Project: Simulation of a pair of Tandem Queues using Event Stack

Wahid Uz Zaman (wzz5219@psu.edu)

Project Description:

Implementation:

In this project, I implemented an `EventStack` (a doubly linked list) in `eventstack.py` and then used it to simulate a network of `TandemQueues` in `tandemqueuesimulator.py`. The `TandemQueueSimulator` is designed to handle any number of queues in a tandem network. To set up the simulation, the user provides inter-arrival time distribution (type, params) for the first queue and service-time distribution for each queue. Any sequential tandem queue network (with two or more queues) should satisfy Jackson's theorem for performance metrics for each queue, assuming each queue follows an M/M/1 structure. For stability, the arrival rate must be smaller than the service rates.

To verify the tandem queue simulation (with two queues, each following M/M/1), I checked Jackson's theorem metrics—mean number of jobs, utilization, and mean sojourn time (I used Little's Law)—for each queue. I also used Little's Law for mean sojourn time in the overall system.

Additionally, I tested with a non-Poisson (uniform distribution [0,1]) arrival for the first queue and calculated the performance metrics.

In `main.py`, I ran simulations with poisson arrival (rate = 2) and non-poisson(uniform[0,1] renewal process. I also used 3 different service distributions (all exponential) for both. I checked 5 metrics.

1. Mean sojourn time in overall system
2. Mean number of jobs in the system
3. System throughput
4. Queue utilization (for each queue)
5. Mean number of jobs in each queue

In the results sections, I showed the results using plots.

Generalization:

My current `TandemQueueSimulator` can already handle any number of tandem queues. I have already used a generic method **generate_times_from_distribution(num_samples, distribution)** to simulate user-specified renewal arrivals. Currently, it supports Uniform (0,1), exponential, Erlang, hyperexponential, and hypoexponential distribution. So, this can be easily extended for other distributions as well because, in this method, we used uniform distribution to model other distributions using **inverse transform sampling.**

To extend this to a general queuing network, like a Jackson network, I would add support for transition probabilities between queues. The user could provide a transition probability matrix, where each element defines the probability of moving from one queue to another. With this setup, upon a queue departure, the job may leave the network entirely or (with a certain probability) transition to another queue based on the probabilities in the matrix.

Alternatively, the user could provide a graph-like structure where directed edges represent transitions between queues and edge weights indicate the transition probabilities. In the departure event handling, I would adjust the code to determine the next queue based on the defined probability distribution. As in tandem queues, the probability is 1 from the previous queue to the next queue in the chain; I didn't make the current code generic to work with probabilistic transitions. But as I said, it can be easily modified to support that.

<u>Consideration of Special Case:</u>

Pair of tandem queues in which the first queue can only have an external arrival rate with Poisson arrival and both the service rates follow exponential distribution; this setup is also treated as a Jackson network. Our simulation results nearly match Jackson formulas.

In Jackson's theorem, the mean sojourn time in each queue and in the overall system can be derived using Little’s law. We can calculate the mean number of jobs in each queue using the Jackson formula, and then using Littles law, we can get the mean sojourn time of each queue by dividing it by the initial arrival rate. One thing to be noted here is that for this pair of tandem queues, the 2nd queue’s arrival rate is equal to the 1st queue’s arrival rate [by Jackson Formula]. Also, we can determine mean sojourn time in the overall system using Little’s law. After getting the mean number of jobs for each queue, we can get the mean sojourn time for the overall system [= Sum(avg_job_in each queue) / initial arrival rate]

<u>Network Partition and Each partition has its own event stack:</u>

If the network is divided into partitions, each partition should have its event stack. When events are processed independently within each partition, inter-partition dependencies can arise. This happens when an event in one partition must be processed before certain events in another partition, leading to potential out-of-order processing. To address this, we can use a rollback strategy.

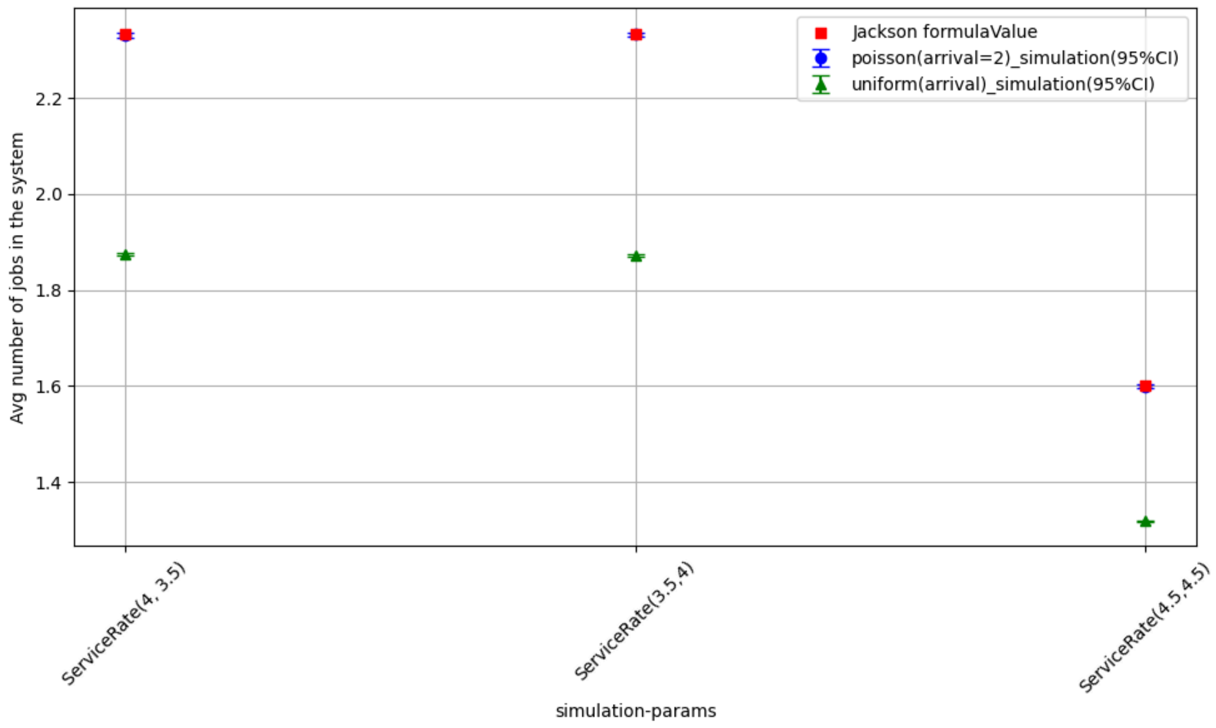
Rollback: If an out-of-sequence event is detected, the affected stack could roll back any events processed after the point in time where the out-of-sequence event was supposed to occur. This rollback would involve undoing the effects of those subsequent events and restoring the state to its original condition. The out-of-sequence event is then processed, followed by reprocessing the rolled-back events. This rollback scheme can be added on top of my code easily. What I can do is I won't pop out events fully from the event stack, but i will keep a pointer to direct current events in the event stack. When an event is pushed from another event stack and the `current` pointer is above that, in that case, I will revert all the events and push down the `current ` pointer to the newly placed event. And then I can process that and reprocess all the reverted events. In this case, the event stack may grow quickly. So, we should determine certain times for all the partitions in which we can be sure no out-of-order will happen, and when that time crosses, we will pop all the events from the stacks that were processed before that time.

Simulation Results:

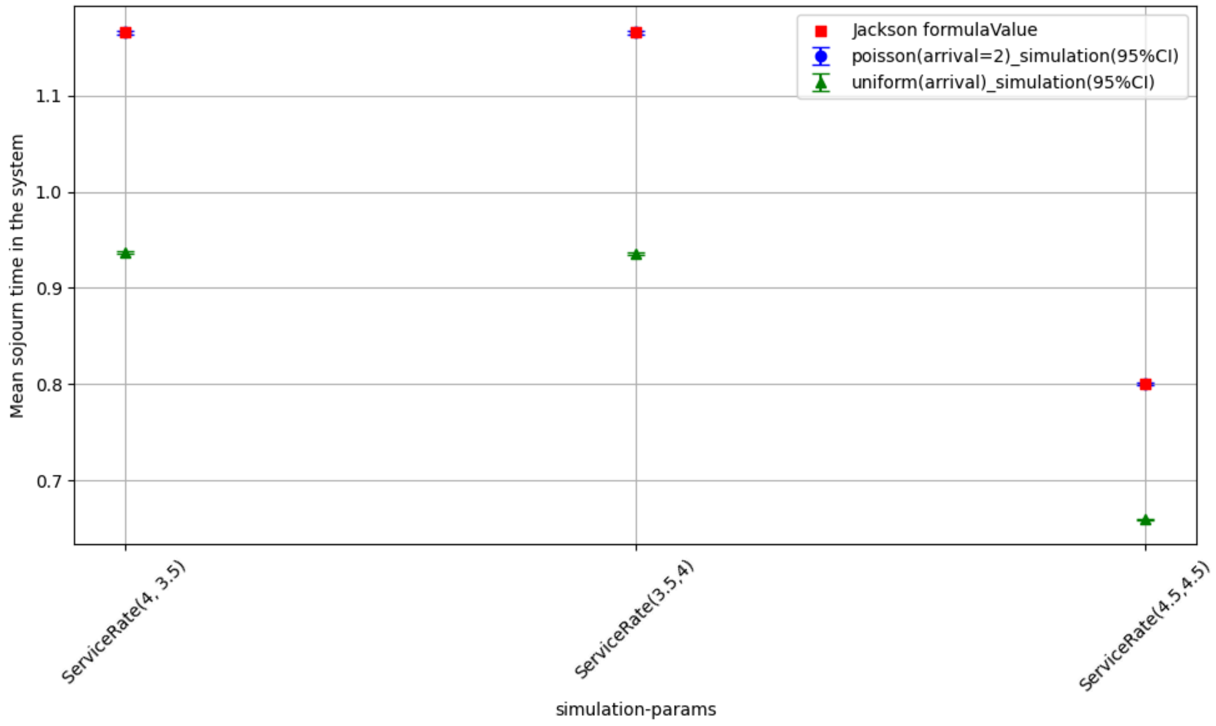
I created a `main.py` script to run various simulations with different configurations. For each run of the simulation, I used 500000 as the number of jobs. Also, I ran 20 simulations for each performance metric and then used mean values with 95% confidence interval in the plot.

I evaluated 5 metrics for these simulations. I used 3 different service rates with poisson and non-poisson(uniform) arrivals. Below, I show the plots for each of these metrics.

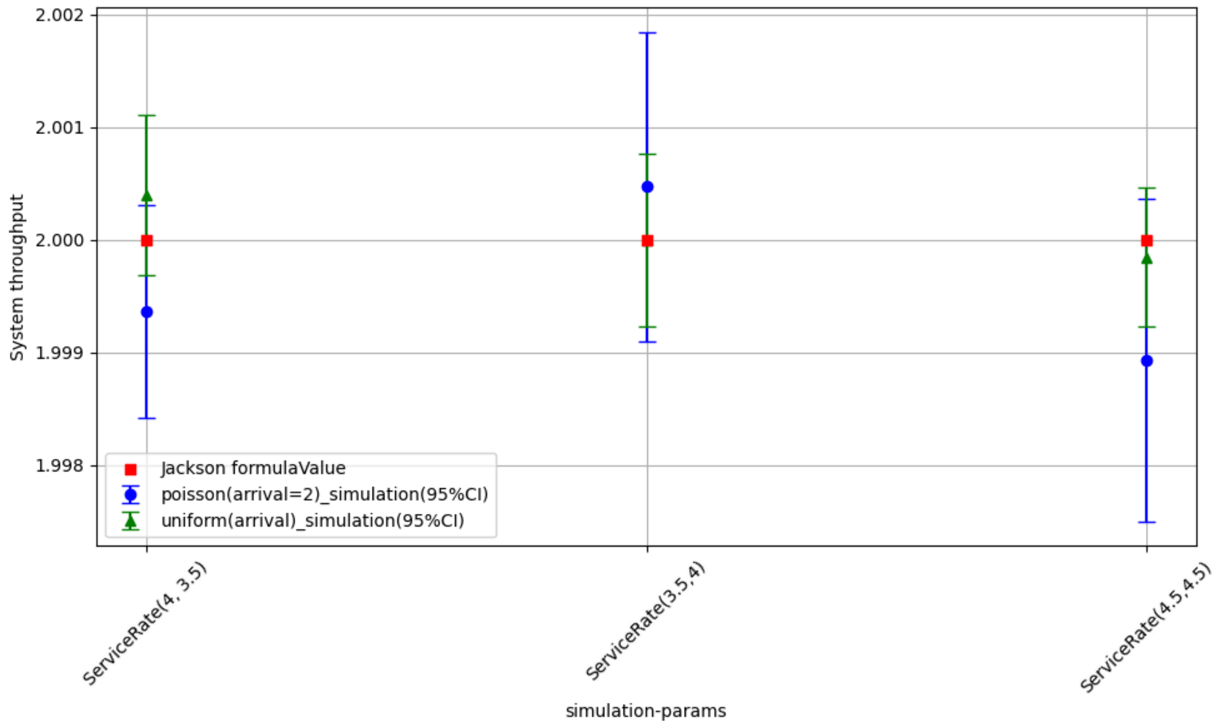
AVG number of jobs in the system



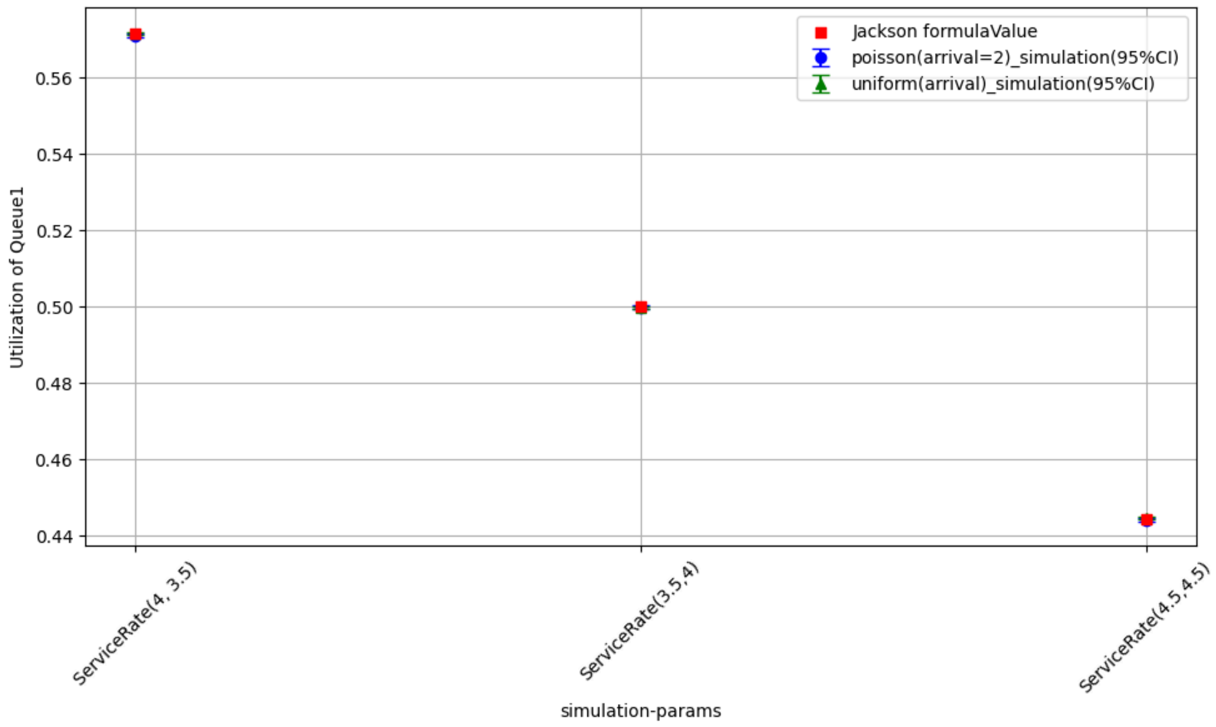
Mean Sojourn time in the system

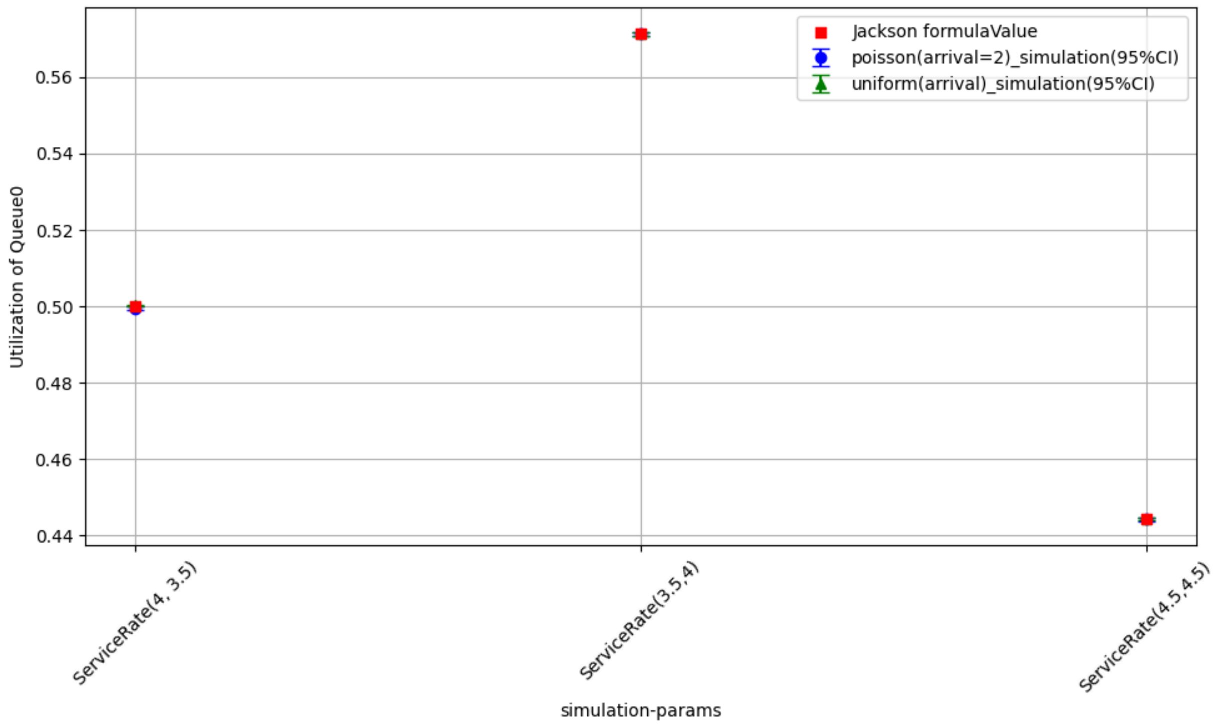


System Throughput:

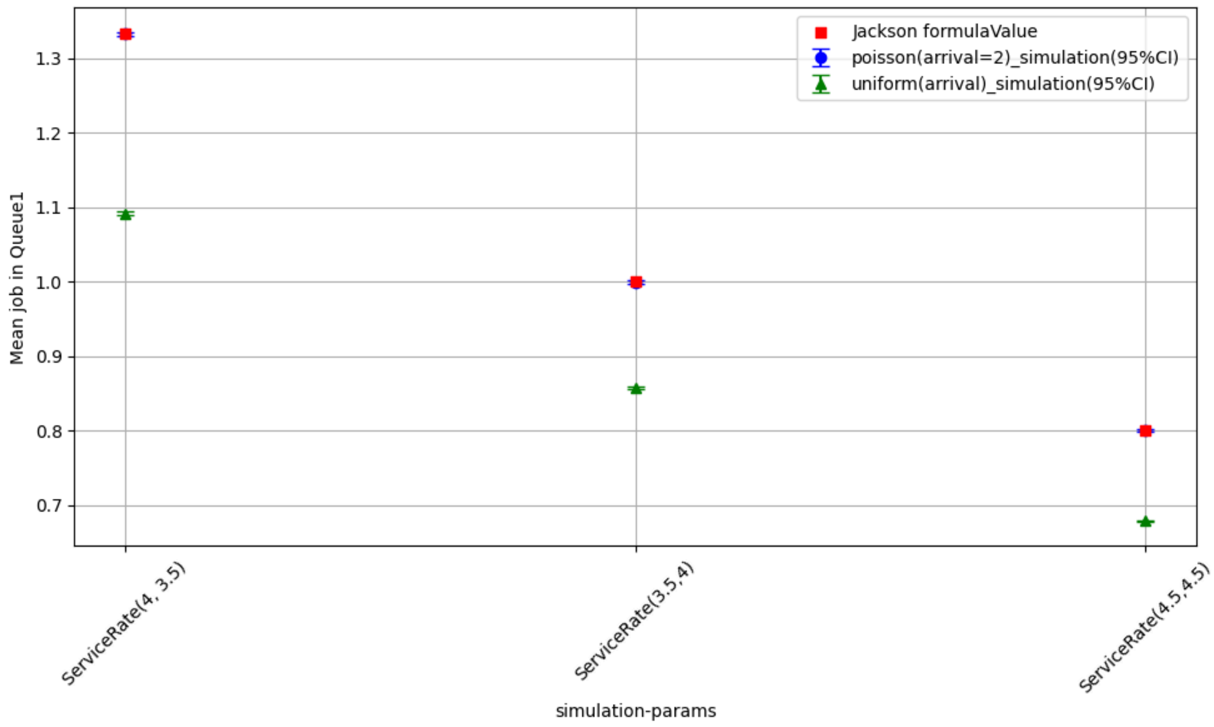


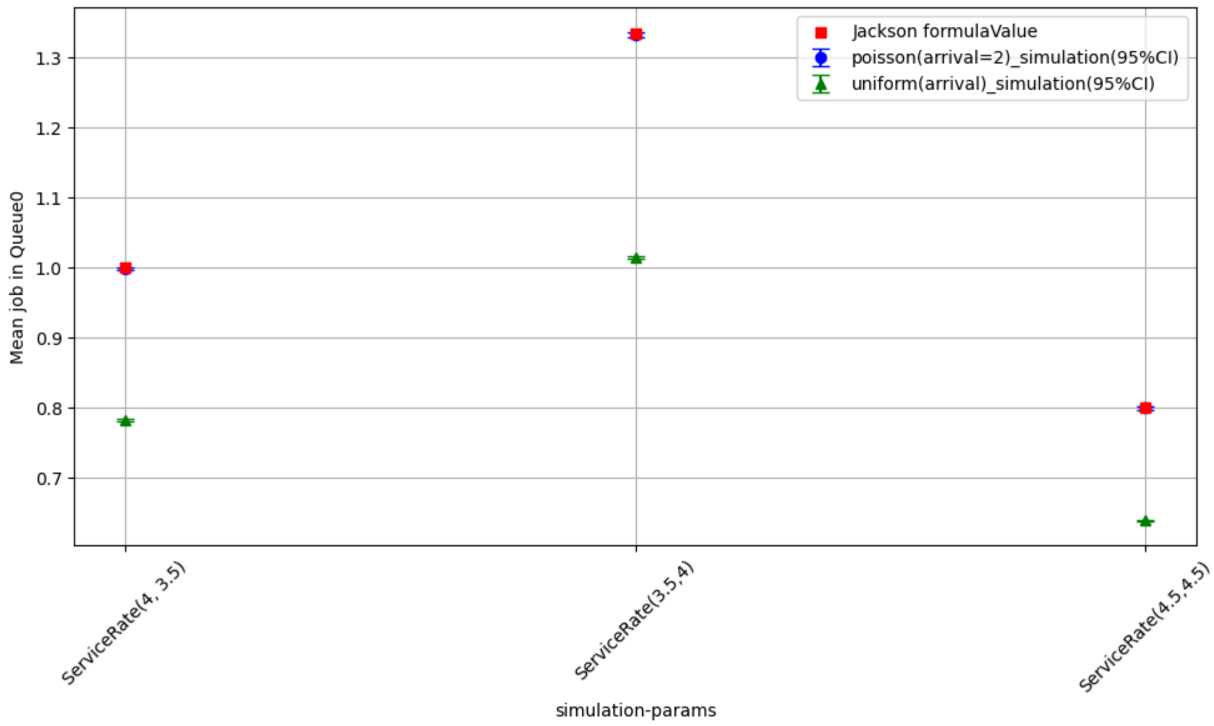
Utilization of Queues:





AVG number of jobs in independent Queues:


Jackson formulaValue
poisson(arrival=2)_simulation(95%CI)
uniform(arrival)_simulation(95%CI)
Mean job in Queue1
ServiceRate(4, 3.5)
ServiceRate(3.5,4)
ServiceRate(4.5,4.5)
simulation-params


Jackson formulaValue
poisson(arrival=2)_simulation(95%CI)
uniform(arrival)_simulation(95%CI)
Mean job in Queue0
ServiceRate(4, 3.5)
ServiceRate(3.5,4)
ServiceRate(4.5,4.5)
simulation-params