Homework 4

Simulation and Performance Evaluation – University of Trento

You can solve the following assignments using any programming language. In doing so, make sure to implement the formulas explained in class, and try not to use functions made available by the languages to achieve the required tasks. However, you are allowed to use any utility functions you wish.

Exercise 1

In this exercise you will implement a simple discrete-event simulator and you will use it to simulate the performance of an M/M/1 queue/server system, where packets come to the system separated by inter-arrival times that are exponentially distributed with parameter λ, and the service time is exponentially distributed with parameter μ. You may want to follow these steps.

1. Define the following types of events (e.g., through an enumeration)
   (a) Start of the simulation
   (b) End of the simulation
   (c) Arrival of a packet
   (d) Departure of a packet
   (e) A "debug" event

   It is useful if each event is some data structure that you can organize as you wish (a `struct`, a table, whatever you prefer) and that has the following attributes
   (a) The occurrence time of the event
   (b) The event type
   (c) A pointer to the next event
2. Create a queue of events that has the following properties:
   (a) Every event always links to the one that immediately follows it in time;
   (b) When you insert an event in the queue, you always insert it in order of increasing time; (i.e., say that the queue contains three events: event 1 taking place at time t_1 and linking to event 2, which takes place at time t_2 and which links to event 3 at time t_3; if another event 4 taking place at time t_4 is inserted in the queue, and $t_2 < t_4 < t_3$, then you have to make event 2 link to event 4, and event 4 link to event 3 (you can program the above manually, or use any data structure that your programming language makes available in order to do this).
3. Create a skeleton for the initialization of your simulation. For the moment:
   (a) initialize the event queue;
   (b) set the current time to 0;
   (c) set a maximum simulation time (e.g., 1000 s);
   (d) insert the simulation start event and the simulation end events in the queue.

   Run your skeleton and test that all of the above works.
   (*Hint*: remember that, whenever you need to debug something, you may schedule a debug event and have it print some useful information about the system, e.g., the event queue, current metrics, etc. You may also trigger the event upon the occurrence of some other event, e.g., if you observe some suspicious behavior.)
4. Test the insertion of more events in the queue: insert them out of order and check that your code pops them from the queue in order.
5. Create the event manager loop, i.e., the routine that pops the event from the queue and executes the code that manages each event until the simulation ends. For example, you can create a `switch`/`case` block and wrap it into a `while(1)` loop. Test that you correctly loop through all events you inserted, and that the simulation actually stops when you reach the simulation end event.

6. Initialize the remaining system parameters: the status of the server and the packet queue. (*Hint*: for the present exercise, what you really need is only the *number* of packets in the queue.)
7. Implement arrivals: the first arrival event should be scheduled in the simulation start event; then, as seen in class, every time you have an arrival you should immediately schedule the next arrival in order to keep the simulation going. Start with small inter-arrival rates λ for your exponential distribution, so that you can check that everything is fine.
8. Implement the system behavior as seen in class, namely:
   (a) When a packet arrives: if the server is free seize the server and schedule the departure of the packet; if the server is busy, increase the number of packets in queue;
   (b) When a departure event is triggered: if the queue is empty, release the server; otherwise keep the server busy and schedule the next departure event.
9. Use your simulator to show how the number of packets in the system (those in queue plus those currently in service) varies over time. Compare your results with the theoretical average number of packets in the system in stationary conditions, $\rho/(1-\rho)$, where $\rho = \lambda/\mu$.
10. Play with λ and μ, and discuss how their values impact the convergence of the system to the theoretical value.
11. Use your simulator to measure the average time that a packet has to wait in queue, on average, and compare it against the theoretical value, $\rho^2/\big(\lambda(1-\rho)\big)$
12. Tune your parameters λ, μ and the simulation time such that you observe both transient and steady-state system condition in a typical run. Give the empirical distribution of the number of packets in the system and of the queue waiting time at different epochs throughout your simulation. (*Hint*: you will need to run your simulator several times to do this.) Comment on the convergence of the system to steady-state conditions.

Exercise 2

1. Repeat exercise 1 for an M/M/c system where
   - there are c servers (start with $c = 2$, then test for larger values);
   - there exists a single queue, shared among the servers;
   - packets are sent to the first available server; if several servers are free, the packet is sent to the server with the lowest ID (for example, if servers 1 and 2 are free, server 1 gets the packet).
2. Additionally, plot a histogram showing the number of packets served by each of the c servers. Play with the service time (e.g., shorten it significantly). What do you observe? Devise and implement a way to change this result.

Exercise 3

At the beginning of year 2000, the Random Waypoint mobility model was very popular for the simulation of mobile wireless networks. The model assumes that N nodes are deployed in an area of $x_{\max} \times y_{\max}$ square meters. At the beginning of the simulation, the initial node positions are drawn at random. Moreover, each node i draws a destination waypoint at random within the area, and starts moving there at a speed v_i, where v_i is also chosen at random within the interval $[0, v_{\max}]$. Every time a node reaches its waypoint, the node randomly chooses a new waypoint within the area and a new movement velocity uniformly at random in the same interval $[0, v_{\max}]$. Each node repeats this behavior until the end of the simulation.

In this exercise, you will implement and test the random waypoint mobility model.

1. Initialize the simulation. You may want to start with an area of 1000×1000 m^2 and with a number of nodes `nNodes=1` and, say, $v_{\max} = 10$ m/s. For each node, you will need to keep track of
   - its location (in 2D)
   - the time when it reached the last waypoint
   - the next waypoint
   - its current speed

   You need to initialize the above parameters for every node at the beginning of the simulation, e.g., when you handle the simulation start event.
2. Add an event "waypoint reached" to your enumeration of events. You will need to compute the time when each node will reach its current waypoint at the present speed, and schedule a "waypoint reached" event to extract the new waypoint and speed values. Since you have to schedule one different event for each node, you will find it useful to extend the definition of your events to add one additional event property, namely the network node involved in the event.
3. Add another event called "measure speed" and schedule it at fixed intervals throughout the whole simulation. Every time the "measure speed" event is triggered, measure the mean speed of all network nodes. In other words, if node n moves at speed $v_n(t)$ at time t, you need to compute the average $\bar{v}(t) = \big(\sum_{n=1}^{N_{\text{nodes}}} v_n(t)\big)/N_{\text{nodes}}$.
4. Validate that everything works as expected, with different number of nodes, and different values of $v_{\max}$.
5. For $v_{\max} = 10$ m/s, repeat the simulation multiple times (at least 100), and collect all measurements of the average speed of the nodes that you do in every run. Show the average of these means as a function of time. Discuss what you observe.
6. Bound the minimum speed of the nodes away from 0, e.g., by extracting it in the interval $[1, v_{max}]$ instead of $[0, v_{max}]$. Choose $v_{max} = 10$ m/s as before. Repeat the Monte-Carlo set of simulation as in the previous point and discuss the results that you obtain.