1 An M/M/1 queue-server system simulator

In this assignment, we have been asked to implement an M/M/1 queue-server system in order to run some simulations.

We have implemented an event-driven simulator where it is possible to specify the arrival rate of packets in the system (λ), the service rate of them (μ) and the duration of the simulation (*simulation_time*).
The details of the implementation are available in the file `simulator.py`.

The queue of scheduled events is managed via a priority queue, implemented with a heap.
When the "end" event is processed (at time *simulation_time*), we stop scheduling arrival events and we start processing the packets waiting in the queue to be served, if there are any.

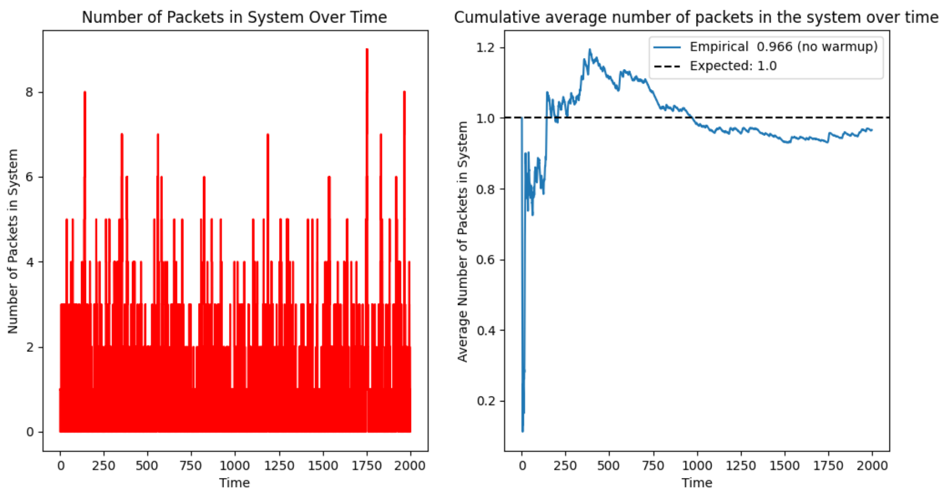
Our class also implements some statistics and empirical measurements about the simulation (we also provide the theoretical values, given the utilization rate of the system $\rho = \frac{\lambda}{\mu}$):

- average number of packets in the queue ($\frac{\rho^2}{1-\rho}$)
- average number of packets in the system (queue + server, $\frac{\rho}{1-\rho}$)
- average time spent by a packet in the system (average traversal time: difference from departure time and arrival time of the packet, $\frac{1}{\mu-\lambda}$)

In the next sections, we show the results we got from running the simulator several times by taking advantage of several techniques to get robust observation and comparable results.

1.1 Running the simulator once

To begin with, we have run **once** the simulator with parameters $\lambda = 1$ and $\mu = 2$, for a simulation time of 2000 and plot two different quantities: number of packets in system over time (very noisy and not explanatory of the system's behavior) and cumulated average of packets.



Proceeding in this way is **not right**: by running the simulator only once, our results are affected by the variability of the sampling and we cannot properly analyze the behavior of the system and the convergence to theoretical values. Running another time the simulator with the same parameters could likely result in getting a quite different plot for the cumulative metric. For that reason, we moved to another simulation setting.

The script used to run the simulation is `comparison_with_theoretical_results_single_run.py`.

1.2 Running the simulator multiple times

To analyze our system and the convergence to theoretical values, we have applied two typical techniques done in output analysis:

- **independent replications**: it consists in running the simulator several times with the same parameters and then average out the observations;
- **warm-up elimination**: this is more particular and related to the typical parameters we choose for the simulation and consists in eliminating some packets at the beginning of the simulation.

Other parameters influencing the simulations were:

- λ and μ: different values of arrival and service rate bring different utilization of the system ($\rho = \frac{\lambda}{\mu}$). We have seen that this ratio has a huge impact on the convergence capabilities of the system;
- *simulation_time*: the higher it is, the easier is to have a convergence of the system to a steady state. With low values of the parameters, it should be even bigger, since we will have small number of packets interacting with the system;
- *warm-up elimination*: to reduce *simulation_time* and help the system converge faster to theoretical values is important to detect visually, in the plot of the *cumulative average of packets in the system over time*, the period of instability of the system and do not use in the statistic computation those first packets. In this way we will consider only the last packets, with more stable statistics.

We have investigated the power of the warm-up and the proper setting of the *simulation_time* in four different conditions: high/low ρ and high/low λ and μ.

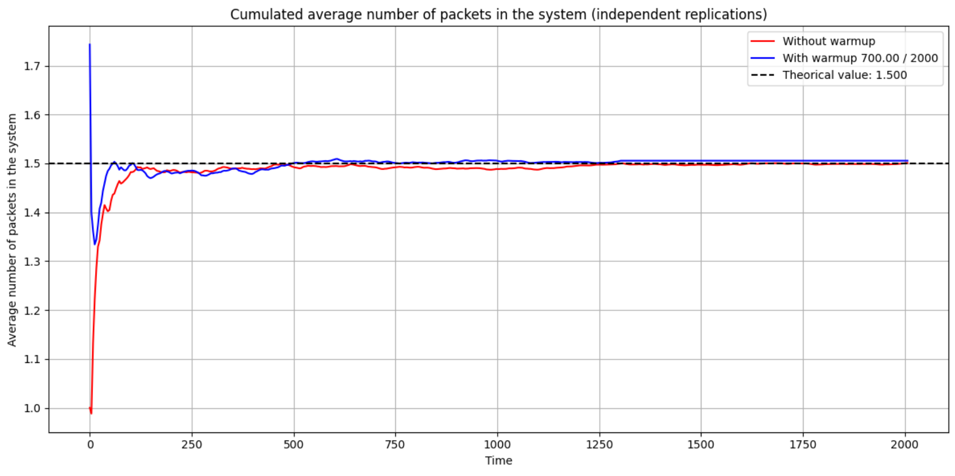
1.2.1 Low utilization ($\rho = 0.6$)

First of all we have investigated the system in condition of low utilization.

Setting of the experiment with *small* ratios: $\lambda = 1.2$, $\mu = 2$, *replications* $= 300$, *simulation_time* $= 2\,000$ and *warm-up_time* $= 700$.

Warm-up elimination helps the system to converge a little bit faster and has less variability in the first phase, but in the long term the high *simulation_time* "wins" and the averages empirically computed are similar and not influenced by the warm-up elimination (1.5056 with warm-up and 1.5009 without).

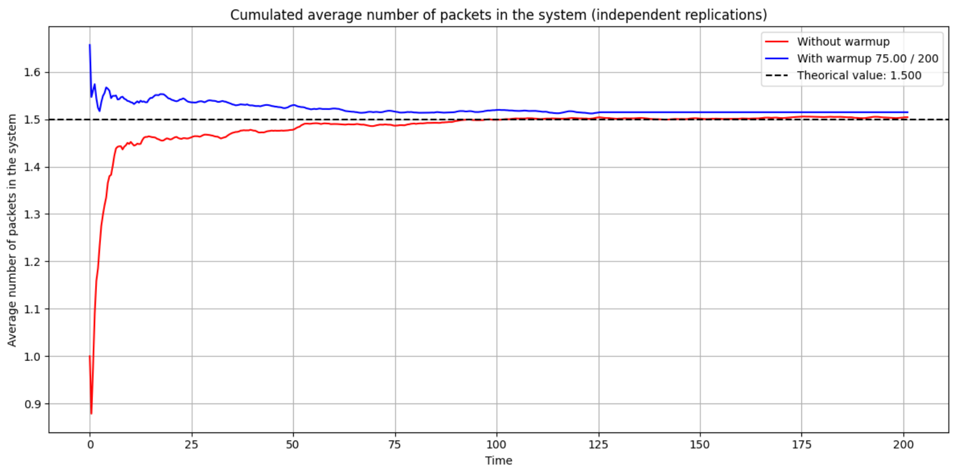
Reducing *simulation_time* or increasing *warm-up_time* does not help, because we end up with too few observations to average out and not precise results.



Setting of the experiment with *high* ratios: $\lambda = 12$, $\mu = 20$, *replications* $= 300$, *simulation_time* $= 200$ and *warm-up_time* $= 75$.

Similar consideration can be done in this case; the only interesting thing is the much more reduced amount of time we need to run the simulation to have good results (with high ratios and low utilization system converges faster because more packets interact with the system in less time).

To conclude the analysis, we can say that with low utilization of the system there are not significance advantages by eliminating warm-up and it is better to increase *simulation_time* to have good results and better convergence.

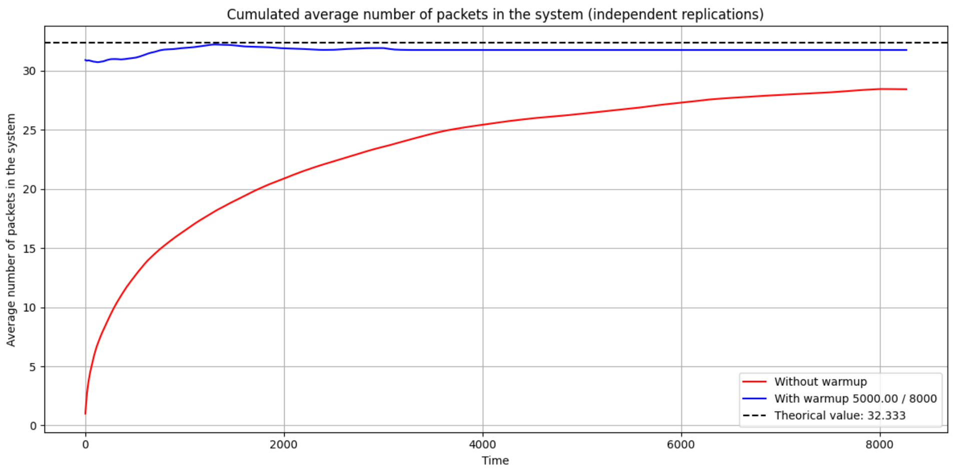


1.2.2 High utilization ($\rho = 0.97$)

With the system near instability, we got the most interesting results.

Setting of the experiment with *low* ratios: $\lambda = 0.97$, $\mu = 1$, *replications* $= 450$, *simulation_time* $= 8\,000$ and *warm-up_time* $= 5\,000$.

In this setting we can see the importance of the warm-up. The instability period at the beginning is very high and only cutting the packets in the first $5\,000$ time units we have a faster convergence and an accurate estimation of the average number of packets in the system (theoretical value is *32.33*).



	With warm-up	Without warm-up
Average number of packets in the system	**31.74**	28.42
Confidence Interval (95%)	**[29.42, 34.05]**	$[26.95, 29.88]$

Making the simulation time longer to overwhelm the initial bias would take too much computational complexity: in this case, warm-up elimination is crucial.

Setting of the experiment with *high* ratios: $\lambda = 9.7$, $\mu = 10$, *replications* $= 300$, *simulation_time* $= 1\,000$ and *warm-up_time* $= 800$.

We avoid introducing the plot in that case since it's similar to the one above, apart from the less simulation time used.

In this case we wanted to visualize the effect of the warm-up in a different way: since with high ratios and high replications making a long simulation is computationally expensive, we wanted to verify what happens with a high warm-up time (we truncate 80% of the packets).

	With warm-up	Without warm-up
Average number of packets in the system	**31.00**	28.44
Confidence Interval (95%)	**[28.19, 33.82]**	$[26.83, 30.06]$

Results are interesting: since we truncate a lot of "bad" packets, even with few packets considered we have better estimations. From our simulation results, this is different from the low utilization case when it was very important to have a lot of packets to have smooth estimations. From our point of view, this is happening because the instability part at the beginning of the simulation is very influential for the entire simulation.

Some general considerations we can make from these two simulations cases:

- increasing *simulation_time* is good for having good estimations and convergence, but it is very computationally expensive
- especially with *high utilization rate* choosing the right warm-up time is not only useful to faster the convergence, but also for having good approximation of the theoretical metric we are computing (that's because with high ρ the variability in the first period is very high)

- increasing the number of the *independent replications* we can reduce variability, do statistically robust considerations and find more general the warm-up and convergence time.

The script used to run the simulations is `comparison_with_theoretical_results.py`.

1.3 Application of variance reduction techniques

Second round of simulation and analysis we did has been about the average traversal time for any packet.

Similarly with the average number of packets in the system, we used independent replications and warm-up to improve the quality of our simulation and we also put in practice two more techniques:

- use the number of packets in queue as a **control variate**;
- **post-stratification**: number of packets in queue when a packet enters the system is our stratium.

Setting of the experiment was the following: $\lambda = 0.97$, $\mu = 1$, *simulation_time* $= 8\,000$, *replications* $= 450$ and *warm-up_time* $= 5\,000$.
In the following table we resume the results we got.

	Estimation	Variance
Theoretical value	33.33	-
Naive estimator	32.49	608.6
Control variate	33.31	0.4899
Post stratium (by computing *empirically* the p_i)	32.69	2.510e-05
Post stratium (by computing *theoretically* the p_i)	33.15	2.606e-05

For completeness: the number of packets in queue was theoretically 31.36 and our estimation was 30.57.

Unlike the naive one, **all the methods implemented help dramatically reduce the variance**. In particular, the post-stratification method has a much lower variance than the control variate one (which still has a variance 1000x lower than the naive method).

An interesting consideration can be made with the post-stratification estimator. We have implemented two versions, which are different depending on how they compute p_i (probability of having i packets already in the system when a packet arrives).
In one case, they are computed *from the theory* (considering this distribution $\pi_i = \rho^i(1-\rho)$), in the other case they are computed *empirically* (in fact, at every arrival we store the number of packets already in the system; then we also store the time needed by a packet to traverse the system, so we can compute empirically p_i and traversal time with a given number of packets alraedy in the system).

Differences between the two are minimal: not only in terms of variance, but also for the estimation of the metric. This suggests that the number of replications, the *simulation_time* and the *warm-up_time* are set properly and empirical values are good approximations of the theoretical values of the system!

The details of the implementation are available in the file `exercise2.py`.