LAB MANUAL

ECE 358 - Project 1

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

(Date: May 11, 2015

Version: May.11.2015)

Table of Content

OBJECTIVE

To design a simulator and use it 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 simulation, and
- the behaviour of a single buffer queue with different parameters C, L, K and λ (as defined below).

EQUIPMENT

A computer / laptop with C/C++ compiler. (You are free to choose another language and compiler)

OVERVIEW

The performance of a communication network is a key aspect of network engineering. *Delay* and *packet loss ratio* are two important metrics. Delay is a measure of the time it takes the packet to reach its destination since it was generate (i.e. how long a packet stays in a system or takes to be delivered). Packet loss ratio represents the percentage of the data packets that are lost in the system. These two metrics 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 and predict its behaviour. A *model* of a system is a mathematical abstraction that captures sufficient features of the system 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 want. Hence depending on the performance we want to study (e.g., delay, loss ratio, reliability et. al.) we will build a different model of 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 such as a network. Therefore, we try to get an approximate analytical model which should be 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 MATERIALS

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 three to five parameters, with a slash ("/") separating each of them, i.e. G/G/1/K/FIFO (it is called the Kendall notation). We are interested in the first four parameters (the fifth parameter is the service discipline which is FIFO by default).

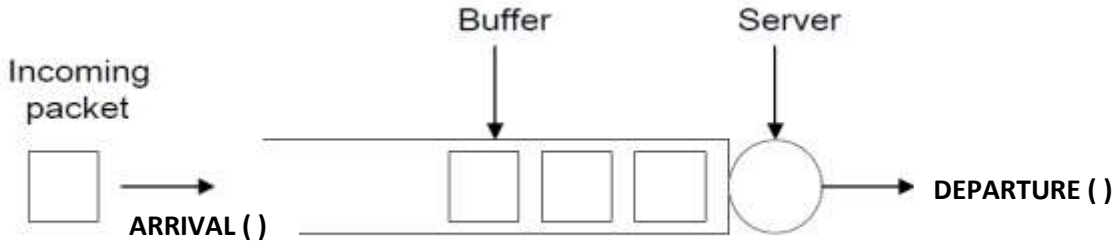


Figure 1 Model of a queue

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 t*he 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 arrival process is *Markovian* and the service process is *general* (some non-Markovian distributions, e.g., Gaussian), and that there is *one* server.

A *Markovian arrival process* means that the distribution of the time between successive arrivals (also called inter-arrival) is identical for all inter-arrivals, is independent from one inter-arrival to another and is exponentially distributed. This is equivalent to say that the process counting the arrivals is Poisson. A *Markovian* service process 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 *Deterministic* service process means that each customer/packet will receive the same constant service time.

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 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 next.

III. Generation of input variables with different distributions:

It is required that you have a uniformly distributed random variable (r.v.) 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 and variance 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 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 need to generate a random variable X with a distribution function[1] F(x), all you have to do is 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 A for detailed steps of generating an exponential random variable.

IV. Designing a Simulator- Discrete Event Simulation (DES)

Discrete Event Simulation (DES) is one of the basic paradigms for simulator design. It is used in problems where the system evolves in time, and the *state* of the system changes at discrete points in time when some *events* happen. Discrete event simulation is suitable for the simulation of queues.

A queue is made of two 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 is finished to be served. If a packet enters an empty system, it enters directly into the server. 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 simulation, but a consequence of an arrival

event. The inter-arrival time between two events is determined by the event's process, and only the nearest future occurrence needs to be recorded. It will be updated when the current event happened. The *state* of the system will depend on what you are interested in, and we will use the number of packets in the queue as the state.

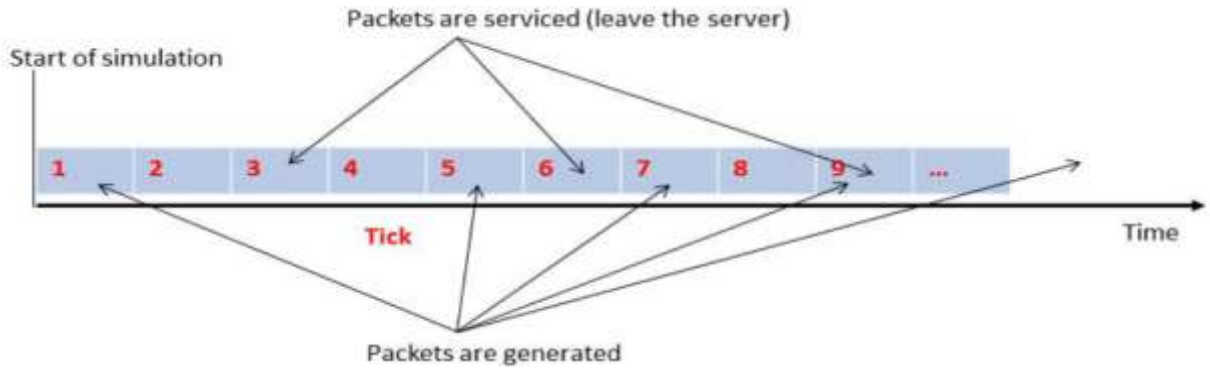


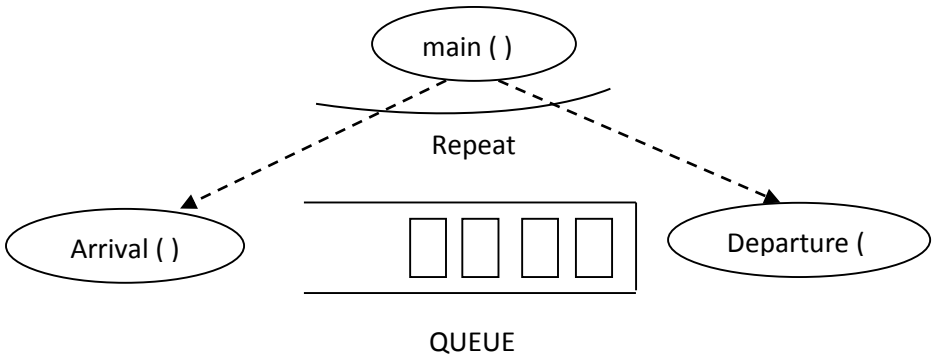
Figure 2: The broad picture of a queue simulation

Figure 2 depicts the interplay among packet generation, packet buffering, and packet servicing, and the concept of discrete time. Each element in simulator is explained as follows:

- **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 Figure 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 Figure 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.
- **Packet data structure**: 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. Pseudocode:



The queue can be implemented as a global data structure (link list or array). Arrival and Departure events are called at every tick. Arrival event will place the packet in the queue whenever a packet is generated (as per the exponential distribution). Departure event will remove the packet from the queue after the elapse of the deterministic service time.

```
main( )
{
      /*Initialise important terms such as t_arrival = exponential r.v, # of
      pkts in queue = 0,  t_departure = t_arrival ( this implies that first
      time departure will be called as soon as a packet arrives in the
      queue*/
      Start_simulation (ticks);
      Compute_performances ( );
}

Start_simulation (int ticks)
{
      for (t=1; t<= Ticks; t++)
      {
            Arrival (t);     /* call the arrival procedure*/
            Departure (t);   /*call the departure procedure*/
      }
}

Arrival (int t)
{
      /* Generate a packet as per the exponential distribution and insert the
      packet in the queue (an array or a linked list)*/
}

Departure (int t)
{
        /* Check the queue for the packet, if head of the queue is empty,
        return 0 else if the queue is non-empty delete the packet from the
        queue after an elapse of the deterministic service time. */
}
```

```
Compute_performances ( )
{
        /*Calculate and display the results such as average number of packets
        in queue, average delay in queue and idle time for the server. */
}
```

Caution: This is just the skeleton of the code, you need to add more code to compute the required results. You should carefully identify how and when **`arrival()`** and **`departure()`** functions will be executed and necessary parameters will be captured.

VI. Important notations:

- λ = Average number of packets generated /arrived (packets per second)
- L = Length of a packet (bits)
- C = Transmission rate of the output link (bits per second)
- ρ = Utilization of the queue (= input rate/service rate = $L\lambda/C$)
- $E[N]$ = 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
- P_{IDLE} = The proportion of time the server is idle
- P_{LOSS}= The packet loss probability (for M/D/1/K queue)
- M = The number of times you repeat your experiments. Typically, $5 \leq M \leq 10$. See the note in red at the end of the **`for()`** loop in Appendix B.

INSTRUCTIONS

I. **Points to remember for discrete event simulation***:*

- Change state and output variables *ONLY* when events happen.
- You should decide what kinds of statistics are useful and record relevant information.

II. **Recording output variables:** The output variables are the result we want. They are usually statistics of the system, like the average delay of a packet, average loss, etc. You need to know what you need to record to obtain the performance results that are requested. 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 must 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 $T \times 10^6 \, \mu secs$ of time (with a tick of 1 μsec and hence **ticks = T x 10^6** # of iterations), take the result and repeat the experiment for 5 times 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. For the purpose of this experiment, the value of T should be more than 5000.

III. **Assumptions and cautions about the time unit for simulation**

a) The number generated by the uniform random generator and thus the calculated exponential random number have a unit of second. It needs to be converted into μs.
b) One tick is equal to 1 μs.
c) The unit of service time should also be in μs.
d) The required results like average delay will be expressed in terms of ticks
e) Be careful that the time elapsed during simulation is independent of the time of the machine on which your program is running
f) You need to take the readings on every tick.

IV. **Graph Plotting:** Plot graphs for relevant questions, and discuss what you observe.

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

where n is the simulation length in number of ticks.

QUESTIONS:

I. M/D/1 Queue

Recall that it is a queue with an infinite buffer.

Question 1: What is the service time received by a packet?

Question 2: 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 3: Assume λ=100 packets/ second, L=2000 bits, C=1 Mbits/second. Give the following statistics using your simulator by repeating the experiments **5** times.

1. $E\ [N]$
2. $E\ [T]$
3. P_{IDLE}

Question 4: Let L=2000 bits and C=1Mb/s. Use your simulator to obtain the following graphs. To vary ρ, you need to calculate the corresponding value of λ (recall the relationship among ρ, λ, L and C).

1. $E[N]$ as a function of ρ (for $0.2 < \rho < 0.9$, step size 0.1).
2. $E[T]$ as a function of ρ (for $0.2 < \rho < 0.9$, step size 0.1).
3. 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, when the buffer is full, arriving packets will be dropped.

Question 5: Build a simulator for an M/D/1/K queue. Explain what you had to do to modify the previous simulator and what new variables you had to introduce.

Question 6: Let L=2000 bits and C=1 Mbits/second. Use your simulator to obtain the

following graphs:

1. $E[N]$ as a function of ρ (for $0.5 < \rho < 1.5$, step size 0.1), for K = 5, 10, 20, 50 packets. (Show one curve for each value of K on the same graph).
2. $E[T]$ as a function of ρ (for $0.5 < \rho < 1.5$, step size 0.1), for K = 5, 10, 20, 50 packets. (Show one curve for each value of K on the same graph)
3. P_{LOSS} as a function of ρ (for $0.5 < \rho < 1.5$, step size 0.1) for K = 5, 10, 20, 50 packets. (Show one curve for each value of K on the same graph). Explain how you have obtained P_{LOSS}.
4. P_{IDLE} as a function of ρ (for $0.5 < \rho < 1.5$, step size 0.1) for K = 5, 10, 20, 50 packets. (Show one curve for each value of K on the same graph).

Discuss the graphs.

DELIVERABLES

Submit a final report and the source code to the Learn Dropbox.

- **Report**: Your report is a text document in PDF or word format with the following details:
  1. Cover page: Make an informative page with a title and student details (names, email addresses, and student ID)
  2. Table of contents
  3. Answer all the questions and provide explanations as asked for.
- **Source code**: provide the entire source code with build instructions in a README file.

Demonstration

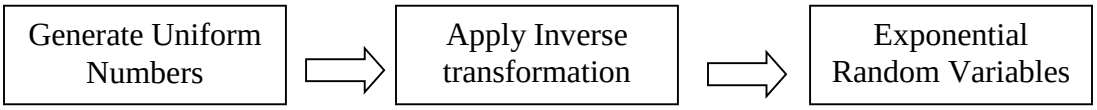
Teaching assistants will schedule project demonstration meetings with your group after the submission to check the functionality of the simulator and the contribution of each group member.

REFERENCES

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

Appendix A

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:

$$\begin{aligned} F(x) &= P(X \leq x) \\ &= \int_{-\infty}^{x} f(x)dx \\ &= \int_{-\infty}^{0} f(x)dx + \int_{0}^{x} f(x)dx, \end{aligned}$$

where

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

Therefore,

$$\begin{aligned} F(x) &= \int_{0}^{x} \lambda e^{-\lambda x} dx \\ &= 1 - e^{-\lambda x}. \end{aligned} \quad (1)$$

Let

$$F(x) = U, \quad (2)$$

we have

$$X = F^{-1}(U).$$

Substitute equation (1) into (2), we have

$$1 - e^{-\lambda x} = U. \quad (3)$$

Rearranging equation (3) gives

$$e^{-\lambda x} = 1 - U, \quad (4)$$

which is equivalent to

$$-\lambda x = \ln(1 - U). \quad (5)$$

From (5), we have the following formulae to compute X given U:

$$X = -\frac{1}{\lambda}\ln(1 - U), \quad (6)$$

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

Example:

Assume that the generated uniformly distributed random number U is 0.3 and λ = 100 packets/sec.
The next value of X based on the currently generated random number 0.3 is calculated as follows:

$$X = -\frac{1}{\lambda}\ln(1 - 0.3)$$
$$= 3.56674943 \times 10^{-3} \text{ sec}$$

Assume that 1 tick = 1 μs = 10^{-6} sec,

$$X = 3566.74943 \times 10^{-6} \text{ sec}$$
$$= 3566.74943\ \mu s$$
$$\approx 3567 \text{ ticks}$$

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