Simulation project

Objectives

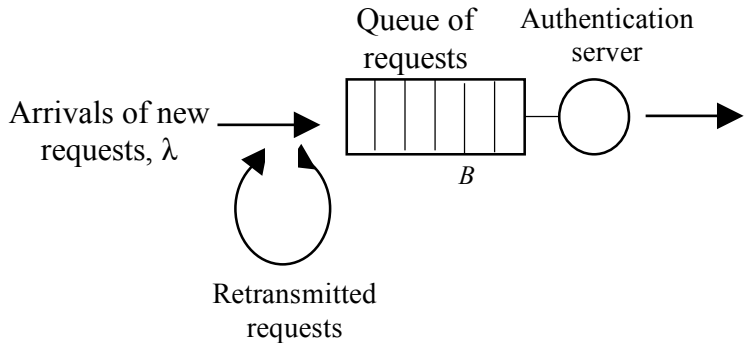
The objective is to simulate the re-certification process of a number of IoT devices with a view to characterizing its perfromance.

You can use any high-level language of your choice. You can develop your program on your personal computer, but it has to run and compile on eos. Programming elegance is not required! What is important is that your program runs correctly. For this, you are *strongly* advised to follow the instructions in this handout!

Remember that any exchange of code with another student is forbidden as it constitutes cheating. We will run Moss at the end of this assignment to verify that there has not been any code sharing.

Project description

We assume that 1,000 IoT devices have to be recertified periodically, and that they do this during a fixed period of 5 hours. The devices send a re-certification request to an authentication server, but not at the same time. Rather, each IoT device waits for a period uniformly distributed in (0, 5] hours, before it transmits its request. In order to avoid overflows of the buffer that is allocated to the certification process which runs on a server, we impose an upper bound B on the queue size (this is the total number of requests waiting to be processed plus the one in service). A request that arrives at the server when there are less than B requests in its queue is allowed to join the queue. Otherwise it is rejected and a message is sent back to the IoT device. In this case the IoT device goes through a random delay d and retransmits its request. (Overflows may occur due to misconfigurations whereby the processing rate of the server and the population of the sensors can be such that the resulting server utilization is very high, thus giving rise to a very large queue which may exceed the buffer space.)



A queueing representation of the model

Due to the large number of IoT devices, it is impossible to simulate each one individually. Instead, we assume that requests arrive in a Poisson manner at the rate of λ=1,000/5x60x60=0.0556/sec. That is, the inter-arrival time of successive new requests is assumed to be exponentially distributed with a mean $1/\lambda$=1/0.0556=17.98 sec. We will refer to these requests as *new request*. A blocked request is retransmitted after a delay *d* randomly chosen between (0, *D*], and it will be referred to as a *retransmitted request.*

Below we describe the implementation of the simulation model. The goal is to estimate the mean and 95th percentile and their confidence interval of a) the time *T* it takes to certify an IoT device, and b) the time *R* it takes a new request to be certified if it is rejected the first time. This will be done using the method of repetitions in order to simulate at least 30 5-hour periods.

The simulation structure

The basic idea in a simulation is to track the occurrence of the events in the system that change the state of the system. In our simulation, we have the following events:

1. Arrival of a new request
2. Arrival of a retransmitted request (there may be several pending such arrivals)
3. Completion of a service time

The occurrence of one event may trigger the occurrence of one or more events, as will be seen below.

I. Events and actions

1. Initial conditions at time master clock (MC)=0:
   - Queue empty, server is idle.
   - Predermine the time of the first new arrival, i.e., draw an exponential variate with mean $1/\lambda$, and add it to 0. This is the time of the first new arrival.

2. Events and logic
   1. Arrival of a new request:
      a. Once a new request arrives, determine the next arrival of a new request. That is, draw an exponential variate with mean $1/\lambda$, and add it to the current time indicated by the master clock (MC).
      b. If the queue size is less than *B*, then the new arrival enters the buffer. If the buffer is empty and the server is idle, then schedule a new service completion.
      c. If the queue size is equal to *B*, then the new arrival is rejected. In this case, the rejected request will be retransmitted after a delay *d* uniformaly distributed in (0, *D*]. Schedule a new event for the arrival of the retransmisted request at time MC+*d*.
   2. Arrival of a retransmitted request:
      a. If the queue size is less than than *B*, the arrival enters the buffer. If the buffer is empty and the server is idle, schedule a new service completion.

a. If the queue size is equal to B, then the arrival is rejected again. In this case, follow 1c.

2. Service completion:
Once a service is completed, generate the next service completion time if the queue is not empty.

3. Event list
The event list consist of the events: a) time of arrival of a new request, b) service completion time, and c) time of arrival of a retransmitted request (there may be more than one). That is, it will have the following structure:

Next new arrival
Service completion
Arrival of retransmitted request
. . .
Arrival of retransmitted request

The event list has a variable number of entries, and it should be implemented as a linked list or a fixed array, but be careful so that it does not overflow!

4. Stoppage rule
Once all 1,000 IoT devices have been recertified, the simulation is stopped. This may happen before the 5 hours are up, or it may take longer than 5 hours. For this you will have to setup a count of the number of requests that have been successfully processed, so you know when to stop the simulation.

II. Logic of the simulation program

The main logic of the simulation is as follows:

a. Locate the event from the event list that has the smaller clock value t. This requires that you search the event list for the smallest clock value so that to identify the next event to execute. Alternatively, you keep the event list sorted in an ascending order according to the event arrival times, and simply select the one at the top of the event list.
b. Advance the simulation to time t, i.e., set MCL=t, and execute the logic associated with the event, and all other events that may get triggered. Populate the event list accordingly.
c. Check stoppage rule and if you are to continue go back to a.

III. Repetition i, i=2,3, … , N

Repeat the above, by zeroing all counters, and starting again from the beginning with the same initial condition, but use a different seed for the pseudo-random number generator.

IV. Statistics calculation

You will estimate the mean and confidence interval and the 95th percentile and confidence interval of:

a. The time *T* it takes to certificate an IoT device, and
b. The time *R* it takes a new request to enter the server's queue if it is rejected the first time.

Also, you will estimate the mean and its confidence interval (no percentile estimation) of

c. The total time *P* it takes to process all 1000 IoT devices.

To estimate *T* you need to enhance your simulation so that to keep track of the elapsed time from the moment a request enters the server's queue to the moment it is processed and departs from the server (the response message sent back to the IoT device is not simulated.) For the second statistic *R*, augment the event list to add the time that a new request arriving at the buffer is rejected, so that to calculate the time elapsed until the request enters the buffer. Finally, for the total time it takes to process all IoT devices, simply record the value of the master clock MCL when the last IoT device is served.

a. Calculate the mean *T* and *R* from each iteration. Then, calculate the super mean and standard deviation and construct the confidence interval.

b. Calculate the mean and confidence interval of the total time it takes to process all IoT devices *P* based on the *N* values you obtain from the *N* iterations.

c. The 99th percentile and its confidence interval of *T* and *R* will be calculated using the following procedure. Given a random variable, the 95th percentile of the random variable is a value such that only 5% of the values of the random variable are greater than itself. Let x_i be the *T* (or *R*) estimate in the *i*-th repetition. Then the 95th percentile of *T* (or *R*) is a value such that Prob[$X \leq T$] = 0.95. Now, suppose that the total number of observations is *N*, i.e., $x_1, x_2, \ldots, x_N$. To calculate the percentile, you have to sort them out in ascending order. Let $y_1 \leq y_2 \leq \ldots \leq y_N$ be the sorted observations. Then, the 95th percentile *T* (or *R*) is the value y_k where $k=ceiling(0.95N)$, where *ceiling(x)* is the ceiling function that maps the real number *x* to the smallest integer not less than *x*. For instance, if $N = 50$, then $k = 48$, and the percentile is the value y_{48}. Calculate the 99th percentile for each repetition, and then as in a) calculate the confidence interval.

IV. Input data.

Your program should read the following input parameters from a file named "input.txt".

$1/\lambda$	Inter-arrival time
$1/\mu$	Service time
B	Buffer size
N	Number of repetition
D	Max retransmission value

Your program should generate a file named “output.txt” stating the following results:

1. Mean and confidence interval of T, R, and P.
2. 95^{th} percentile of T and R.

Deliverables
The tasks will be defined as we progress through the simulation exercise.