LAB MANUAL

ECE 358 - Project 1

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

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, the students should understand:

i) The basic elements of simulation.

ii) The behaviour of a single buffer queue with different parameters C, L, K and λ (as defined below).

EQUIPMENT

A computer / laptop with C compiler. (You are free to use any language and compiler)

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 behaviour. 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 want. 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 of 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 will then try to get an approximate solution of the model. This approximate solution 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 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).

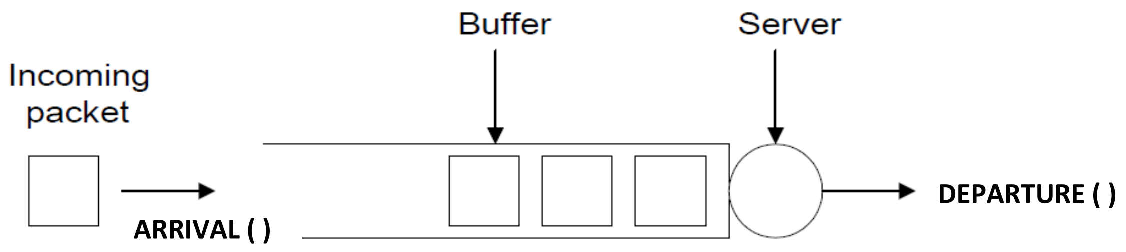


Figure 1 Model of a queue

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 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). An arrival process is M, 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 service process is 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 is D, 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 (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 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:

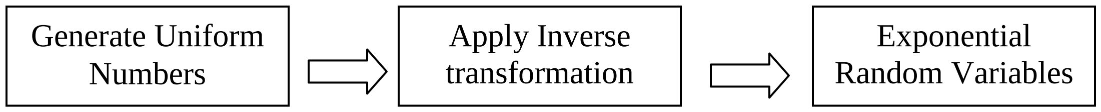
- Generate $U \sim U(0,1)$
- Return $X = F^{-1}(U)$

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

Note that 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.

The exponential random variable can be calculated as follows:

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



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

$$F(x) = P(X <= x)$$
$$= \int_{-\infty}^{x} f(x)dx$$
$$= \int_{-\infty}^{0} f(x)dx + \int_{0}^{x} f(x)dx$$

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

Therefore,

$$F(x) = \int_{0}^{x} \lambda e^{-\lambda x} dx$$
$$F(x) = 1 - e^{-\lambda x}$$
$$F(x) = U \quad == > \quad X = F^{-1}(U)$$
$$1 - e^{-\lambda x} = U$$

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

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

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

Where, U is the uniformly generated number and X is the exponential random variable.

/* the C code for the generated exponential random variable is enclosed*/

To verify that the C code is working properly:

Calculate the expected value of Mean and Variance as :

Expected Mean = $1/\lambda = 1/100 = 0.01$

Expected Variance = $1/\lambda^2 = 1/10000 = 0.0001$

And generate 1000 exponential r.v , calculate the mean and variance as follows and compare the results with expected value:

```
main( )
{
   int j;
          double u,lambda=100,x,sum=0,Mean,var1=0,Variance;

   sgenrand(4357); /* any nonzero integer can be used as a seed */

   for (j=0; j<1000; j++) {
                        u = genrand();
               x=(-1/lambda)*log(1-u);
      printf("urand = %5f \n", genrand());
                        printf("exp. rv = %5f\n ",x);
                        sum = sum +x;
                        var1=var1 + x*x;
      if (j%8==7) printf("\n");
   }
          Mean = sum/1000;
          Variance = 0.001*var1-(Mean*Mean);
          printf("Mean = %5f\n",Mean);
          printf("Variance = %5f\n",Variance);
   printf("\n");
}
```

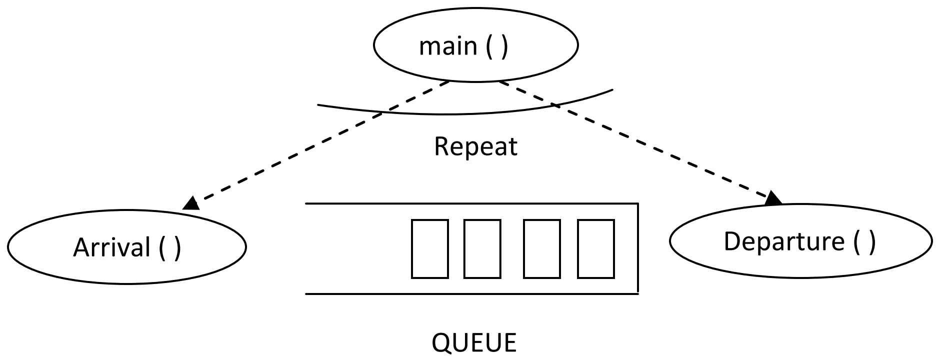
Mean and variance obtained from the program should be equal to the expected mean and variance

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 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 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.

V. Pseudocode:



Here 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

- 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/K/1 queue)

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 $T \times 10^6$ *μsecs* of time (with a tick of 1 μsec and hence **ticks =** $\boldsymbol{T \times 10^6}$ # 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 till the simulation reaches the required end time.

IV. **Graph Plotting:** Plot graphs for relevant questions. Comment on graphs, like what is your observation.

V**. Working Source Code:** There should be a provision of 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}

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? Compute the utilization of the queue (ρ) in terms of L, λ and C.

Question 2: Build your simulator and explain your program in details. Explain how you have defined the variables. Explain the complete logic of your program. Explain how you are going to compute the performance metrics.

Question 3: Assume λ=100 packets/ second, L=2000 bits, C=1 Mbits/second. Give the following statistics using the simulator you have programmed. Explain how you computed each of the parameter in your simulator.

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

Repeat the experiment for 'K' times for 'T x 10^6' ticks and compute the mean, where K= 5 and T = 5000.

Question 4: Let L=2000 bits and C=1Mb/s. Use the simulator you have developed to obtain the following graphs. Note that for varying ρ, you need to actually calculate the corresponding value of λ (you need to recall the relationship between ρ, λ, L and C) , which will generate the required exponential random numbers (inter-arrival time between incoming packets).

1. E[N] as a function of ρ (for $0.2 < \rho < 0.95$, step size 0.05).
2. E[T] as a function of ρ (for $0.2 < \rho < 0.95$, step size 0.05).
3. P_{IDLE} as a function of ρ (for $0.2 < \rho < 0.95$, step size 0.05).

Comment all the graphs.

Question 5: Repeat question 4 for L=5000 bits, C=1Mbits/s, and vary ρ from 0.5 to 0.95.Compare your results to that of question 4. Find a relationship between the average number of packets in buffer in the two cases? Find a relationship between the sojourn time in the two cases?

Question 6: For the same parameters as in question 4, explain what happens for ρ=1.2 ?

III. 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 7:*Build a simulator for a 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 8: Let L=2000 bits and C=1 Mbits/second. Use your simulator to obtain the following graphs:

1. N as a function of ρ (for $0.5 < \rho < 1.5$, step size 0.1) for K=5, 10, 20, 50 packets. (One curve per value of K on the same graph).
2. T as a function of ρ (for $0.5 < \rho < 1.5$, step size 0.1) for K=5, 10, 20, 50 packets. (One curve per 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. (One curve per value of K on the same graph). Explain how you have obtained P_{LOSS}. For K=5, also try $\rho = 5$.
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. (One curve per value of K on the same graph)

Comment the graphs above. What happens to P_{LOSS} when $\rho = 5$?

FINAL REPORT:

1. Solution for question 1.
2. Answers for questions 2 and 3.
3. Three graphs for question 4, three graphs for question 5. You can either write your comments on the graphs or on a separate sheet
4. Answers for questions 6 and 7.
5. Four graphs for question 8. You can either write your comments on the graphs or on a separate sheet.
6. Please attach the source codes of your simulators.

FORMAT:

A **hard copy** as well as a **digital file** of all documents must be submitted. The soft copy must include your code.

SUPPLEMENTARY MATERIAL

Enclosed is the source code for exponential random number generator in C

Reference:

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