ECE358 – Project 1

Queue Simulation

Objective:

After this experiment, you should understand:

i. The basic elements of discrete event simulation. In particular, the notion of an event scheduler for simulating discrete event systems and the generation of random variables for a given distribution.

ii. The behavior of a single buffer queue with different parameters C, L, K and λ (as defined below), as it is a key element of a computer network.

iii. The need to be careful about when a system should be observed to obtain the correct performance metrics.

Programming Languages: You may only use C or C++.

Overview:

The performance of a communication network is a key aspect of network engineering. Performance measures like delay (how long a packet stays in a system or takes to be delivered), loss ratio (the percentage of the packets that are lost in the system), etc. are some of the criteria that network engineers are trying to predict or measure since they affect the Quality of Service (QoS) offered to the users. In order to do so, models are built to help understand the system and predict its behavior. A *model* of a system is a mathematical abstraction that captures enough of the system's features to give good estimates of the system's performance. The purpose is to build a model and to analyze it to get results on the performance we can expect. Hence depending on the performance we want to study (e.g., reliability, loss, delay,…) we will build a different model of the system. In real situations the complexity of a system can be extremely high (this is the case in a network), and hence the model describing it can be very complicated. In that case it may be difficult to analyze the model exactly. We can either simplify the model (which is an art, since we need to keep it relevant) or try to solve the complicated model using approximations. In both cases, the results thus obtained **should** be validated by *simulation*.

Simulation is not only useful for validating approximate solutions but also in many other scenarios. During the design of a system, it allows us to compare potential solutions at a relatively low cost. It is also very useful to dimension a system, i.e. to decide how much resources to allocate to the system based on an a-priori knowledge of the inputs. It is also used to check how potential modifications of an existing system could affect the overall performance. Simulation is also especially useful when it is difficult or impossible to experiment with the real system or take the measurements physically, e.g., measuring the performance of the Internet as the whole, performing nuclear reactions, airplane crash tests, etc. With the use of computer simulations, the above mentioned scenarios can be reproduced to help us gain insights about the behavior of the system.

To perform a simulation we need a model of the system, as well as models for its input(s). This will be discussed later in more detail.

In communication networks, the most frequent basic-block model we encounter is a queue. You are going to design a simulator, and use it to understand the behavior of two basic types of queues.

Background Materials:

In this experiment, you will be asked to simulate a FIFO (First-In-First-Out) queue (see Figure 1). In general a queue is described by three to five parameters, with a slash ("/") separating each of them, e.g., G/G/1/K/FIFO. This is referred to as *Kendall notation*. We are interested in the first four parameters (the fifth parameter is the service discipline, which is FIFO by default).

The first and second parameters are descriptions of the arrival process and the service process, respectively. "M" means *memoryless* or *Markovian*, "D" means *Deterministic* and "G" means *General*. The third one is the number of servers, and the fourth one is the size of the buffer. So, for example M/M/c means that both the arrival and service processes are Markovian, and that there are c servers. If the fourth parameter is not present, it means that the buffer size is infinite. M/G/1 means the queue has one server, the arrival process is Markovian and the service process is general (some non-Markovian distributions, e.g., Gaussian).

If the arrival process is M, it means that the distribution of the time between successive arrivals (also called *inter-arrival time*) is identical for all inter-arrivals, is independent from one interarrival to another and is exponentially distributed. This is equivalent to saying that the process counting the arrivals is Poisson.

If the service process is M, it means that the distribution of the service time is identical for each customer/packet, is independent from one customer to another and is exponentially distributed. The exponential distribution is often used in performance evaluation because it is an easy distribution to use (it is characterized by only one parameter) and it is adequate to model call durations and call arrivals in a telephone system. In a data communication system such as the Internet, it is not so adequate for modeling the arrival process of packets but is still used due to its simplicity.

If the service process is D, it means that each customer/packet will receive the same constant service time.

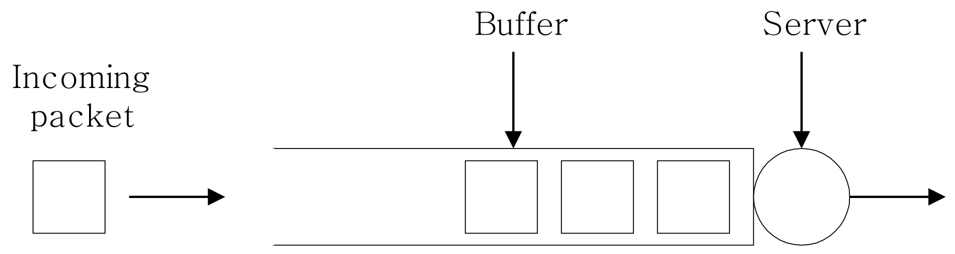

Figure 1 Model of a queue

The queues you need to simulate in this experiment are M/M/1 and M/M/1/K.

<u>Simulation Basics</u>

A simulation model consists of three elements: *Input variables, Simulator and Output variables*. The simulator is a mathematical/logical relationship between the input and the output. A simulation experiment is simply a generation of several instances of input variables, according to their distributions, and of the corresponding output variables from the simulator. We can then use the output variables (usually some statistics) to estimate the performance measures of the system. The generation of the input variables and the design of the simulator will be discussed below.

<u>Generation of input variables with different distributions:</u>

You will need a uniformly distributed random variable (random variable) generator. In particular, you will need to generate U(0,1), a uniform random variable in the range (0,1). It is important to use a "good" uniform random variable generator for better simulation results. A good uniform random variable generator will give you independent, identically distributed (i.i.d.) uniform random variables. You can test how good a uniform random variable generator is by checking the mean (0.5) and variance (1/12) of the random samples you have generated. The mean and variance of a set of such samples should be very close to the theoretical values for 1000 or more samples. Note that, in general, it is not enough to check the mean and variance to conclude on the validity of your random generator, but we will assume that it is a good enough test for our purposes. Since this is not a course on simulation we cannot spend too much time on the topic of the generation of a random variable but note that you should always start by verifying the performance of your random variable generator by checking that you are indeed generating what you want.

If you're running your code on ecelinux (which you should be), gcc's standard rand() function should be adequate.

If you need to generate a random variable X with a distribution function[1] F(x), all you have to do is the following:

- Generate U~U(0,1)
- Return $X = F^{-1}(U)$

where $F^{-1}(U)$ is the inverse of the distribution function of the desired random variable.

<u>Question 1</u>: How would you generate an exponential random variable with parameter λ from U(0,1)? Show all your calculations. Write a short piece of C code to generate 1000

[1] To learn more about density and distribution functions, please refer to the book by Alberto Leon-Garcia, "Probability and Random Processes for Electrical Engineering", second edition(July 1993), pp. 87.

exponential random variable from your equation, using $\lambda=75$. What is the mean and variance of the 1000 random variables you generated? Do they agree with the expected value and the variance of an exponential random variable with $\lambda=75$**?** (if not, check your code, since this would really impact the remainder of your experiment).

<u>Simulator design:</u>

Discrete Event Simulation (DES) is one of the basic paradigms for simulator design. It is used in problems where the system evolves over time, and the *state* of the system changes at discrete points in time when some *events* happen. These discrete points in time are not necessarily separated by equal duration. DES is suitable for the simulation of queues. *However, please note that it is not the only way to simulate a queue as simple as the ones that we are asking you to simulate. In this lab, we are not interested in simulating a queue for the sake of it (this can easily be done with Matlab) but to use it as an application for learning about DES.*

A queue is made of 2 components, a buffer and one (or many) server(s). Since our service discipline is FIFO, the first packet in the buffer enters the server as soon as the previous one has been served. If a packet enters an empty system (by "system" we mean the queue and server), it enters directly into the server. If the buffer has the maximum number of packets that it can hold (i.e. K), then the new packet is dropped.

The *event*s that can happen in a queue are either an *arrival* of a packet in the buffer or the *departure* of a packet from a server. *Dropping* of a packet because the buffer is full when the packet arrives is **NOT** considered to be an event in the simulation, but a consequence of an arrival event.

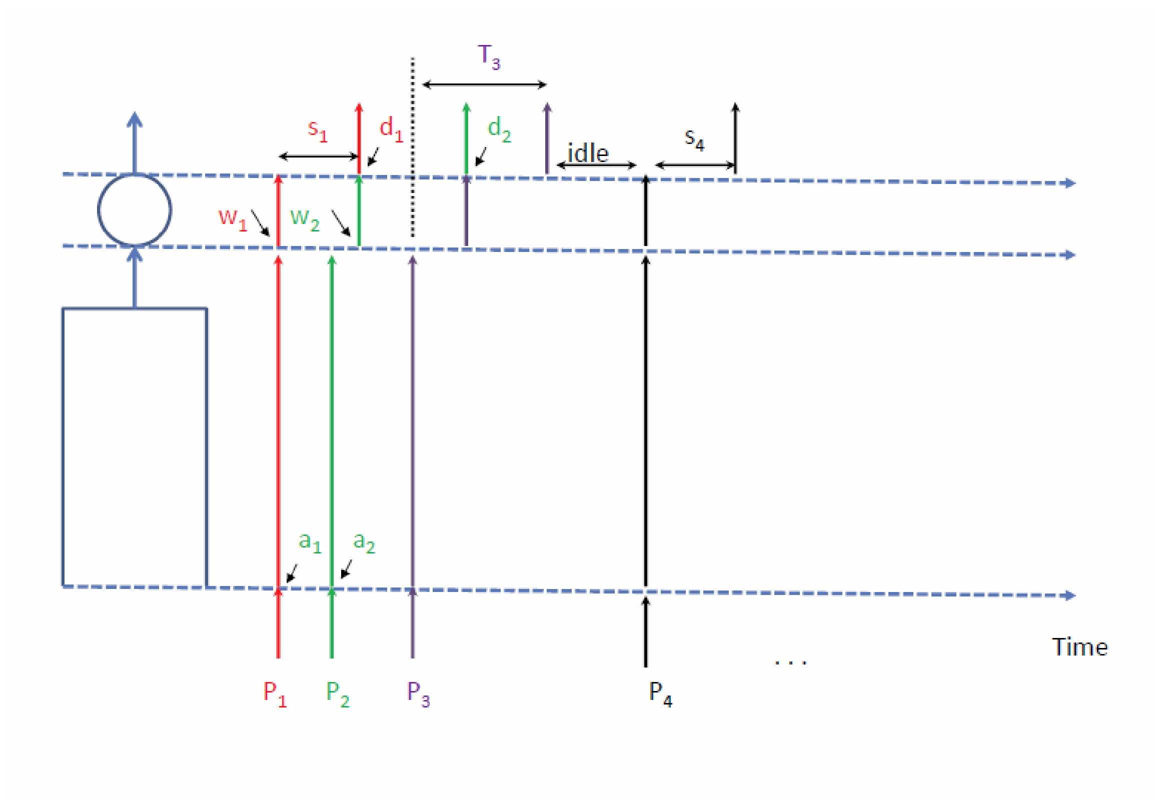

Figure 2: The dynamics of a infinite single server queue

Figure 2 depicts the behavior of a typical single server queue with an infinite buffer. Let P_n be the nth packet, a_n is its arrival time, s_n is its service time, d_n is its departure time and T_n is its sojourn time (i.e., the total time spent in the system (buffer and server)). You might find it useful to record the time w_n at which packet P_n enters the server.

The *state* of the system will depend on what you are interested in. In this lab, we will use only *the number of packets in the system* N(t) as the state of the system. We are interested in computing the time-average of the number of packets in the queue, E[N], the proportion of time the server is idle (i.e., the system is empty), P_{IDLE} and in the case of a finite queue, the probability that a packet will be dropped (due to the buffer being full when it arrives). The measurement and recording of the state along time should be performed such that we have statistically representative information on the performance that we are interested in.

Important note: if we record the state of the system only at times when a packet arrives or departs, we will in general see a biased (i.e., non representative) state of the system. As an example, consider a D/D/1 queue as shown in Figure 3, where packets arrive at a constant rate of 1 packet every 5 minutes and the service time for each packet is 1 minute.

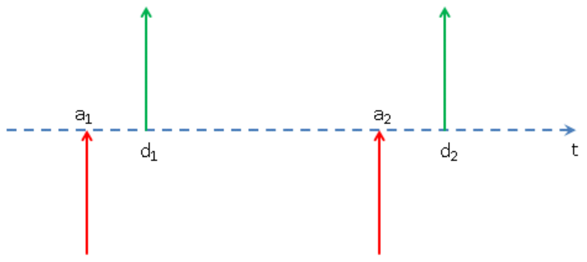

Figure 3: Arrival and departure times in a D/D/1 queue

In this case, if we look at the system at only arrival and departure times (a_1, d_1,a_2,d_2,…), P_{idle} = 100% which is not true (Question: what is P_{idle} in this system?). Note that a random observer (i.e., an observer who would come at random times) would see the proper state (i.e., 20% of the time, the system is non-idle and 80% it is idle). A solution to this problem is to add another type of events called the *observer* event. Here, we use an observer coming at random times generated according to a Poisson distribution of parameter α (it can be shown mathematically that a random Poisson observer will see a statistically representative state of the system if the simulation is long enough and if α is chosen correctly). Whenever you process an observer event, you will record the state of the system. Therefore, the simulation will now contain **three** events: packet arrival, packet departure, and observer arrival. Note that you have to ensure that the observer event is *independent* of the other two events.

DES uses the notion of an *event scheduler (ES)*. Let A_n be the arrival event of packet P_n, D_n be the departure event of packet P_n, and O_n be the nth observer event arrival. For each event, the time at which it will occur is recorded, i.e., $t(A_n) = a_n$, $t(D_n) = d_n$, and $t(O_n) = o_n$. Then *ES* contains a set of time-ordered events E_i, i=1,…, N, such that $t(E_i)<t(E_{i+1})$ for all i along with the time of their occurrences. Hence, an entry for event E_i

in the ES contains its type (A_n, D_n or O_n) as well as the time of its occurrence t(Ei). Here is an example of an *ES*: *ES* = {(E_1=A_1,10), (E_2=D_1,11), (E_3=A_2,12), (E_4=O_1,16), (E_5=A_3,17), (E_6=O_2,18), (E_7=A_4,21), (E_8=D_2,23), (E_9=D_3,28), (E_{10}=O_3,31),…}.

Given an arrival process, service process, and observation process, it is easy to generate an *ES beforehand*. Then, you are going to move from event E_i to E_{i+1} and decide how to update your variables. Note that in general, you cannot generate all the events beforehand.

In summary, in order to create a DES for a simple queue with an infinite buffer, you should do the following:

1. Choose a duration T for your simulation (see later how to choose T).
2. Generate a set of random observation times according to a Poisson distribution with parameter α and record the observer event times in the ES (generate new events as long as their arrival times are less than T). Put all these events in ES.
3. Generate a set of packet arrivals (according to a Poisson distribution with parameter λ) and their corresponding length (according to an exponential distribution with parameter 1/L), and calculate their departure times based on the state of the system (the departure time of a packet of length L_p depends on how much it has to wait and on its service time L_p/C where C is the link rate). Put all these events (packet arrivals and departures) in ES. Recall ES is by definition a time-ordered list of events.
4. Now initialize the following variables to zero: N_a = number of packet arrivals so far, N_d = number of packets departures so far, N_o = number of observations so far.
5. Dequeue one event at a time from your event scheduler (i.e., read one event at a time from the ES, starting with the first in the list), and call the corresponding *event procedure*. Each event procedure (one per type of event) will consist of updating some of variables. Clearly when the event is the arrival of an observer, you record the values of your performance metrics.

<u>Note:</u>

1. Change state and output variables *ONLY* when events happen.
2. An *event procedure* consists of the update of the system variables (depends on what the event is) and the update of the output variables.

An extremely important problem is to determine how long you should run your simulation (i.e. how many repetitions do you need) in order to get a *stable* result. There are a lot of variance reduction techniques out there that help you to determine when to stop, but that is not a requirement for this experiment. An easier way (again, not the best way, but sufficient for this experiment) to do this is to run the experiment for T seconds, take the result, run the experiment again for 2T seconds and see if the expected values of the output variables are similar to the output from the previous run. For example, the difference should be within x% of the previous run, where in our case x should be less than 5. If the results agree, you can claim the result is stable. If not, you try again for 3T, 4T, … If you cannot seem to find a proper T, it may mean that your system is unstable. For the purpose of this experiment, the value of T should be more than 10,000 seconds but you should still use the procedure described above to check the stability (and hence the validity) of your results.

Note that while this is adequate for this lab, in practice you should become familiar with the concept of a confidence interval.

Experiment:

Simulate an **M/M/1** queue, and **an M/M/1/K queue**. Let:

- λ = Average number of packets generated /arrived per second
- L = Average length of a packet in bits
- α = Average number of observer events per second
- C = The transmission rate of the output link in bits per second.
- ρ = Utilization of the queue (= L λ/C)
- E[N] = Average number of packets in the buffer/queue
- E[T] = Average sojourn time (queuing delay + service time)
- P_{IDLE} = The proportion of time the server is idle
- P_{LOSS} = The packet loss probability (for M/M/1/K queue). This is the ratio of the total number of packets lost because the buffer was full when they arrived, to the total number of packets generated.

Note that α and λ should be of the same order.

M/M/1 Queue

Recall that this queue has an infinite buffer.

Question 2: Build your simulator for this queue and explain in words what you have done. Show your code in the report. In particular, define your variables.

Question 3: Assume **L=12000 bits, C=1 Mbits/second** and give the following figures using the simulator you have programmed. Provide comments on all your figures:

1. E[N], the average number of packets in the system as a function of ρ (for $0.25 < \rho < 0.95$, step size 0.1). Explain how you do that.
2. P_{IDLE}, the proportion of time the system is idle as a function of ρ, (for $0.25 < \rho < 0.95$, step size 0.1). Explain how you do that.

Question 4: For the same parameters, simulate for ρ=1.2. What do you observe? Explain.

M/M/1/K Queue

Let K denote the size of the buffer in number of packets. Since the buffer is finite, when the buffer is full, arriving packets will be dropped.

Question 5: Build a simulator for an **M/M/1/K** queue. Explain what you had to do to modify the previous simulator and what new variables you had to introduce. Show your code in the report. (Hint: do not forget that dropping is NOT considered an event)

Question 6: Let **L**=12000 bits and C=1 Mbits/second. Use your simulator to obtain the following figures (provide comments for each figure):

1. E[N] as a function of ρ (for $0.5 < \rho < 1.5$, step size 0.1) for K=5, 10, 40 packets. (One curve per value of K on the same graph). Compare it with the case $K=\infty$.
2. P_{LOSS} as a function of ρ (for $0.4 < \rho < 10$) for K=5, 10, 40 packets. (One curve per value of K on the same figure). Explain how you have obtained P_{LOSS}. Use the following step sizes for ρ:

   For $0.4 < \rho \leq 2$, step size 0.1.
   For $2 < \rho \leq 5$, step size 0.2.
   For $5 < \rho < 10$, step size 0.4.

What To Turn In:

1. Submit a print copy of your report with the following details:
   - Cover page: provided at the end of this document
   - Table of contents
   - Answers/explanations/comments for all the questions.
2. Submit your source code in the LEARN dropbox. Be sure to include a makefile that builds and runs your code.

Reference:

Averill Law, W. David Kelton, *Simulation Modeling and Analysis*, 2nd edition, McGraw-Hill, 1991.

<Cover Page>

ECE358: Computer Networks

Winter 2018

Project 1: Queue Simulation

Date of submission:

Submitted by:

Student ID:

Student name <Last name, First name>

Waterloo Email address

Marks received: <Leave this blank>

Marked by: <Leave this blank>