*Benjamin Rayner & Eduardo Sanchez*

*ECE 358: Lab one*

M/M/1 and M/M/1/K Queue Simulation

TABLE OF CONTENTS

[Question 1: 1](#_Toc116076181)

[Question 2: 2](#_Toc116076182)

[Question 3: 4](#_Toc116076183)

[Question 4: 5](#_Toc116076184)

[Question 5: 5](#_Toc116076185)

[Question 6: 6](#_Toc116076186)

# Question 1:

The expected mean and variance of an exponential function is 1/λ (0.013̅) and 1/λ2 (0.00017̅̅) respectively when λ = 75 (⍴ = 0.15). The sampled mean and variance of the 1000 random variables we generated were 0.0133969 and 0.000172881 which agrees with the theoretic values.  
Text

Description automatically generated

# Question 2:

For our M/M/1 queue, all constant control variables are defined in Simulator.h and can be changed depending on the question. The constant control variables are as follows:

sim\_time (T): The simulated time in seconds that arrivals and departures will be generated for.  
avg\_size (L): The average bits per packet of the arrivals.  
net\_speed (C): Transmission rate of the network in bits per second.  
intensity (⍴): Utilization of the queue in arrival packets per serviced packets.  
avg\_service (C/L): Average number of departure packets in packets per second.  
avg\_arrive (λ): Average number of arrival packets in packets per second.  
test: Determines if Q1 is ran.  
inf: Activates infinite buffer case.  
fin: Activates finite buffer case.

The local variables that are defined in Simulator.cpp are as follows:

event\_queue: Stores all the events in the DES in the order of ascending time.  
time: Tracks absolute time when adding the pre calculated events.  
num\_arrive (Na): Number of packet arrivals.  
num\_depart (Nd): Number of packet departures.  
num\_observe (No): Number of observations.  
avg\_queue (E[N]): Average number of packets in queue.  
avg\_idle (PIDLE): Proportion of time no packets are in the system.  
Event class (Event.h and Event.cpp): Stores type of event and time that event occurs.

Our simulator works by having an event queue that stores generated events and then processing those events by looping through the queue in order. This implements an efficient simulator that does not have to simulate time that no events occur and instead simulates on an event-by-event basis.

For generation of events, we start off with arrival events. There’s a for loop that is constrained by the sim time to prevent arrivals outside the allocated time. On every loop, we generate a random inter-arrival time and add that offset to the absolute time. The arrival event is then added to the queue at that time as a new Event object. We are then able to precompute departure events based on the number of arrival events since we will have no packet loss with an infinite buffer.

Text

Description automatically generated

The departure event generation instead uses a random inter-departure time and a condition for what the absolute departure time is. If the current time is after the arrival time, we know that the buffer has been non-empty and thus the next departure is simply the service time (immediate). Else, we must wait for the next arrival and the departure time will be wait time plus service time.

Text

Description automatically generated

Observer events are generated the same as arrivals except at 5 times the arrival rate to accurately capture the state of the system. The queue is then sorted in ascending time.

A screenshot of a computer

Description automatically generated with medium confidence

For processing of events, the sorted queue loops through until no events remain. Local variables are used to keep track of the state of the system. On arrivals, departures, and observers, increment their respective counters. Specifically on an observer event, we know queue is empty when the number of arrivals equals the number of departures, so we increment the idle counter. Else, we accumulate the total number of packets in the buffer. At the end of the simulation, we divide our performance metrics E[N] and PIDLE by the number of observers to get an average.

Text

Description automatically generated

# Question 3:

The following results are consistent with simulation time changes:  
Graphical user interface, application, table, Excel

Description automatically generated

1. The graph below shows the exponential relationship between traffic intensity and the number of packets in the queue. The system is struggling to service incoming packets at only 1MB/s.
2. The graph below shows the linear relationship between traffic intensity and the proportion of time the buffer is idle. This makes sense since as the average number of arrivals per service approaches 1 the buffer on average will never be empty.

# Question 4:

For an intensity of 1.2 we observe a high average number of packets (125.91) in the queue and practically no idle time (1.26×10-6). This is because the average rate of arrivals is much higher than our rate of departures and in response the buffer backs up and cannot recover.

# Question 5:

Arrival and observer events are generated in the same way as the M/M/1 queue while the departure events will be generated during runtime.

During runtime, the events are processed in slightly different ways compared to the M/M/1 queue.

For arrival events, the current buffer capacity is checked before processing, if there is no capacity left (difference between arrivals and departures is equal to the buffer size), the arriving packet will be dropped, and a local variable will track how many have been dropped.

If the buffer queue is not full, the arrival event is processed and a departure event is generated. If the queue is empty, the departure time is equal to the arrival time plus the service time which is calculated by the exponential (same as M/M/1). If queue is non-empty, departure time will be wait time plus service time.

Text

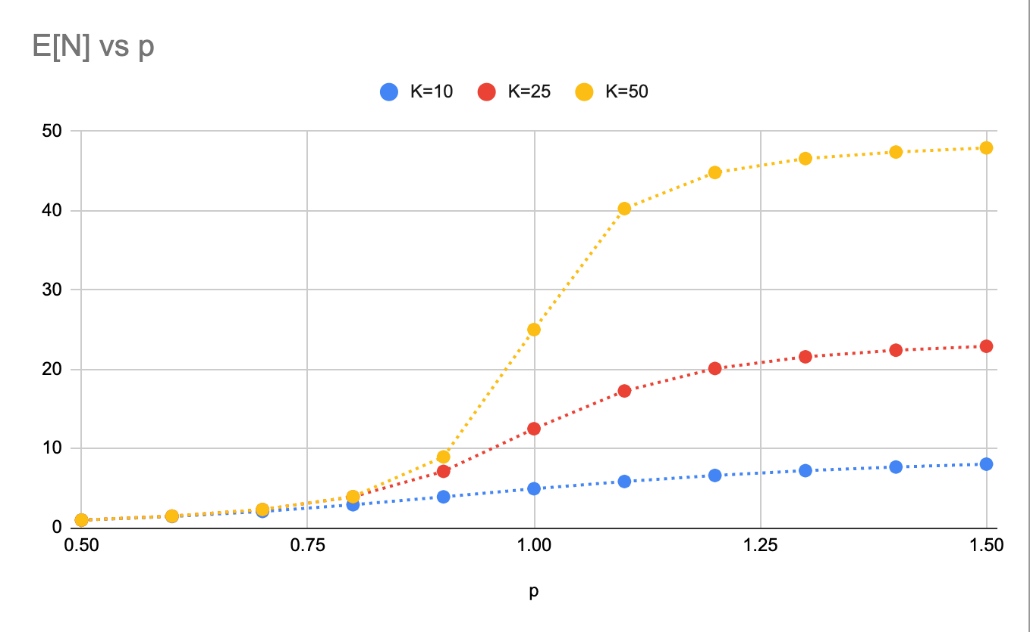
Description automatically generated

Departure events are stored in a separate queue from the observer and arrival events. The queue is first in, first out. Departure events are handled by comparing the departure time of the departure at the front of the queue to the incoming event time stamp. If the departure time is less than the incoming event time than it will be handled by increasing its counter and removing it from the queue. This process is repeated with the following departure until the departure time is after the incoming event time.

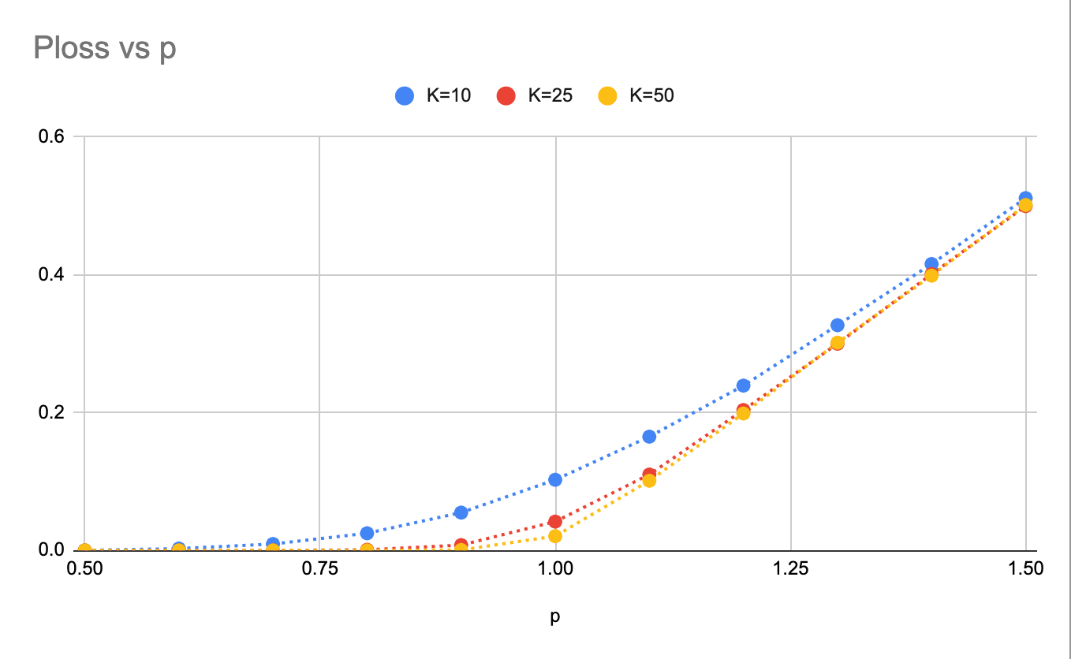
Text

Description automatically generated

# Question 6:



This graph shows the relationship between intensity and average number of packets in the queue for 3 different buffer sizes. In all cases, the graph starts out with exponential growth (although minimal in some cases), following the behavior of the relationship seen with infinite buffer size. When the average number of packets starts to approach the buffer size, the graph tapers off because even though there are more packets arriving, the limited buffer size means that more packets will be dropped. The buffer size determines the value that E[N] will converge to.

  
This graph shows how the probability of a packet being dropped changes as the arrival intensity increases. As intensity increases, the probability of a packet being dropped increases with it. The smaller buffer size has the probability increase at a faster rate initially when compared to the larger buffer sizes however as intensity increases, they all converge to around the same value.

PLOSS was calculated when an observer event arrived by dividing the number of packets dropped by the number of packet arrivals.

The following tables show the percent difference when the simulation was run with different time values. Some values of PLOSS have a huge difference because the PLOSS values are very small. However, once the PLOSS values are over 0.01, the %difference stays under 5%.

