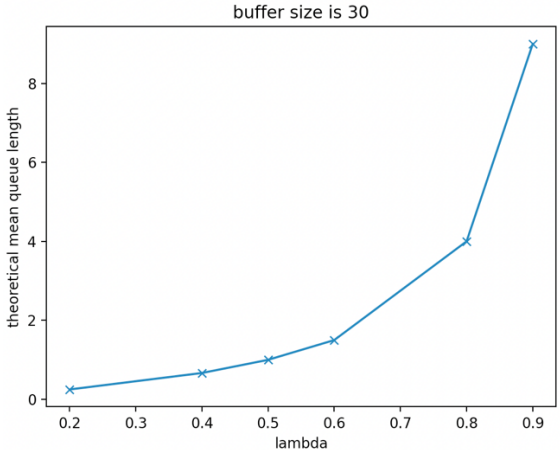


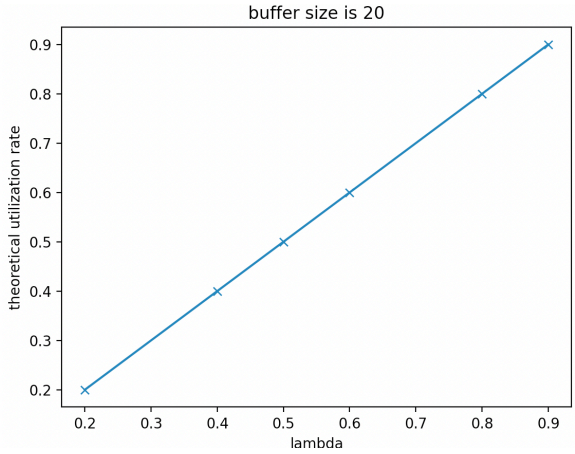




In order to further test the correctness of the results return by our program, we also compare it with the theoretical value:

We first compute the Utilization factor $p = \frac{\lambda}{\mu}$ and gets values with is essentially identical to the values of λ, and we also get the average utilization with the formula $p = \frac{\lambda}{\mu}$





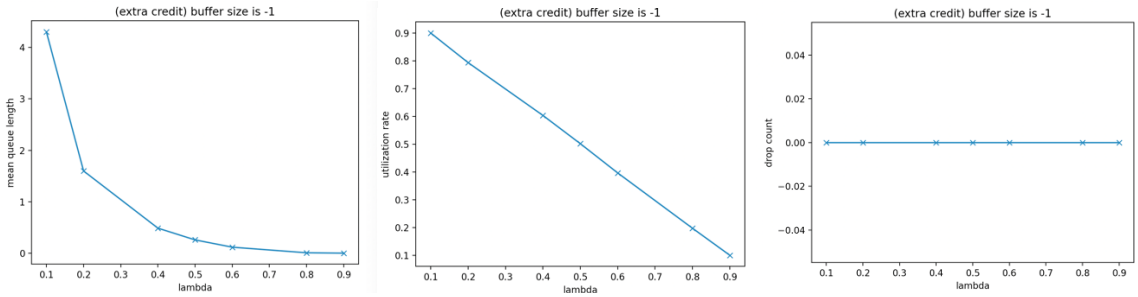
As the above pictures show, our results are the same as the theoretical values, which proven the correctness of our program.

Conclusion:

By comparing the data returned by our program and the theoretical results, our program shows a satisfying result. The only inconsistency which happens when max buffer size is 1 can also be explain with the fact that the formula does not take buffer size limitation into consideration. Therefore, our program shows a promising result overall.

Extra credit:
We let arrival rate follows the Pareto distribution and let service rate still follows negative exponential distribution as before. More specifically, we set both "scale" and "shape" parameters of Pareto distribution to 1 and get the following results.

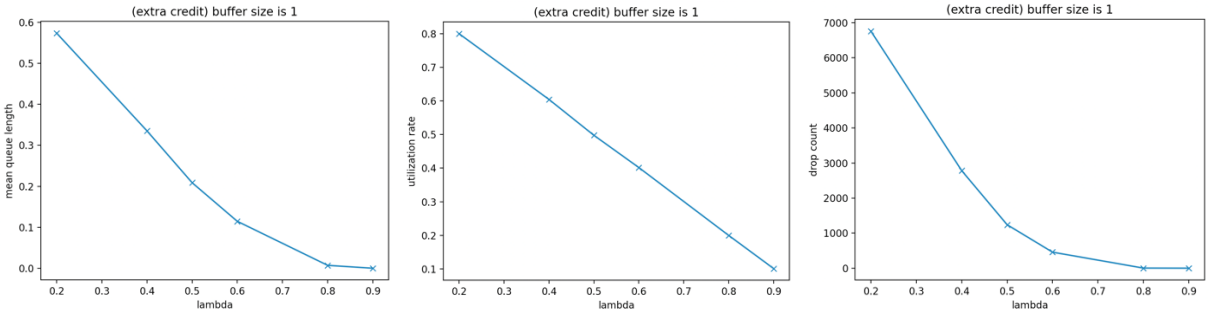
1)
Assume that μ = 1 packet/second. Plot the queue-length and the server utilization as a function of λ for λ = 0.1, 0.2, 0.4, 0.5, 0.6, 0.80, 0.90 packets/second when the buffer size is infinite.



(The left most chart is the chart for mean queue length using the above setup, the middle chart is the chart for utilization rate using above setup, and the right most chart is the chart for drop count using above setup.)

As the chart shows, the drop count is always zero as expected because we have infinite buffer so no packets should be dropped, and both mean queue length and utilization rate has a negative gradient, which are opposite to the trench when using negative exponential distribution for arrival rate.

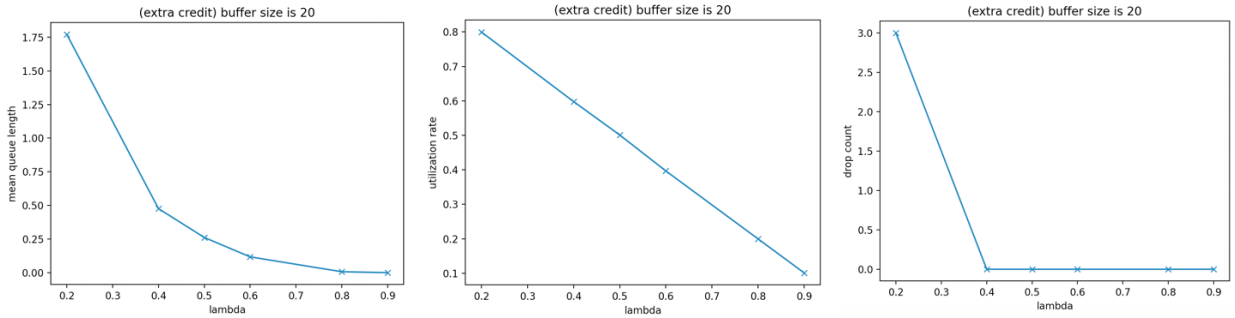
2)
Assume that μ = 1 packet/second. Plot the total number of dropped packets as a function of λ for λ = 0.2, 0.4, 0.5, 0.6, 0.8, 0.9 packets/second for MAXBUFFER = 1.



Since the maximum buffer size is now set to 1, there are now packets being dropped. As the charts shows, the gradient for mean queue length, utilization and drop count are not all negative and are still opposite to the trend when using negative exponential distribution for arrival rate.

3)

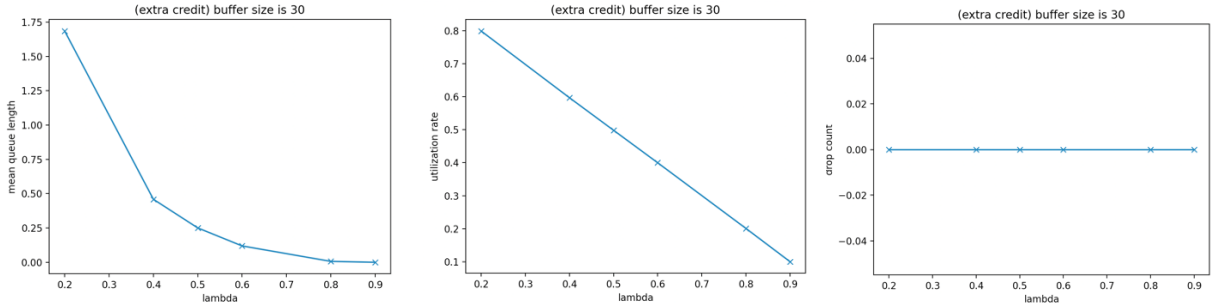
Assume that μ = 1 packet/second. Plot the total number of dropped packets as a function of λ for λ = 0.2, 0.4, 0.5, 0.6, 0.8, 0.9 packets/second for MAXBUFFER = 20.



The overall trend of mean queue length and utilization rate did not change much. However, the drop count reaches and stays at 0, probably because of the pareto distribution has reached the flat area.

4)

Assume that μ = 1 packet/second. Plot the total number of dropped packets as a function of λ for λ = 0.2, 0.4, 0.5, 0.6, 0.8, 0.9 packets/second for MAXBUFFER = 30.



The overall trend of mean queue length and utilization rate did not change much. However, the drop stays at 0, probably because that the processing speed is much faster than arrival speed and the buffer never reaches its maximum size.

Conclusion:

Overall, using Pareto distribution for arrival rate and negative exponential distribution for service rate shows an opposite trending for mean queue length and utilization rate. As for drop count, it also shows the opposite trending when we set max buffer size to 1 and 10, and probably because of Pareto distribution's property, the drop count is constantly zero when we set max buffer to 30.

Project Insights and Individual contributions

Yuliang Dong(yuldong@ucdavis.edu)

I have done the most coding implementation including data structures, overall logic and tests. In addition, I have done report except analysis part. Also, I have done shell script and python data processing.

In my opinion, the most challenging and interesting aspect of my implementation is learning discrete event simulation. Discrete event simulation is quite counter-intuitive to me who have lived in a world with linear time. However, I think it is a powerful tool for me to simulate the real world with higher efficiency.

Jui-yang Cheng(zrycheng@ucdavis.edu)

Main tasks: Code reviewing, debugging, python graph code, plot and analysis for experiment, extra credit Pareto distribution random sample generator.

Debugging is always the part that take most effort since we did not fully understand the logic at the beginning. In terms of coding, I was initially trying to plot a 3D graph using python but was unsuccessful since I was new to 3D plotting, so I eventually decide to give up the 3D graphing and turns to normal 2D charts. Also, it's pretty difficult to determine the scale and shape of Pareto distribution since some choices will cause the result to be too small that the program simply return 0. It requires some research to determine the best value for these two variables.

Sampson Ezieme (siezieme@ucdavis.edu)