Lab 1:
M/M/1 and M/M/1/K Queue Simulation

Group 42
Kelvin Tezinde (20619069)
Brynn Davis (20610315)

ECE358
Prepared for: Prof. Wasef
University of Waterloo, Ontario, Canada
September 25, 2019

ECE358 - Laboratory 1

How to run code:

- Requires python 3.6 or greater
- From command line `py main.py`

Question 1)

After using the equation provided in the lab manual of:

$$x = -\left(\frac{1}{\lambda}\right) ln(1-U)$$

A function was derived to build a list of 1000 results of this function, and then Numpy was used to calculate the mean and variance of the list. The results are as follows:

- Mean = 0.013466
- Variance = 0.000184

Using the exponential distribution process from the equation above, we can calculate an expected mean and variance for a set of data from the above Poisson point process:

- Mean = $\frac{1}{\lambda}$
- Variance = $\frac{1}{\lambda^2}$

Using the given lambda value of 75, the mean and variance can be expected to be 0.0133, and 0.000177 respectively. These calculated values match those obtained from out experiment very closely, indicating a will written and reliable function.

Question 2)

The simulation is built with a larger for loop going over the different rho values with steps of 0.1. For each rho value a simulation is performed, spanning values 0.25, 0.35, ..., 0.95. In each simulation lambda is calculated from knowing values of the capacity, avg packet length, and rho. With this, two lists of events are made to simulate event arrivals. This is done by starting at a time of zero, and adding a random value from the random number generator built in question 1, given the argument of the lambda previously calculated for the specific rho being used. Additionally, the lab specifies observations events are to occur five times as often so the given lambda is multiplied by 5 for the observation event list. These lists are then merged and sorted based on the timings of these events to create one large event queue containing both observations and arrivals, aptly called `event_queue`.
The core of the simulation occurs at this point. While the event queue has events, the next event gets popped. If it's an arrival, then a counter called `departures_waiting` gets incremented, if it's an arrival, we logs the value of `departures_waiting` in a list called `departures_waiting_log`. This is repeated until every event has been popped and the queue is empty. But what about departures? When the first arrival occurs, a variable called

`departure_time` is changed, from an impossible value (i.e. 100000), to the current time of this arrival event + transmission time based on the size of the packet. This packet transmission time is found using the same RNG method from Q1 but instead the rate parameter given is 1/2000 rather than lambda, and then the result is divided by the capacity to find the time taken to transmit those bits.
So, this newly made `departure_time` is actually checked every iteration after popping an event. The time of the event is compared to `departure_time`, and if `departure_time` is less than the event, a "departure" must be handled before the main event.
Handling a departure is a simple check to see if we have remaining departures or not (done by checking if `departures_waiting` is greater than 0). If we do, then we decrement the counter and calculate the new departure event's time as we did before (for the first arrival). If we don't have waiting departures, then we simply set `departure_time` to the impossible value again to indicate to arrivals that they will need to handle the next `departure_time` on their own, rather than just increment the `departures_waiting` counter.
After this has been completed, the list/log of `departures_waiting` counter values is returned where maths can be used to find the sum, length, and ultimately the average of the set. This is completed for each rho until the simulation is officially completed.
The performance metrics are largely obtained from the returned `departures_waiting_log` list. It can be used to find the maximum number of events ever queued at once (helpful for determining a suitable non-infinite buffer size). After calculating the average packets in the buffer, it can then be used to determine on average how many packets one could expect to wait before their got serviced. Lastly, taking a count of the times `departures_waiting` was 0, will give an idea of how idle the buffer was for a given rho.

Question 3)

The process described in question two was used to produce the following graph:

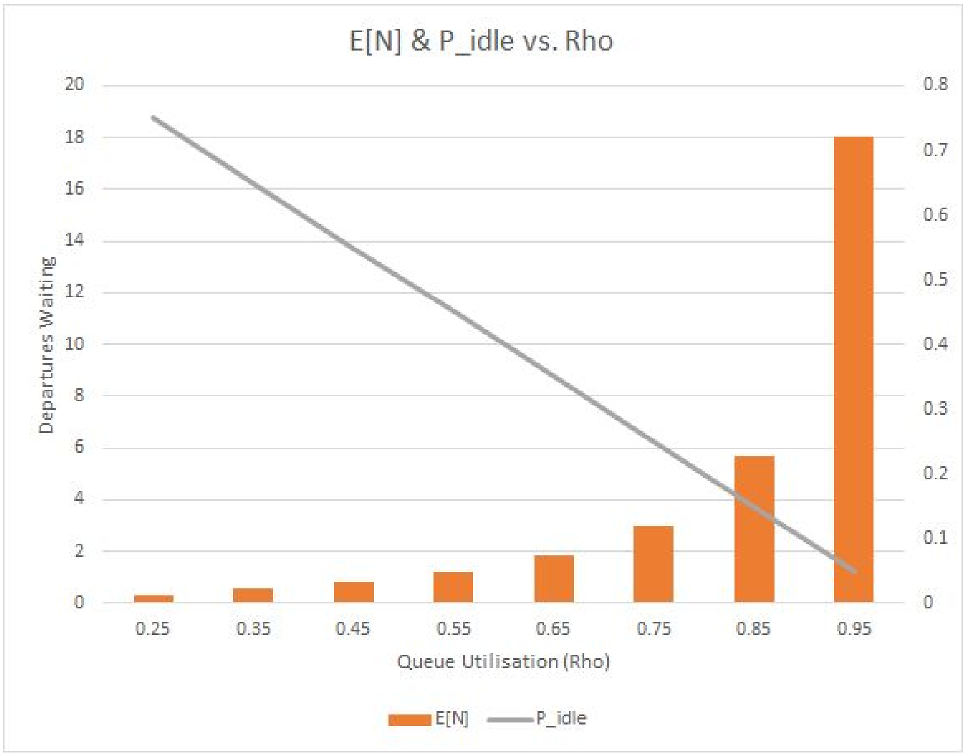

Figure 1: Average Packets in Buffer(E[N]) and Idle Proportion vs. Queue Utilisation (Rho)

It can be seen from Figure 1 that the quantity of average packers in the buffer increases exponentially as the queue utilisation increases linearly. This is an expected outcome, since as described in question 2, as rho increases, lambda does as well by the linear relationship of: $\rho = \lambda * L/C$. As this is an exponential distribution process, with lambda in the denominator, the resulting generated random numbers will amount to smaller values, and thus more arrivals. When obtaining the packet departure time, however, the argument passed of 1/L is independent of this change, and thus the transmission time stays. Essentially boiling down to faster arrivals, just as slow departures, and consequently a larger buffer as rho increases.

The same data used to find the average packets in the buffer is used to find the idle time. Essentially, whenever this value is zero, a counter is incremented and the total tally is divided by the number of events. As expected, as the queue utilisation increases the buffer becomes more often used. This results in less idle time which explains the decreasing nature of the experiment results, trending to 0% idle proportion with a fully utilised queue.

Question 4)

Reproducing the simulations with a queue utilisation of 1.20 results in the following:
Average packets in buffer: 50592.3
Average proportion: 6.99023e-06 (~0)
This is an expected result, because the queue utilisation parameter rho is greater than 1, meaning that packets will be entering at a rate faster than they can be processed. The average number of packets in the buffer would increase to even greater numbers had the simulation time been increased further.

Question 5)

The code used to build the m/m/1/k simulator is the exact same code that was used for the m/m/1 simulator in question 3, with only the following change:

```
            # arrival event
-           departures_waiting += 1
+           if departures_waiting == k:
+                 arrivals_dropped += 1
+           else:
+                 departures_waiting += 1
```

This simply checks to first see if we've reached the limit of the buffer size, and if so drops the packet. When a packet is dropped, a counter is incremented and at the end this counter is divided by the number of initial arrival events to find `p_loss`, rather than `p_idle`.

Question 6)

After producing a new simulation to cap the maximum amount of departures in the buffer, the following graph is obtained from experimenting with different rho and buffer limit values:

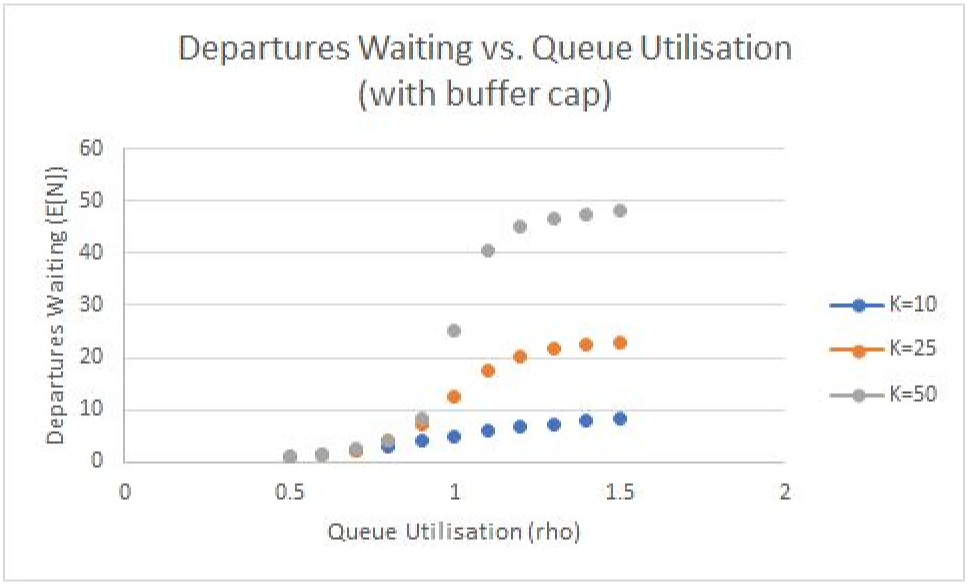

Figure 2: Average Packets in Buffer(E[N]) vs. Queue Utilisation (Rho) with buffer cap

Interestingly, all 3 series in Figure 2 follow a similar characteristic curve. With lower rho values, the average number of waiting departures remains relatively low (as previously discovered in the experiment of question 3). However, now the difference is that instead of increasing to large values (as they did in question 3), the average number of queued departures increases with rho only up until it reaches the hard limit of K in the m/m/1/k model.

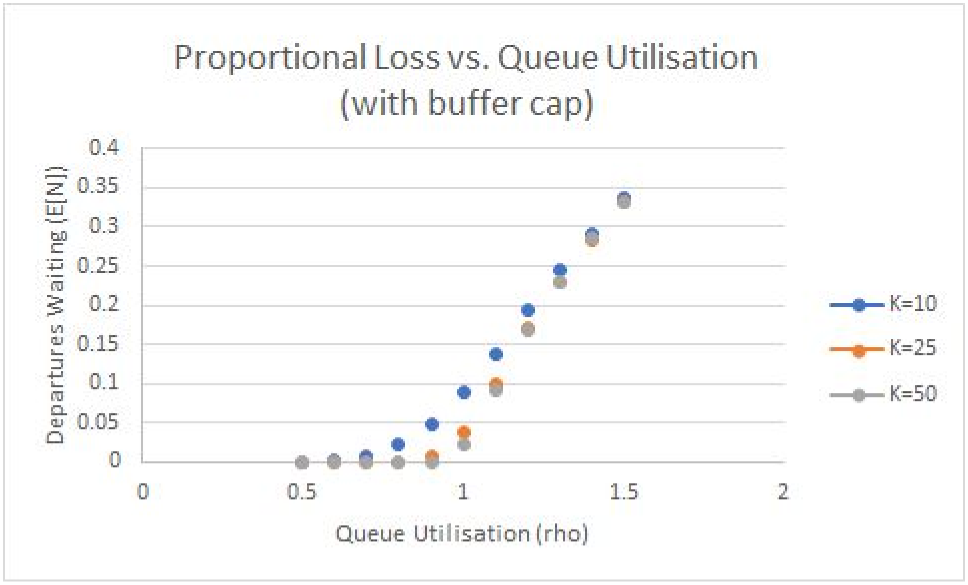

Figure 3: Average Packets in Buffer(E[N]) vs. Queue Utilisation (Rho) with buffer cap

Conversely, when examining the proportional loss introduced with the cap, it can be seen that as rho increases, so does the proportion of lost packets. This is expected, because as previously mentioned, this increasing rate of incoming packets without an increasing rate of departure will result in a more trafficked buffer. It’s expected that if this experiment were to continue onwards for larger rho values, eventually the proportion of lost packets would reach nearly 100%. This capture alone demonstrates the relationship of this parameter and its effect on the packet loss, it seemed needless to continue into higher rho values.

Something else worth mentioning, is the lack of impact the increase in rho had on the series of experiment ran with a buffer of 50. For earlier rho values, the packet loss remained at ~0% until finally reaching a threshold where the incoming packets overwhelmed the size of the buffer and eventually resulted in loss.