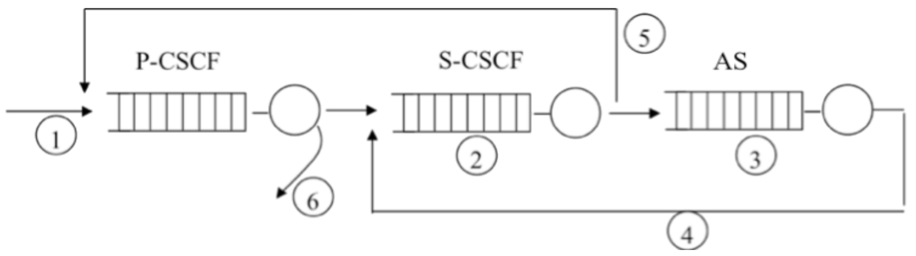
CSC/ECE 576 - Simulation Project:

1. Objectives

In this project, you will develop a simulation model of the flow of SIP messages as they go through a P-CSCF, an S-CSCF, and an application server (AS), with a view to estimating the 95th percentile of the end-to-end delay.

2. Project description

We will consider a P-CSCF, an S-CSCF, and an AS, and an infinite number of UEs. A SIP message generated by a UE arrives at the P-CSCF, gets processed and then relayed to the S-CSCF. The S-CSCF processes the message, and then it sends it to the AS. The AS processes the message and then returns a SIP message to the S-CSCF, which after some processing returns it to the P-CSCF which finally sends it out to the UE. A SIP message is identified with a customer, and the flow of customers through the three servers is depicted by the queueing network shown below.



A customer arrives at the P-CSCF queue (step 1), waits for its turn, gets served and then moves into the S-CSCF queue (step 2). There it waits for its turn and gets processed by the server. Upon service completion, the customer moves to the AS queue (step 3). After the AS processes the request, the customer goes back and joins the S-CSCF queue (step 4), and then after it is processed it goes back and joins the P-CSCF queue (step 5). Upon service completion, it departs from the system (step 6).

The arrival process of customers to the P-CSCF is Poisson distributed with rate λ. This is equivalent to saying that the inter-arrival time between two successive customers is exponentially distributed with mean $1/\lambda$. The processing time at the P-CSCS, S-CSCF and AS is assumed to be exponentially distributed with mean $1/\mu_P$, $1/\mu_S$, *and* $1/\mu_{AS}$ respectively. Customers are processed in each queue in the order in which they come in, that is, first in first out (FIFO).

Tasks

You are required to develop a simulation model, according to the instructions given below, in 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 hand out! Once your program runs correctly, you will use it to calculate the 95th percentile of the *end-to-end delay*, as explained below.

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.

3. The simulation structure

Your program should prompt the user for the following inputs:

1. The rate of the arrivals: λ.
2. The exponential service mean times: $1/\mu_P$, $1/\mu_S$, *and* $1/\mu_{AS}$.
3. The total number of simulated departures.

You program will stop when the number of customers departed from the network reaches the total number of departures you entered.

For each simulated customer, your program should keep track of the total time it spent in the queueing network, which is the end-to-end delay. This is the time elapsing from the moment a customer arrives at the P-CSCF to the moment the same customer departs from the P-CSCF, after it has visited the S-CSCF and AS.

For each run of your simulation, you will divide the total number of arrivals into 30 batches (i.e., groups) of departures with equal size. For each batch, you have to calculate a specified percentile of the end-to-end delay time. Therefore, you will get 30 different percentiles. Based on these 30 percentiles, you will calculate the confidence interval of the percentile of the end-to-end delay time. The details of how to do these calculations are given below later in the statistical analysis section.

Generation of the inter-arrival and service times

In the simulation, you will have to generate random numbers that follow the exponential distribution with a given mean. For this, you will need a good pseudo-random number generator that yields a random number between 0 and 1. Such a generator is readily available in most programming languages. Occasionally, these generators do not have good statistical behavior. Consequently, it may not be a bad idea to download one from a reliable mathematical package, such as Matlab. *Do not make up your own generator!*

Make sure you use one that is known to be reliable. Also, be aware that a random number generator requires an initial value, known as *seed*. Some of the random number generators are automatically seeded. In all your experiments you should always use the same seed, and in view of this, you should use a random number generator that you can seed yourself. (Make sure you read the rules of the random number generator as to what type of number it will accept as a seed).

A random value from an exponential distribution with a given mean can be generated by the expression:

$$-\,(\textit{mean of exponential distribution}) * \log_e(\textit{random}()) \qquad (1)$$

where random() is pseudo-random number. Each time you need a new exponential value you use a new pseudo-random number.

Events

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

1. Arrival at the P-CSCF queue (of a new customer)
2. Completion of the service time at the P-CSCF queue
3. Completion of the service time at the S-CSCF queue
4. Completion of the service time at the AS queue.

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

Start your simulation assuming that all the queues are empty. In order to get the simulation going you need to pre-generate the time the first arrival will occur at the P-CSCF queue. When the first customer arrives it will join the P-CSCF queue and since the system is empty it will start to receive service, and for this, the time at which the service is completed has to be generated. After the customer completes its service it will move to the next queue, and so on. In the meantime other arrivals will start coming into the system and slowly making their way through the three servers.

In order to track the simulation we use a *master clock* (MC). This is just a real variable that gives the current time of the simulation, and the time that each event will be completed in the future. The MC helps us decide which event will be executed next. Let us now take a look at the events and the action to take each time an event occurs.

- *Arrival of a customer to the P-CSCF queue (new arrival):*
  We will assume at the beginning of the simulation that the first (new) arrival

occurs at time MC=0.05. The first task we have to do is to decide when the next arrival will occur. For this we have to generate an exponential time *t* of the inter-arrival time with a mean $1/\lambda$. Then, the time this event will occur in the future is MC+*t*, where MC is the current clock time. Store this time is a list. When this time is reached, the arrival will occur.

The new customer that arrives at the P-CSCF joins the queue. If the queue is empty and the server is idle, it goes into service and we generate a service time *t* using expression (1) with a mean $1/\mu_P$. The service completion time is MC+*t,* where MC is the current time. If the server is busy, then the customer will join the queue and no further action will be taken.

- *Service completion at the P-CSCF*

  If it is a new customer, then it moves on to the S-CSCF queue. If the customer has already been in the system and it has come back from the S-CSCF, then it leaves the system.

  If it moves to the S-CSCF queue, then we follow the same logic as above. That is, if S-CSCF server is idle and there is no one in the queue, then the customer goes into service and we generate a service time *t* using expression (1) with a mean $1/\mu_P$. The service completion time is MC+*t,* where MC is the current time. If the server is busy, then the customer will join the queue and no further action will be taken.

  Also, if there are customers waiting in P-CSCF queue, then the next customer will start its service and for this we have to generate a new service time *t* at queue *i* using expression (1) with a mean $1/\mu_P$. The service completion time is MC+*t,* where MC is the current time.

- *Service completion at the S-CSCF*

  If the customer has arrived from the P-CSCF, then it moves on to the AS queue. Otherwise, it moves to the P-CSCF queue. The rest of the logic is the same. That is, if the customer finds the P-CSCF or AS server idle (i.e., it is not busy and no one is waiting in the queue), then it starts a new service. Otherwise, it just joins the queue. If there are customers waiting in the S-CSCF queue, then the next in line starts service.

- *Service completion at the AS*

  The customer moves to the S-CSCF queue. If it finds the S-CSCF queue empty, then it starts a new service. Otherwise, it just joins the queue. If there are customers waiting in the AS queue, then the next in line starts service.

Simulation logic

The main simulation logic is very simple. After completing servicing an event, check all the future events to see which of them will occur next. Advance the clock to that time and service the event according to the logic described above.

The total number of events is 4, namely, new arrival to P-CSCF, and service completion at P-CSCF, S-CSCF, and AS. For better tracking these events, you could place them into an array, and associate each event with the time that it is scheduled to occur in the future, and also where to jump to in the code to service this event. Events that have not been scheduled should be associated with a time equal to a very large value, so that they will never occur.

Also, you should associate a data structure (could be an entry in an array for simplicity) for each customer. In it you can keep the time of arrival, and also a flag associated with whether it is a new customer (i.e. moving from left to right) or a re-circulating customer going backwards

4. Statistical analysis

When a customer departs from the system (i.e. from the P-CSCF after it has been to the S-CSCF and AS), calculate the time it spent in the entire queueing network and store it in an array for further processing. This is the end-to-end delay of the customer.

The 95th percentile of the end-to-end delay time.

Based on the observations stored in the array you will calculate the 95th percentile as follows. Given a random variable with a probability distribution, the 95th percentile is the value of this random variable such that only 5% values of this random variable are greater than itself. Let X indicate the end-to-end delay time in the queueing network, and let x_i be the end-to-end delay time of i-th customer. Then the 95th percentile of the end-to-end delay time T 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 T, 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 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}.

The confidence interval of the 95th percentile of the end-to-end delay time.

A confidence interval provides an indication of the error associated with our estimate, which in this case is the 95th percentile. To calculate the confidence interval you will have to obtain about 30 different percentiles of the end-to-end delay time for the same input values. An easy way to do this is to use the *batch method*. Run your simulation for a large number of departures, and then group your results as follows. Let us say that you run the simulation for 9,000 departures, and that you divide all departures into batches of 300 arrivals. Therefore, there are 30 batches. For the first 300 departures in batch 1, you calculate the 95th percentile of the end-to-end delay time and store this number in an array. Repeat this process for the next 300 arrivals without changing anything in your simulation, and so on until you have obtained 30 different percentiles. Now, let T_1, T_2, T_3 ... T_n be the calculated percentiles for n=30 batches. Then, the mean of the percentiles is:

$$T_{mean} = \frac{1}{n}\sum_{i=1}^{n} T_i ,$$

and the standard deviation s is:

$$s = \sqrt{\frac{\sum_{i=1}^{n}(T_{mean} - T_i)^2}{n-1}}$$

The confidence interval at 95% confidence is given by:

$$(T_{mean} - 1.96\frac{s}{\sqrt{n}}, T_{mean} + 1.96\frac{s}{\sqrt{n}})$$

The confidence interval tells us that the true 95th percentile lies within the interval 95% of the time. That is, if we repeat the above simulation 100 times, 95% of these times, on the average, the true 95th percentile will be within the interval. Accurate simulation results require that the confidence interval is very small. That is, the error $1.96(s/\sqrt{n}) \sim 0.10T_{mean}$. If the error is not small enough, increase the batch size from 300 arrivals to 350, and run the simulation again for 30 batches. Keep increasing the batch size until you are satisfied that the error is within the limits.

Initial condition

When you start the simulation, you will assume that there are no customers in queueing network, i.e. the system is empty. The initial behavior of the simulation will be affected by this "system empty" initial condition. In order to eliminate the effects of the initial condition, you should run the simulation for 100 departures first. After that, start the batch method described above.

5. Deliverables

Task 1: Percentile Calculation

Run your simulation to obtain the specified percentile of the end-to-end delay of the customers and its confidence interval. Following are the input values:

Inputs	**Value**
Arrival rate λ	
Mean service time $1/\mu_P$	0.1 sec

Mean service time $1/\mu_S$	0.2 sec
Mean service time $1/\mu_{AS}$	0.5 sec
Total number of departures	30100
The number of batches	30

Your program should read these input parameters from a file named “input.txt”. Your program should generate a file named “output.txt” stating the following results:

1) Simple mean of end-to-end delay
2) 95th percentile without using batch means
3) T_{mean} and error percentage of T_{mean}
4) Confidence interval of T_{mean} as [lower limit, upper limit]

Task 2: End-to-end delay as function of λ.

Run your simulation for different values of the λ and plot your results in a graph. Comment on your results.