Simulation of a queuing system with bulk service in which the server is able to serve two customers at the same time. Whenever the server completes a service, then serves the next two customers at the same time. However, if there is only one customer in line, then that customer is served alone.

Performance Evaluation of Computer Networks

ECE 541

SANCHO SILGADO, Andrea
CWID: A20315328

Introduction

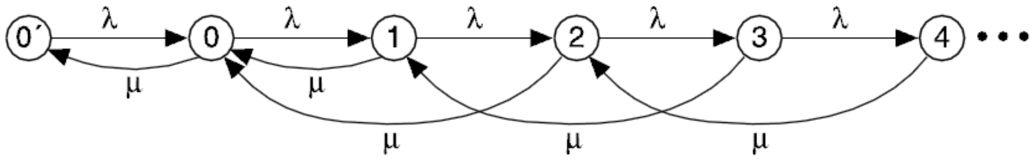
The objective of this project is the simulation of a queuing system with bulk service in which the server is able to serve two customers at the same time. Whenever the server completes a service, then serves the next two customers at the same time. However, if there is only one customer in line, then that customer is served alone.

The model consists of a queue with infinite capacity and a batch server. The server has a maximum service capacity of2. Customers finding the server busy upon their arrival and customers that are waiting for a batch to form are queued in FIFO order.

The required parameters that should be provided are:

- Service time is exponential at rate μ whether serving one or two customers.
- Customers arrive at an exponential rate λ.

Given these rates, we need to create two exponential random variables. For the implementation of the simulation I will use those random variables in time units, so I will have: the meanArrivalTime (time units per arrival) and the meanServiceTime (time units per service) as parameters of each exponential random variable.



In order to define the states of the system we look at the number of customers waiting in queue. We will need to be able to look at the length of the waiting queue and if the server is or not busy in order to decide which is the next state in the time line of the simulation.

Also we should be able to choose the number of customers served in total which is used to determine when the simulation should stop. An optional implementation can be provided if we rather determine the end of the simulation by defining a simulation run length time.

The requested results are the average system response time and the average waiting time in the queue. For the mathematical calculations we will need to use Little's formula:

$$L = \lambda W \qquad W_Q = L_Q/\lambda, \qquad W = W_Q + (1/\mu),$$

Where:
L, the average number of customers in the system;
L_Q, the average number of customers waiting in queue;
W, the average amount of time a customer spends in the system;
W_Q, the average amount of time a customer spends waiting in queue.

Mathematical Model (textbook)

State	*Rate at which the process leaves = rate at which it enters*
$0'$	$\lambda P_{0'} = \mu P_0$
0	$(\lambda + \mu) P_0 = \lambda P_{0'} + \mu P_1 + \mu P_2$
$n, n \geqslant 1$	$(\lambda + \mu) P_n = \lambda P_{n-1} + \mu P_{n+2}$

Now the set of equations

$$(\lambda + \mu) P_n = \lambda P_{n-1} + \mu P_{n+2}, \qquad n = 1, 2, \ldots$$

has a solution of the form

$$P_n = \alpha^n P_0$$

$$\alpha = \frac{\sqrt{1 + 4\lambda/\mu} - 1}{2}$$

$$W_Q = \frac{L_Q}{\lambda},$$

$$W = W_Q + \frac{1}{\mu},$$

$$L = \lambda W$$

$$\begin{aligned} L_Q &= \sum_{n=1}^{\infty} n P_n \\ &= \frac{\lambda(1-\alpha)}{\lambda + \mu(1-\alpha)} \sum_{n=1}^{\infty} n\alpha^n \\ &= \frac{\lambda\alpha}{(1-\alpha)[\lambda + \mu(1-\alpha)]} \end{aligned}$$

Design

I will base the simulation on an *event-scheduling time-advance algorithm,* which basically consists of ordering chronologically all the events in a list. The first element in the list will be the first event to occur: imminent event. After processing this event, a new event (end of activity) will be added to the schedule in the future event list.

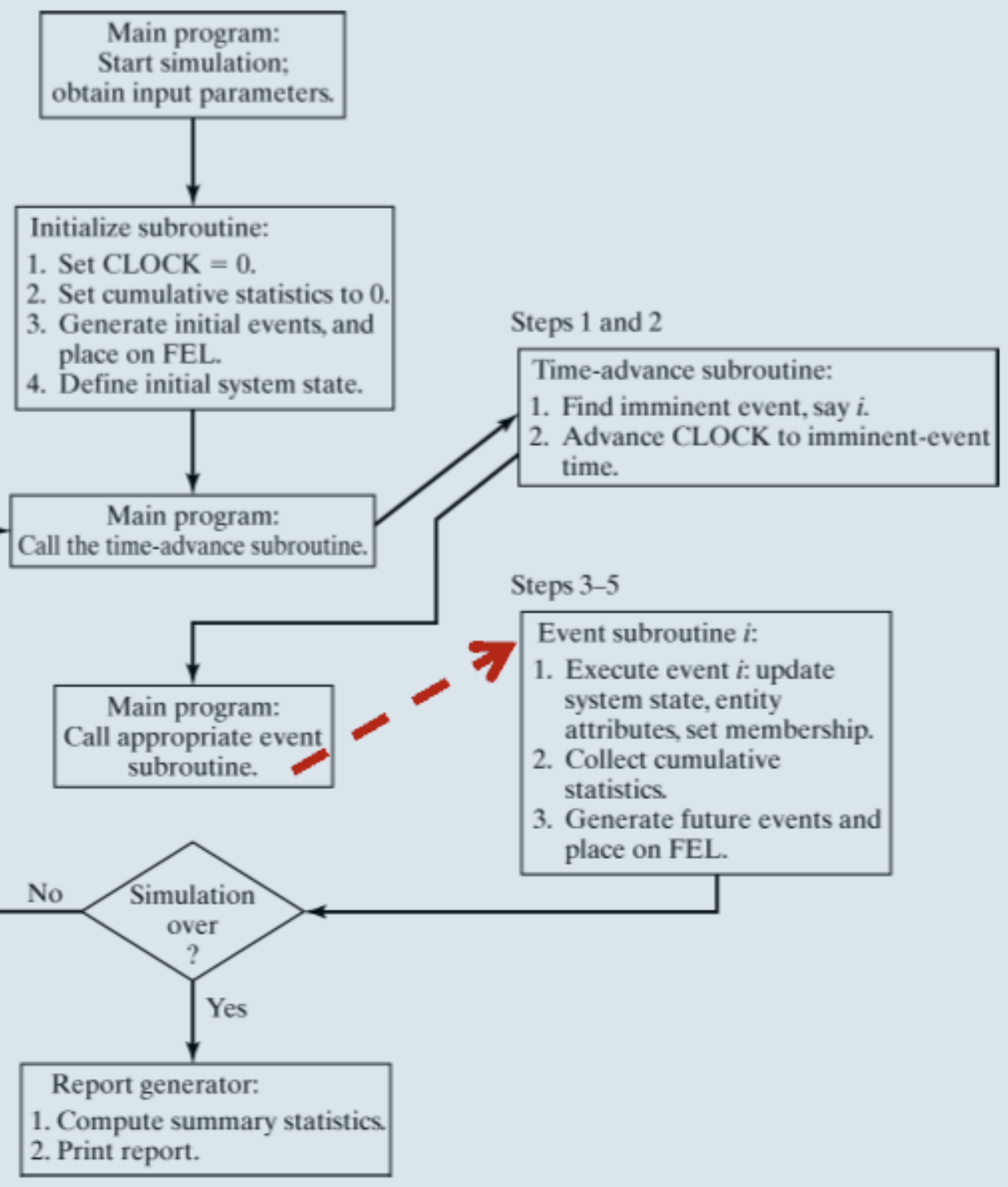
Event Scheduling Time Advance Algorithm

```
1. Initialize all elements in the system, including the system clock
2. Initialize the Future Event List (FEL) with one or more future-event notices
3. While the FEL is nonempty
        (a) Remove the minimum event-notice E from the FEL heap
        (b) Advance the system clock C so that C = t, where t is the time that E occurs
        (c) Process E and update system state. Such updates may include:
            ▪ Updating the state of all objects involved with the event by calling the
              appropriate state-changing object methods
            ▪ Making a server available or busy
            ▪ Removing (adding) objects from (to) queues
            ▪ Cancelling a future event
        (d) Generate any future events that were caused by E and insert them into the FEL
4. Stop the simulation
        (a) At time 0, schedule a stop simulation event at a specified future time
        T → Simulation will run over [0, T]
        (b) Run length T is determined by the simulation itself.
            ▪ T is not known ahead.
            ▪ Example 1: T = When FEL is empty
            ▪ Example 2: T = When k-th customer leaves the system (that is the condition
              used)
```

In order to design the simulation it is necessary to distinguish the different elements involved in it. I have structured them in the following categories to provide an easy understanding of the process:

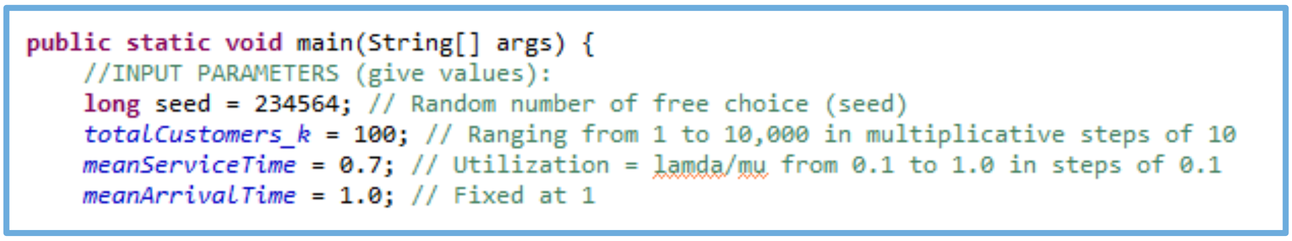
- System state: queueLength, numberInService, totalCustomersServed
- Entity attributes: customersQueue (Events: arrivals and departures)
- Future event list (FEL): futureEventList
- Activity durations: meanArrivalTime, meanServiceTime
- Input parameters: meanArrivalTime, meanServiceTime, totalCustomers (K), seed (for generating random values)
- Simulation variables: clock, lastEventTime, totalBusy, totalServiceTime, totalWaitingTime, maxQueueLength, lastServiceTime, totalDepartures
- Statistics: averageResponseTime, averageWaitingTime, utilization, doubleDepartures, singleDepartures
- Methods: main, initialize, processArrivalEvent, processDepartureEvent, scheduleNextEvent, reportGenerator

Simulation Process Diagram


Main program:
Start simulation;
obtain input parameters.
Initialize subroutine:
1. Set CLOCK = 0.
2. Set cumulative statistics to 0.
3. Generate initial events, and place on FEL.
4. Define initial system state.
Main program:
Call the time-advance subroutine.
Steps 1 and 2
Time-advance subroutine:
1. Find imminent event, say i.
2. Advance CLOCK to imminent-event time.
Steps 3–5
Event subroutine i:
1. Execute event i: update system state, entity attributes, set membership.
2. Collect cumulative statistics.
3. Generate future events and place on FEL.
Main program:
Call appropriate event subroutine.
No
Simulation over ?
Yes
Report generator:
1. Compute summary statistics.
2. Print report.

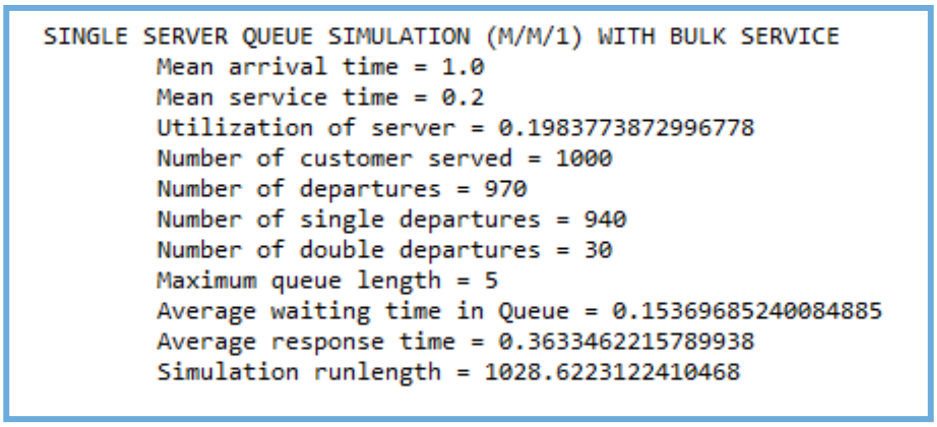
To see how the program works, **documentation is provided in the code**. To simplify things, all variables and methods implemented have a name that reveals itself the purpose of the element.

In order to run the simulation with a set of input values, we have to modify the values inside the main() method in class Simulation. Here is an example:



```
public static void main(String[] args) {
    //INPUT PARAMETERS (give values):
    long seed = 234564; // Random number of free choice (seed)
    totalCustomers_k = 100; // Ranging from 1 to 10,000 in multiplicative steps of 10
    meanServiceTime = 0.7; // Utilization = lamda/mu from 0.1 to 1.0 in steps of 0.1
    meanArrivalTime = 1.0; // Fixed at 1
```

With that, the output of the simulation is shown as:



```
SINGLE SERVER QUEUE SIMULATION (M/M/1) WITH BULK SERVICE
        Mean arrival time = 1.0
        Mean service time = 0.2
        Utilization of server = 0.1983773872996778
        Number of customer served = 1000
        Number of departures = 970
        Number of single departures = 940
        Number of double departures = 30
        Maximum queue length = 5
        Average waiting time in Queue = 0.15369685240084885
        Average response time = 0.3633462215789938
        Simulation runlength = 1028.6223122410468
```

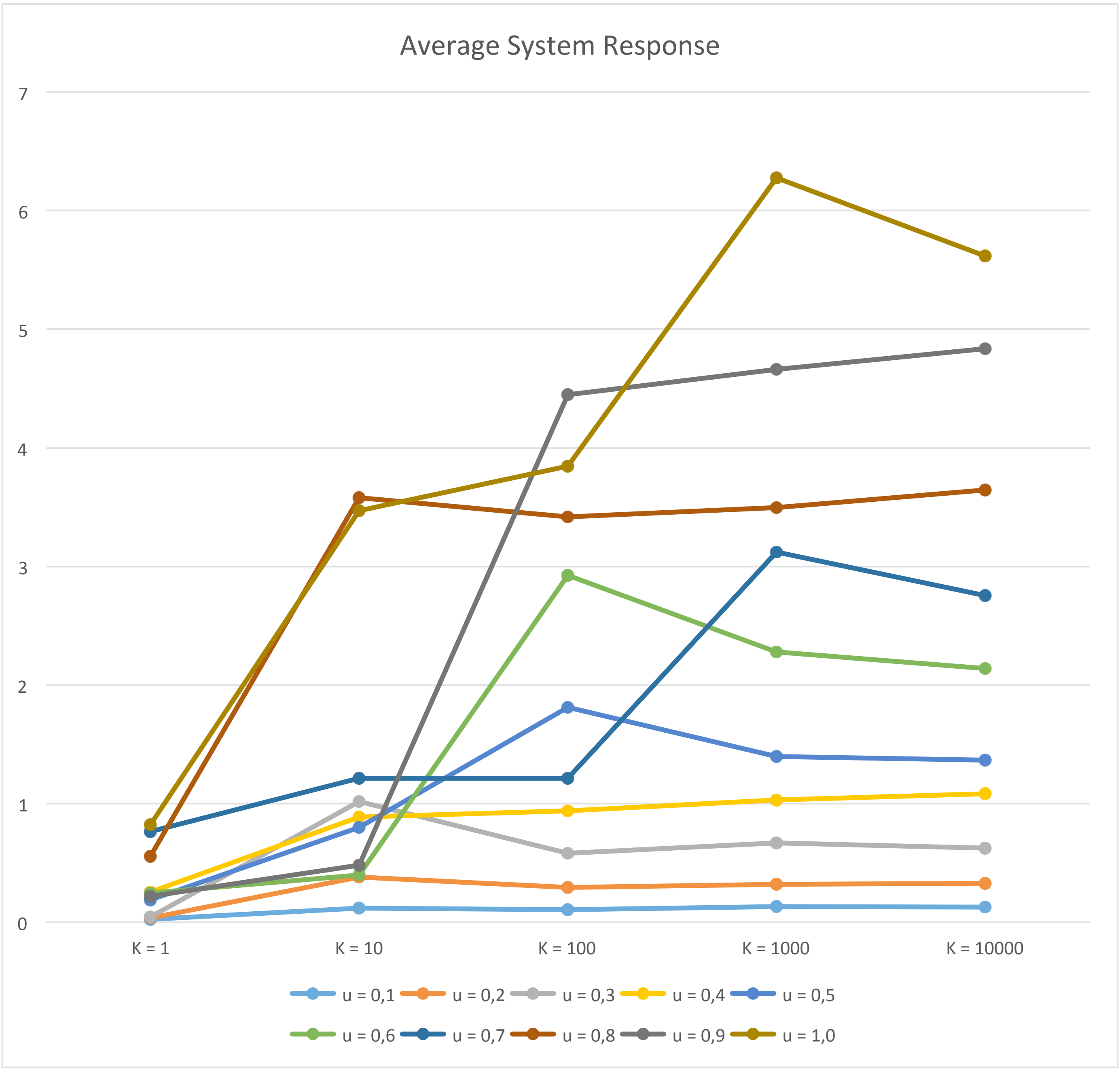
Testing and results

As I used seeds to implement the random variables of arrival and service times, for each simulation with fixed values of *totalCustomers_K*, *meanServiceTime* and *meanArrivalTime*, if we use the same *seed*, we will always get the same results.

Average system response:

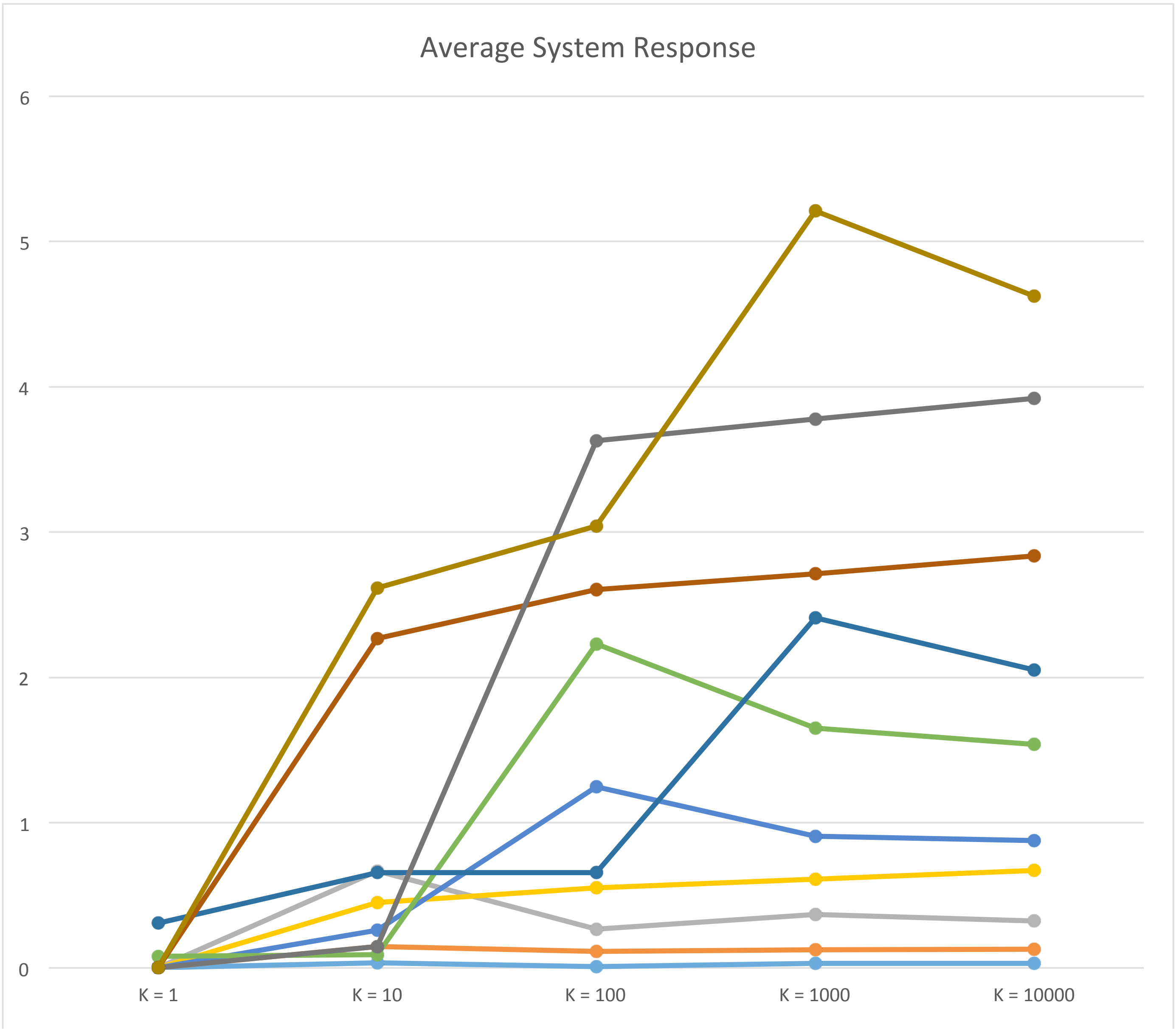
totalCustomers_K / *Utilization*	1	10	100	1,000	10,000
0.1	0.0254 (seed:234564)	0.1179 (seed:213)	0.1051 (seed:423003)	0.131214 (seed:346280)	0.128764 (seed:13)
0.2	0.034561 (seed:1713)	0.38024 (seed:456)	0.2958 (seed:9001)	0.319243 (seed:451)	0.3272466 (seed:30178)
0.3	0.044858 (seed:3462)	1.018664 (seed:221)	0.58282 (seed:67892)	0.6705968 (seed:7462)	0.6249 (seed:541)
0.4	0.25559 (seed:16)	0.888698 (seed:4673)	0.941645 (seed:8743)	1.0293553 (seed:830)	1.083292 (seed:4454)
0.5	0.189979 (seed:56705)	0.79977 (seed:905)	1.814438 (seed:2356)	1.397624 (seed:8451)	1.367638 (seed:783)
0.6	0.247238 (seed:8973)	0.399798 (seed:73)	2.92422 (seed:987)	2.2791389 (seed:6547)	2.137459 (seed:3216)

0.7	0.76312276 (seed:36900)	1.21518 (seed:258963)	1.21518 (seed:124578)	3.120653 (seed:3791)	2.752707 (seed:900)
0.8	0.55759 (seed:258)	3.5805216 (seed:4862)	3.41985257 (seed:41222)	3.497225 (seed:6112)	3.64604286 (seed:603)
0.9	0.218597 (seed:3000)	0.4797328 (seed:45)	4.44927 (seed:8595)	4.65988 (seed:5312)	4.8347 (seed:995)
1.0	0.8237641 (seed:987677)	3.471712 (seed:8410)	3.8455017 (seed:32147)	6.27778 (seed:12369)	5.617439 (seed:840)



Average waiting time in queue:

totalCustomers_K ***Utilization***	**1**	**10**	**100**	**1,000**	**10,000**
0.1	1.040E-17 (seed:234564)	0.0350 (seed:213)	0.00963 (seed:423003)	0.031472 (seed:346280)	0.03099 (seed:13)
0.2	2.081668E-17 (seed:1713)	0.14579 (seed:456)	0.115 (seed:9001)	0.125687 (seed:451)	0.13033958 (seed:30178)
0.3	1.387778E-17 (seed:3462)	0.666 (seed:221)	0.2679 (seed:67892)	0.36938 (seed:7462)	0.322847 (seed:541)
0.4	5.551115E-17 (seed:16)	0.451714 (seed:4673)	0.55264 (seed:8743)	0.611503 (seed:830)	0.670787 (seed:4454)
0.5	2.775557E-17 (seed:56705)	0.2576599 (seed:905)	1.24565 (seed:2356)	0.906387 (seed:8451)	0.87831 (seed:783)
0.6	0.0791508 (seed:8973)	0.0894668 (seed:73)	2.2321 (seed:987)	1.652414 (seed:6547)	1.537843 (seed:3216)
0.7	0.306806 (seed:36900)	0.657106 (seed:258963)	0.657106 (seed:124578)	2.409364 (seed:3791)	2.050986 (seed:900)
0.8	0.0 (seed:258)	2.2661219 (seed:4862)	2.602653 (seed:41222)	2.71385 (seed:6112)	2.8348317 (seed:603)
0.9	0.0 (seed:3000)	0.146838 (seed:45)	3.630611 (seed:8595)	3.780086 (seed:5312)	3.92215 (seed:995)
1.0	2.220446E-16 (seed:987677)	2.6137707 (seed:8410)	3.0427485 (seed:32147)	5.21002 (seed:12369)	4.625172 (seed:840)



To fill the tables I run the simulation 50 times, each time changing the value of the seed to have different random value generators.

Explanation and Analysis of results

We can see that for high values of customers in system (K), we obtain that both the average system response and the average waiting time in queue tend to increase as the utilization of the system increases. This is due to the fact that increasing the number of customers in system, we are increasing the simulation run length time, so we get closer to the ideal case (the long term), improving the accuracy of the obtained results.

To better trust the obtained values of simulations, we should have run more simulations for each value of the number of customers in system and the utilization (using different seeds).

Also, to apply the mathematical model, we need a fixed value of mean service time. In the required results we are asked to provide results for several values of utilization (implying different service times) without deriving the required quantities from the mathematical expressions which, I think, were successfully obtained.

Among the provided results, there are more than the required ones. At the end of each simulation we can obtain the maximum length of the queue reached, the number of single and double departures of customers, the simulation run length time…

Conclusions

The simulation runs suitably to the described model, in spite of the fact that for some seed values we can eventually return incoherent values, that is due to the randomness of the rng (random number generator) provided in Java.

It is interesting to see how the events arrive to the system (they are generated). This simulation could be used for didactical purposes.