ECE 358: Computer Networks
Project 1: Lab Manual
Project Title: M/D/1 and M/D/1/K Queue Simulation
(Date: September 8, 2015)

Table of Contents

OBJECTIVE

On the Internet, messages are queued up in routers and host computers, and queues impact the end-to-end delay suffered by messages. Therefore, it is important to understand the impact of various network parameters on network delay. The objective of this lab work is to design a simulator to understand the behaviour of two basic types of queues (M/D/1 and M/D/1/K). After this experiment, students are expected to learn:

- the basic elements of a network simulator; and
- the behaviour of a single buffer queue with different parameters.

EQUIPMENT

A computer/ laptop with a C compiler. (You may choose a different language.)

OVERVIEW

The performance of a communication network is a key aspect of network engineering. Delay and packet loss ratio are two important performance metrics. The first metric is a measure of the time it takes the packet to reach its destination since it was generated. The second metric represents the percentage of data packets that are lost in the system. Delay and packet loss are components of the broader concept of Quality of Service (QoS) offered by a network to the users. In order to evaluate the QoS offered by networks, models are built to help understand the system. The three common types of network models are *analytical* (i.e. mathematical) models, *simulation* models, and *prototype implementation* models. It is difficult to build exact analytical models of complex systems. Therefore, we try to get an approximate analytical model which is validated with a *simulation* model.

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 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 measures physically, e.g., measuring the performance of the Internet as the whole, performing nuclear reaction, plane crash tests, etc. With the use of computer simulations, the above mentioned scenarios can be reproduced to help us gain insights about the behaviour of the system. To perform a simulation we need a model of the system, as well as models for input(s). This will be discussed later in more details. In communication networks, the most frequent basic-block model we encounter is a queue. In this project, you are going to simulate and understand the behaviour of two basic types of queues.

BACKGROUND MATERIAL

I. Kendall Notation (G/G/1/K/FIFO)

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 3-5 parameters, with a slash (“/”) separating the parameters. For example, G/G/c/K/FIFO describes a queue in Kendall notation. 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. 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 that the queue has one server, the arrival process is Markovian and the service process is general (some non-Markovian distributions, e.g., Gaussian). A Markovian arrival process 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 inter-arrival to another, and is exponentially distributed. A service process M 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 was 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. A service process D means that each customer/packet will receive the same constant service time.

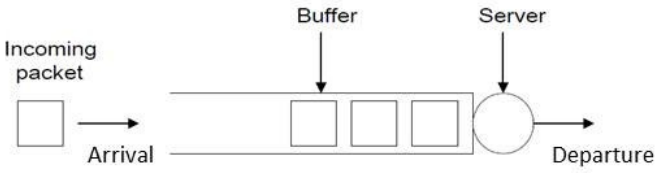


Figure 1: Model of a queue.

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

II. Basics of Simulation Modeling

A simulation model consists of three kinds of 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 then use the output variables (usually some statistics) to analyze the performance measures of the system. The generation of the input variables and the design of the simulator will be discussed next.

III. Generation of input variables with different distributions

It is required that you have a uniformly distributed random variable (r.v.) generator. Specifically, 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. 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 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.

Note: 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 need to generate a random variable *X* with a distribution function F(x), you need to do the following:

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

where $F^{-1}(U)$ is the inverse of the distribution function of the desired random variable. If F(x) is an exponential distribution function, then X will be the corresponding exponential random variable. The inter-arrival time between two packets in a Markovian Process will be X.

See Appendix for an explanation of the computation of an exponential random variable.

IV. Designing a Discrete Event Simulator (DES)

Discrete Event Simulation (DES) is commonly used to study the performance of evolving systems whose *states* change at discrete points in time in response to external and internal *events*. Systems involving queues are generally simulated with a DES simulator. A queue comprises two components: a buffer and one or many servers. The *event*s that can happen in a queuing system

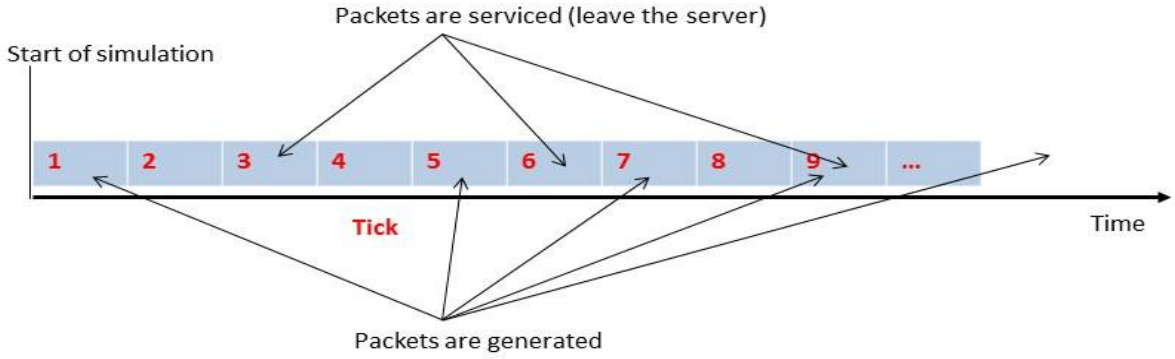


Figure 2: The broad picture of a queue simulation.

are *arrival* of packets in the buffer and *departure* of packets from the server(s).

Figure 2 depicts the interplay among packet generation, packet buffering, and packet servicing, and the concept of discrete time. In the following, all those elements are explained in sufficient detail:

- **Discrete time:** It is important to note that your computer runs many *processes*, including system processes and user processes. Your DES is not going to be the only application running on your computing hardware. Therefore, you must not be dependent on system calls to keep track of time and generate the "next" data packet as and when it should be generated. Rather, you are going to use the concept of "ticks" to interpret time and delay in your DES. In Figure 2, the time axis has been shown to comprise an infinite number of ticks, with each tick having a programmer-defined duration. For example, one tick can be

one second, one hundred milliseconds, one millisecond, ten microseconds, one nanosecond, and so on. Smaller tick values give better simulation accuracy at a huge expense of actual simulation time. For example, to simulate the behavior of a wireless network for ten minutes, a simulator using a tick value of 10 ms may take two hours, whereas the same simulator using a tick value of 1 ms may take five days to complete. Therefore, choose an appropriate tick duration depending upon your needs. A good practice is to use a longer duration in the debugging phase of your coding but use a shorter duration while collecting performance data.

See Appendix B for the pseudo code of the implementation of the concept of ticks.

- **Packet generator:** A packet generator produces data packets in some ticks, as shown in Fig. 2, depending upon the mathematical model explained before. The time gap between successive packet arrivals is known as inter-arrival time. For example, you generate a packet in tick #1 and compute the next tick (say, #5) when you will generate the next packet. In tick #5, you generate a data packet and compute the next tick (say, #7) when you will generate the next packet. In tick #7, you generate a data packet, and compute the next tick when you will generate the next packet. And, the process repeats till the end of the simulation. So, when the packet generator is called, it decides whether or not to generate a packet in that tick. If a packet is generated, then call the module that implements the buffer (see Fig. 1) and compute the tick when you will generate the next packet.

- **Packet server:** The packet server (see Fig. 1) is called once in each tick from the for() loop implementing ticks (Appendix B). The server decides if the packet will continue to receive service or if this is the final tick for the packet. All this depends upon the nature of the server and packet characteristics. For the purpose of this project, you will be given enough details to decide when a packet has been completely served. Note that if the buffer is empty, the server remains idle. Also, if the server finishes serving its current packet, it picks up the next packet in the following tick.

- Define an appropriate data structure to represent each data packet so that you can record a variety of information, such as, when the packet was generated, and so on. You should decide what kinds of statistics are useful and record relevant information.

V. Notations

- λ = Average number of packets generated /arrived (packets per second)
- L = Length of a packet in bits
- C = The service time received by a packet. (Example: The transmission rate of the output link in bits per second.)
- ρ = Utilization of the queue (= input rate/service rate = L λ/C)
- E[N] = Average number of packets in the buffer/queue
- E[T] = Average sojourn time. The sojourn time is the total time (queuing delay + service time) spent by the packet in the system. You need to use the tick when the packet was generated and the tick when the server pushed it out.
- P_{IDLE} = The proportion of time the server is idle
- P_{LOSS} = The packet loss probability (for M/D/1/K queue). It is the ratio of the total number of packets lost due to buffer full condition to the total number of packets generated.
- M = The number of times you repeat your experiments. Typically, 5 <= M <= 10. See the note in red at the end of the for() loop given in Appendix B.

VI**. Working Source Code:** There should be a provision for entering the relevant inputs and obtaining the relevant outputs for the queue. The TA should be able to perform the simulation while running your source code.

(1) M/D/1 : Input – TICKS, λ, L, C Output – E[N], E[T], P_{IDLE}
(2) M/D/1/K : Input – TICKS, λ, L, C, K Output – E[N], E[T], P_{IDLE}, P_{LOSS}

QUESTIONS: Read all the questions before you start designing your simulator(s).

I. M/D/1 Queue: Recall that it is a queue with an infinite buffer.

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.

Question 2: Assume λ=100 packets/ second, L=2000 bits, C=1 Mbps (Megabits per second). Give the following statistics using your simulator by repeating the experiments **5** times. Here Mega means 10^6 and Kilo means 10^3.

- E [N]
- E [T]
- P_{IDLE}

Question 3: 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).

- E[N] as a function of ρ (for $0.2 < \rho < 0.9$, step size 0.1).
- E[T] as a function of ρ (for $0.2 < \rho < 0.9$, step size 0.1).
- P_{IDLE} as a function of ρ (for $0.2 < \rho < 0.9$, step size 0.1).

Briefly discuss your graphs.

II. M/D/1/K Queue: Let K denote the size of the buffer in number of packets. Since the buffer is finite, packets arriving at a full buffer will be dropped.

Question 4: Build a simulator for an M/D/1/K queue, and briefly explain your design.

Question 5: Let L=2000 bits and C=1 Mbps. Use your simulator to obtain the following graphs:

- E[N] as a function of ρ (for $0.5 < \rho < 1.5$, step size 0.1), for K = 10, 25, 50 packets. Show one curve for each value of K on the same graph.
- E[T] as a function of ρ (for $0.5 < \rho < 1.5$, step size 0.1), for K = 10, 25, 50 packets. Show one curve for each value of K on the same graph.
- P_{LOSS} as a function of ρ (for $0.5 < \rho < 1.5$, 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}.
- P_{IDLE} as a function of ρ (for $0.5 < \rho < 1.5$, step size 0.1) for K = 10, 25, 50 packets. Show one curve for each value of K on the same graph.

Briefly discuss your graphs.

FINAL REPORT

Submit a print copy of a document with the following details:

1. Cover page: A sample cover page can be found at the end of this document
2. Table of contents

3. Answer all the questions and provide explanations as asked for.
4. Source code.

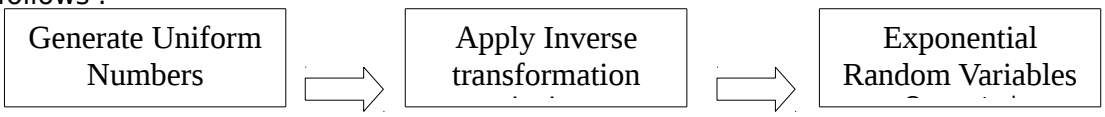
You may be asked to give a demo of your simulator. Should there be a need for electronic submission, you will be provided with additional details.

REFERENCES: A. Law, W. D. Kelton, *Simulation Modeling and Analysis*, 2nd edition, McGraw-Hill, 1991.

Appendix A

The exponential random variable can be calculated as follows:

Exponential random variables can be generated from uniformly generated numbers U(0,1) as follows :



Let 'X' be an exponential random variable we want to generate. The cumulative distribution function can be given as:

$F(x) = P(X <= x)$

$= \int f(x)\,dx$

$= \int f(x)\,dx + \int f(x)\,dx$

where $f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$

Therefore,

$F(x) = \int_{}^{x} \lambda e^{-\lambda x}\,dx$

$F(x) = 1 - e^{-\lambda x}$

$F(x) = U \implies X = F^{-1}(U)$

$1 - e^{-\lambda x} = U$

$e^{-\lambda x} = 1 - U$

$-\lambda x = \ln(1-U)$

$X = (-1/\lambda) \ln(1-U)$

where U is a uniformly generated number and X is the exponential random variable.

Example: Assume that your random number U is 0.3. You generated the random value in the range [0,1].
Assume that λ = 100 packets/sec. The next value of X based on the currently generated random number 0.3 is calculated as follows:

$X = (-1/\lambda) * \ln(1 - 0.3)$ sec <== See the details in the project description

$= (-1/100) * \ln(0.7)$ sec

$= (-1/100) * (-0.356674943)$ sec

= 0.356674943/100 sec
= 0.00356674943 sec
= 3.56674943 ms

Assume that 1 tick = 1 microsecond = 10^(-6) sec
X = 3.56674943 ms
= 3566.74943 micro-sec
= 3566.74943 ticks
=~ 3567 ticks

In summary, if you generated a packet in the current tick (say, n), you chose a random number 0.3 in the current tick, and the next packet will be generated in tick number n + 3567. In tick number (n + 3567), generate another random number and from it the next value of X.

Appendix B

Given the outlined abstract nature of ticks, one can easily implement the concept of ticks as follows:

```
for (i = 1; i <= TICKS; i++) {
        /* TICKS is an integer constant that represents the duration of simulation */
        /* The time duration for which you want to simulate a system = TICKS *
tick_duration
          where tick_duration has been explained in the Discrete Time paragraph.
          Choose those two constants depending upon your need. */

        /* This for() loop essentially implements the time axis with all those ticks in Fig. 2. */

        /* Now you can do whatever you want to do in the i-th tick:
           Call the data packet generator to try to generate a new data packet.
           Call the server to let it know that another tick has elapsed, and the sever will
           decide if servicing the current data packet will end in this tick, thereby pushing
the
           data packet out of the queue.
        */
} // Essentially, this loop drives all other modules in your simulator
/* Now compute the various performance metrics using data that you have stored while
executing
    the for() loop. */
// NOTE: Repeat the for() loop 5-10 times to compute average values…. One run
of the for() loop with, say, TICKS = 100000 is not enough.
```

<Cover Page>

ECE 358: Computer Networks

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

Date of submission:

Submitted by:

Student ID: Student name <Last name, First name>: Waterloo Email address

Student ID: Student name <Last name, First name>: Waterloo Email address

Marks received: <Leave this blank>

Marked by: <Leave this blank. The TA will put his/her initial here.>