**ECE 358: Computer Networks**

**Project 1: M/D/1 and M/D/1/K Queue Simulation**

**Date of submission:** January 23, 2017

**Submitted by:**

**Student ID:** 20506122 **Student name:** Greavette, Ethan **Waterloo Email:** egreavet@uwaterloo.ca

**Student ID:** <#id> **Student name:** Gupta, Divyansh **Waterloo Email:** d29gupta@uwaterloo.ca

**Marks received:**

**Marked by:**

Contents

[Question 1 3](#_Toc472363090)

[Junit Tests 3](#_Toc472363091)

[Simulator Class 3](#_Toc472363092)

[Generator Class 4](#_Toc472363093)

[Server Class 4](#_Toc472363094)

[Reporter Class 4](#_Toc472363095)

[Question 2 5](#_Toc472363096)

[Question 3 6](#_Toc472363097)

[Question 4 7](#_Toc472363098)

[Source Code 9](#_Toc472363099)

# Question 1

**Explain your program in detail. Should there be a need, draw diagrams to show your program structure. Explain how you compute the performance metrics.**

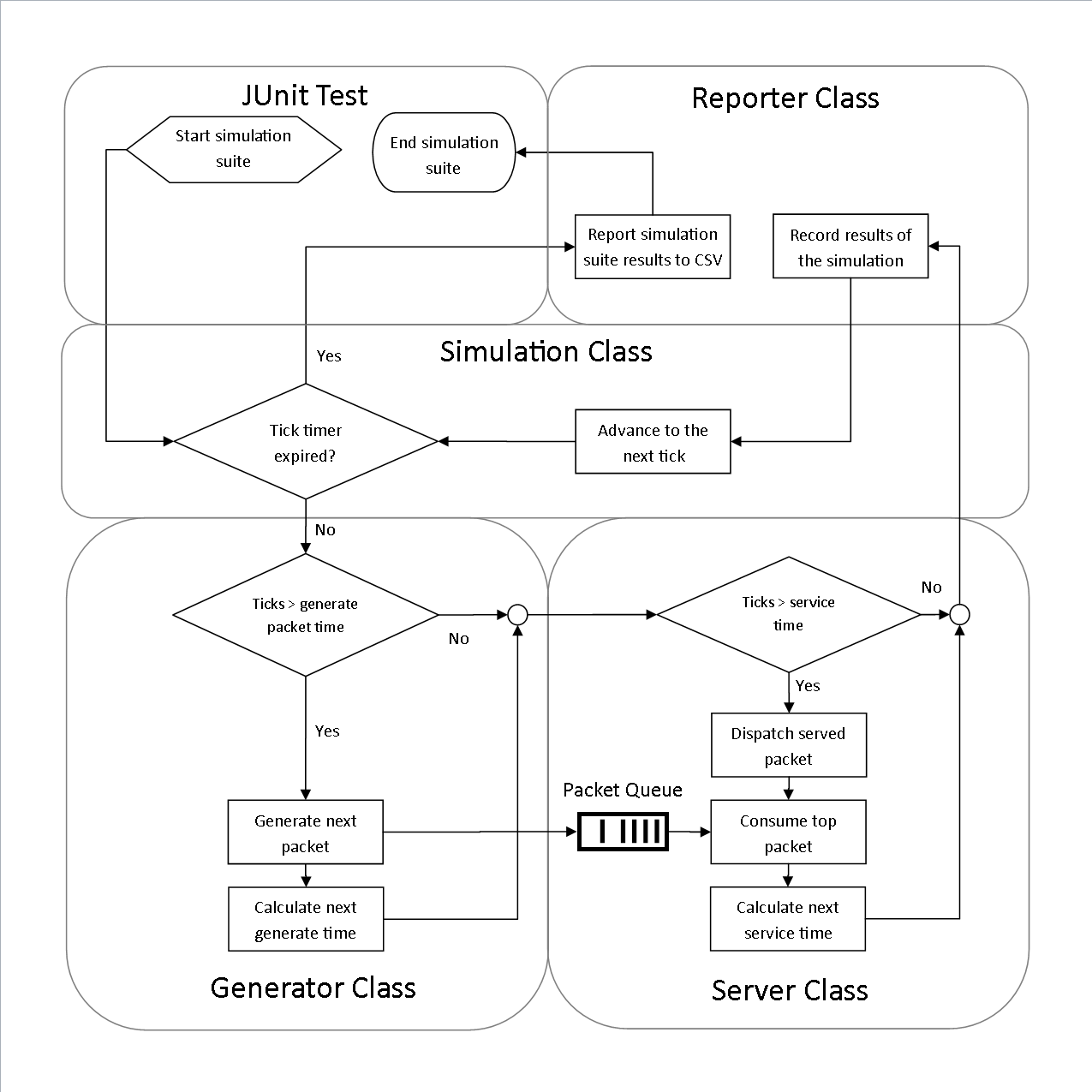


Figure 1 Diagram of simulation structure and flow

## Junit Tests

We created design of the simulation (seen above in Figure 1) with the idea that simulations would be ran multiple times with different variables. The meet this requirement we designed several JUnit tests that would setup the simulation parameters, run the simulation, record the results, change simulations parameters as required, and repeat. The additional value of using Junit was creating a series of tests to run while developing simulation that would validate the results. If one of the Junit tests failed, it might indicate a code has regression occur. Although the project did not specify either the JUnit tests or the Reporter class the additionally code made the project much easier to compare simulation results.

## Simulator Class

The core of the project is the Simulator class, which controls the overall flow of the simulation. When a JUnit test begins the simulation, it provides setup parameters to run the simulation. The simulation class contains the main program loop that updates the Generator, Server, and Reporter class with the current time tick until the current tick is greater than the total simulation time parameter.

## Generator Class

The Generator class encapsulates the packet arrival process such that the Simulator and Server are not aware of when a new packet will arrive. By specification, the packet arrives at a random interval, which we model by an exponential random variable that simulates packets arriving at packets as second. Each time a new packet is generated the Generator adds the packet to the queue and calculates the time in ticks until next packet arrives and then remains idle until that time expires.

## Server Class

The Server class is idle until there is a packet in the queue at which point the packet moves into a buffer service of the packet begins. The length of time it takes to service a packet is a function of the packet size in bits divided by the link speed in bits per second. Unlike the Generator, the Server takes same amount of time to process a packet. The deterministic nature of the service time means that the server has an upper limit of to the number of packets per second that it can process. When the throughput limit of the server is reached the packets will begin to accumulate on the packet queue. Once the service time expires, the packet is dispatched and the server starts on the next packet in the queue.

## Reporter Class

We saw that both Questions 2 and 4 required us to plot the simulation results so we created a Reporter class that would record the results of a simulation and write the results to a specified Comma Separate Variable (CSV) file. Spreadsheet programs like Excel can import CSV natively, which made plotting the simulations significantly easier. The Reporter is updated with the events that occurred for later reporter at the end of each tick loop. This process continues until the simulation time has expired and control of the application is returned to the Junit test. The results of the simulation are appended to a CSV file after which more simulations can be ran if required.

## Performance Metrics

The performance metrics are handled by the Reporter class which is updated each turn with the events of a simulation tick. For the average queue size E[N] metric the reporter stores a total sum of queue size each tick and then divides the sum by the total number of ticks for the simulation. For the average sojourn time E[T] the reporter stores a total sum of how long it take a packet from arrival to departure each time a packet departs the server and then divides the sum by the total number of sent packets for the simulation. The packet loss percentage P\_LOSS is calculated by recording the total number of packets lost by the total number of packets that arrived. For the switch idle percentage P\_IDLE metric the reporter stores a total sum of the queue was empty each tick and then divides the sum by the total number of ticks for the simulation.

## Simulation Optimization

The simulation process is balance between having accurate results and the length of time it takes to run a test. If a larger tick ratio used then the total number of packet arrived would decline due to the random nature of the packet arrival rate. The larger the time slices between program cycles the more likely multiple packets could arrive in between a time slice. This would result in a skipped packet that never arrived in the queue and was never included in the final metrics. If a small tick ratio is used the simulation will take a long time to complete however less skipped packets would occur. To pick an accurate tick ratio a series of simulations with the tick ratio ranging between one to one and selected one as the point where the maximum arrival rate from question four had a steady state skipped packet rate. Using one tick ratio for both simulations gave us a good idea of where the metrics for the JUnit validation should be set for a pass.

In an effort to both optimize the simulation run time and improve the simulation accuracy an event based simulation was developed. Instead of advancing time in incremental steps every loop, time would increment to the next event. Only two events can occur in this simulation: when a packet arrives in the queue and after a packet is serviced. By advancing to the next event many idle program cycles are skipped greatly improving the speed of the simulation. The pre-optimized question four simulations, repeated 10 times with one tick ratio, took 27 minutes to complete. The post-optimization time for the same simulation took 76 seconds to complete which is a speed improvement of 2031%. The post-optimization accuracy results where acceptable close to the pre-optimized version, which supports the accuracy of the post-optimization. In theory, the accuracy is limited to the java double data type which stores the next packet arrival tick random variable U is calculated. If the random variable U is close to one the packet will arrive very closely to the previous packet and would be skipped if the tick time slice is too large. An event-based simulator would advance time as small the possible resolution of the double type and catch every packet that arrive. As a result the event based optimization both improved both the speed and the accuracy of the simulator.

# Question 2

**Let L = 2000 bits and C = 1Mbps. Use your simulator to obtain the following graphs. To vary ρ, you need to calculate the corresponding value of λ (recall the relationship between ρ, λ, L and C). All the simulations must be repeated at least 5 times.**

Figure 2 E[N] as a function of ρ (for 0.2 < ρ < 0.9, step size 0.1).

Figure 3 E[T] as a function of ρ (for 0.2 < ρ < 0.9, step size 0.1).

Figure 4 P\_IDLE as a function of ρ (for 0.2 < ρ < 0.9, step size 0.1).

# Question 3

**Build a simulator for an M/D/1/K queue, and briefly explain your design.**

Both the M/D/1 and M/D/1/K simulators are very similar in functionality except that the M/D/1 has an infinite queue size whereas the M/D/1/K is limited to a queue size of K. To design the simulator so it works for both type of queues we defined the queue size value -1 to mean infinity. Thus, for question two all of the simulations were ran with the queue size as -1 and the P\_LOSS value is ignored. For question four, the queue size was set to value of 10, 25, or 50 and then P\_LOSS value is included in the plot. For any queue size value other than -1 the queue size is checked before adding a new packet. If the packet would overflow the queue it is dropped and the recorded as a lost packet. The reporter records the lost packet so the packet loss percentage is calculated in the simulation’s report. Using this design, we are able to use the same source code for both types of queues.

The addition of a queue limit changed the dynamic of model when the network utilization was greater than one. In early simulations, it was noticed that the packets received from the Server class began to stabilize when was greater than one. After some investigation, it was discovered that the throughput limit of the switch had been reach and the switch began to drop packets. The link speed of switch for both simulations was one Mbps and each pack was 2000 bits in length which meant a theoretical maximum throughput of 500 packets a second. The simulation supports the theoretical maximum throughput as the virtual switch’s throughput approached 500 packets a second for values of greater than one. As a result, all of the plots for question four have a significant data trend shift when the network utilization is approximately one.

# Question 4

**Let L=2000 bits and C=1 Mbps. Use your simulator to obtain the following graphs:**

**E[N] as a function of ρ (for 0.5 < ρ < 1.5, step size 0.1), for K = 10, 25, 50 packets. Show one curve for each value of K on the same graph.**

Figure 5 E[N] as a function of ρ (for 0.5 < ρ < 1.5, step size 0.1), for K = 10, 25, 50 packets

**E[T] as a function of ρ (for 0.5 < ρ < 1.5, step size 0.1), for K = 10, 25, 50 packets. Show one curve for each value of K on the same graph.**

Figure 6 E[T] as a function of ρ (for 0.5 < ρ < 1.5, step size 0.1), for K = 10, 25, 50 packets.

**P\_LOSS as a function of ρ (for 0.5 < ρ < 1.5, step size 0.1) for K = 10, 25, 50 packets. Show one curve for each value of K on the same graph. Explain how you have obtained P\_LOSS.**

To obtain P\_LOSS the Reporter class records every instance of packet arriving at the queue and every instance a packet is lost. After the simulation is complete, the Reporter class calculates the packet loss percentage (P\_LOSS) by the following formulas:

Figure 7 P\_LOSS as a function of ρ (for 0.5 < ρ < 1.5, step size 0.1) for K = 10, 25, 50 packets.

**P\_IDLE as a function of ρ (for 0.5 < ρ < 1.5, step size 0.1) for K = 10, 25, 50 packets. Show one curve for each value of K on the same graph.**

Figure 8 P\_IDLE as a function of ρ (for 0.5 < ρ < 1.5, step size 0.1) for K = 10, 25, 50 packets.

# Source Code