Computer-aided simulations - First Delivery

Alessandro Maini
s346219@studenti.polito.it

***Abstract*—This document provides an overview of the first three laboratory activities of the *Computer-aided simulations lab* course.**

I. Car-Sharing System

The first task undertaken was the design and implementation of a car-sharing system simulator. In the following subsections, the modeling of the system, the code implementation, and the test performances are discussed in detail.

A. Model Design

The simulation models a car-sharing system based on a fleet of electric vehicles (EVs) distributed across the city at various charging stations. Throughout the day, user requests for vehicles arrive at varying rates, reflecting realistic urban mobility patterns: higher demand is observed during morning commuting hours (e. g., people traveling to work or school) and again during the evening commute (typically 4-8 p. m.). In particular, a fixed arrival rate is assumed for each defined time window:

- From 6 a. m. to 10 a. m.: maximum arrival rate (peak rate), the users travel from home to work.
- From 10 a. m. to 4 p. m.: 60% of maximum rate (day rate), the users travel between residential distribution.
- From 4 p. m. to 8 p. m.: maximum arrival rate (peak rate), the users travel from work to home.
- From 8 p. m. to 6 a. m.: 20% of maximum arrival (night rate), the users travel between residential distribution.

Residential areas and key destination hubs (such as workplaces and schools) follow distinct spatial distributions, which together define the predominant travel flows between residential and activity areas. The probability density distributions are generated using a multivariate Gaussian mixture model, with 2 dimensions, trying to simulate multiple population centers within a city. Figure 1 and Figure 2 show respectively the residential and workplaces distribution used throughout the simulation.

Each user can request a vehicle through the mobile application. If no nearby vehicle is currently available, the user is placed in a waiting queue. When a suitable car becomes available within a defined proximity threshold, the user is notified and proceeds to the assigned vehicle.

Once the trip begins, the vehicle is driven to the specified destination and parked either at the nearest available charging station or directly at the destination (assuming there is always at least a normal parking spot at that exact location) if no charging station is nearby. Vehicles parked at charging stations automatically recharge their batteries.

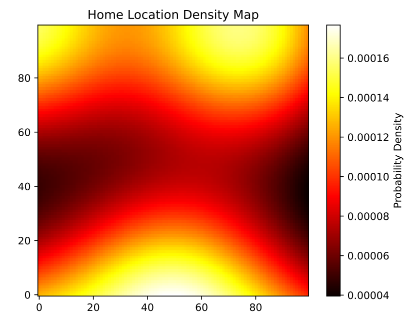

Fig. 1. Residential probability density distribution.

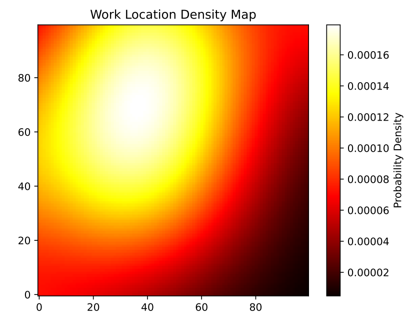

Fig. 2. Probability density distribution of workplaces.

To balance supply and demand, vehicles are relocated during nighttime hours to better match the spatial distribution of expected requests. Cars are always relocated to charging stations.

When a vehicle's autonomy becomes low, it must be relocated to a recharging station and marked as unavailable until it reaches a minimum level of autonomy.

Trip origins and destinations are stochastically generated based on time-dependent spatial distributions. For instance, during the morning hours, trip origins follow the distribution of houses, while destinations follow the distribution of workplaces; this pattern is reversed in the evening. User request arrivals are modeled using a Poisson process, while vehicle speed varies throughout the day as a function of traffic density (i. e., the number of active cars), thereby simulating congestion effects. The travel time for each vehicle is estimated under the assumption of a direct, straight-line route from the starting point to the finishing point.

The locations of charging stations are fixed and determined according to both residential and workplace/school densities, ensuring that high-demand areas for trip origins and destinations are adequately covered.

B. Hyperparameters and Key Performance Indicators

The system incorporates several operational constraints and behavioral parameters, including the following:

- Maximum waiting time: the maximum time a user tolerates before a nearby vehicle becomes available (after which they abandon the request).
- Maximum pickup distance: the maximum allowable distance between a user's location and the assigned vehicle.
- Maximum destination-charging distance: the maximum acceptable distance between the trip destination and a charging station (beyond which the car is parked directly at the destination, with no charging).
- Vehicle autonomy: the total distance a vehicle can travel without recharging.
- Minimum trip distance: users will not choose a shared car for very short trips below a given distance threshold.
- Recharging rate: the rate at which the cars are recharged.
- Minimum autonomy: vehicles with low battery levels must be relocated to a charging station or marked as unavailable.

Finally, several key performance indicators (KPIs) are monitored to evaluate both system efficiency and user satisfaction, including:

- Vehicle availability rate,
- Vehicle utilization rate,
- Average user waiting time before a nearby car becomes available,
- Percentage of user abandoning their request,
- Average trip distance traveled per user,
- Utilization rate of charging station parking spots,
- Number of active users (i. e., users requesting cars),
- Number of moving cars (i. e., users being served).

C. Events and System Structure

The system considers 5 possible events, stored in a Future Event Set (a simple priority queue) and handled in chronological sequence:

1) **User Request**: occurs when a user enters the system asking for a car. If a nearby car is available, he is immediately served, otherwise, he is placed in a waiting queue. This event also triggers the next arrival, according to a Poisson process.
2) **User Abandon**: occurs when a user has been waiting in queue for the maximum waiting time, without car assignment.
3) **Car Parking**: occurs when a car arrives at destination. If a nearby charging station is available, then the car is parked there and charged, otherwise, it is parked at a normal parking spot exactly at destination point. This event also checks the residual autonomy of the car: if it has enough autonomy a *Car Available* event is triggered, otherwise, it becomes unavailable (if the car is also not charging, it becomes dispersed and needs relocation).
4) **Car Available**: occurs when a car has finished its travel and still has enough residual autonomy. If there is nearby waiting user, he is assigned to the car and served, otherwise, the car remains at its location waiting for other user requests.
5) **Relocation**: occurs once a day, at midnight. A predefined number of cars is relocated to charging stations with low utilization, assuming that there is high-demand in those areas. First all the dispersed cars (not available and not charging) are relocated, then those cars that are most distant with respect to high-demand areas.

Focusing on the structure of the code implementation, three main objects are defined:

- *Car*: that, depending on internal parameters, can be either available for travel, in usage by a user, in recharging at a station or dispersed (i. e., with low autonomy and not charging);
- *User*: that can be immediately served or placed in the waiting queue; additionally the object stores statistics regarding the waiting time;
- *Charging station*: that has a fixed location and a fixed capacity; the object tracks also information about the parked cars.

The key performance indicators of the overall simulation are computed and stored in a *CarSharingSystem* object, that incorporates many utility functions to: track users and cars, sample statistics (e. g., queue length, number of total users), compute time-weighted statistics (e. g., utilization rate, availability rate) and calculate cumulative statistics (e. g., abandonment rate, average travel distance). Additionally, peak arrival rate window statistics are separately calculated to analyze the performance of the system in the most critical hours of the day (i. e., morning and evening commute).

Finally, the core of the simulation is implemented in the *SimulationEngine* class, in which all the hyperparameters are set, the probability density distribution maps are created (one for residential area and another for working places), the charging stations are placed on the map (according to both the previous probability maps) and all the cars are allocated in the stations. Then the main event loop handles all the events that appears in the Future Event Set, until the end of the simulation. The class contains also some methods to retrieve and print the statistics collected during the simulation.[1]

D. Experiments and Results

To evaluate the performance of the simulated car-sharing system under different scenarios, several configurations of hyperparameters are taken into account. Table I summarizes the hyperparameters with the corresponding tested values.

As shown in Figure 3, the performance of the system are almost independent from the number of charging stations,

[1]Additional information about the code implementation can be found in the docstring of `car_sharing.py`.

TABLE I
TESTED HYPERPARAMETER VALUES FOR SIMULATION CONFIGURATIONS

Hyperparameter	Tested Values
Number of cars	300, 400, 500
Number of charging stations	150, 250, 350
Maximum destination charging distance (m)	600, 800, 1000
Maximum pickup distance (m)	800, 1000, 1200
Maximum waiting time (min)	15
Number of cars to relocate	100
Maximum arrival rate (requests/min)	4
Maximum car speed (km/h)	30
Maximum station capacity (#cars)	4
Simulation time (days)	7
Total car autonomy (km)	150
Minimum trip distance (km)	3

in fact the 3 lines largely overlap. On the other hand, the evaluation metrics change significantly when the number of cars is varied, going from an average drop of 9.7% when the total cars are 300 to only 3.9% when they are 500 (60% less).

This better performance, however, comes with a cost, in fact the overall utilization of cars notably drops when increasing their number, since the arrival rate of users remains the same.

Another thing to notice is that the average waiting time of users is very low (less than half a minute), this means that the large majority of requests are served immediately after they are made.

A reasonable solution consists of having the lowest amount of stations (i. e., 150), because this way construction costs are reduced without remarkably affecting the system, and a medium number of cars (i. e., 400), to reduce the abandonment rate of users and still have a not too low overall utilization.

Once the dimensioning of the system is set, the simulation is evaluated under different user-depending conditions, i. e., the maximum distance a person is willing to walk to reach his assigned car (maximum pickup distance) and between the parking stop and his destination (maximum destination charging distance).

As shown in Figure 4, the most influential parameter is the pickup distance, that directly correlates with both abandonment rate and utilization rate of cars, while for these 2 metrics the distance between destination and charging station is almost irrelevant.

On the contrary, the destination charging distance is a crucial factor with respect to the stations utilization rate, having much higher utilization when the maximum distance increases. In a nutshell, if users are disposed to walk a greater distance, then the performance of the system largely improves, with lower drop of users and higher overall cars and stations utilization.

Also for these system-independent parameters a reasonable hypothesis is that users will accept to walk no more than one kilometer both from their starting point and the arrival parking spot. For these reasons, from this point on, the configuration of the hyperparameters considered will be the one shown in Table II.

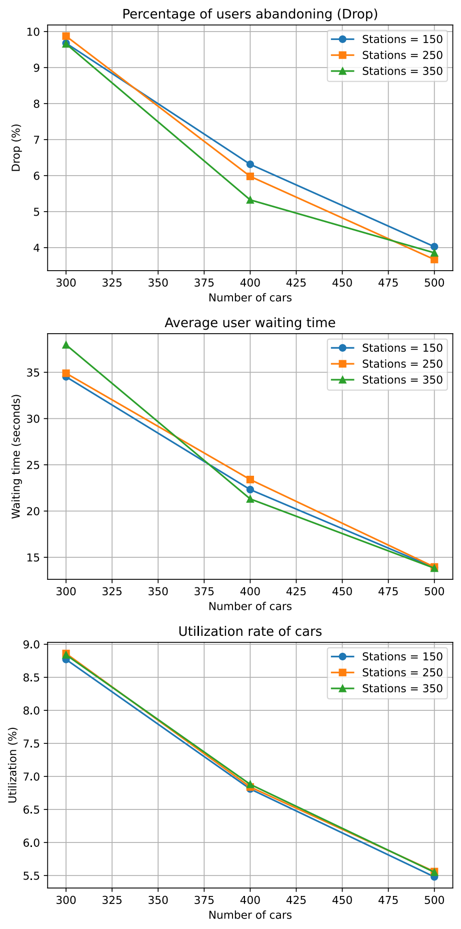

Fig. 3. KPIs for various configurations of cars and charging stations (max destination charging distance is set to 800 m and max pickup distance is set to 1000 m).

TABLE II
FINAL HYPERPARAMETER CONFIGURATION

Hyperparameter	Final Value
Number of cars	400
Number of charging stations	150
Maximum destination charging distance (m)	1000
Maximum pickup distance (m)	1000

Other important metrics to evaluate are those concerning the behavior of the system during peak arrival hours (i. e., 6 a. m. - 10 a. m. and 4 p. m. - 8 p. m.), that are the most critical. For this analysis the KPIs tracked are the average waiting queue length, the vehicles utilization and their availability (that should be approximately complementary). Further measures, such as: user abandonment rate, average waiting time and stations utilization, do not differ significantly between peak hours and the rest of the day. Different values for the user request rate are considered, in particular: 1, 2, 4 and 6 requests/min. The results are shown in Figure 5.

As expected, the tests show that during peak hours vehicle utilization increases notably and conversely vehicle availability

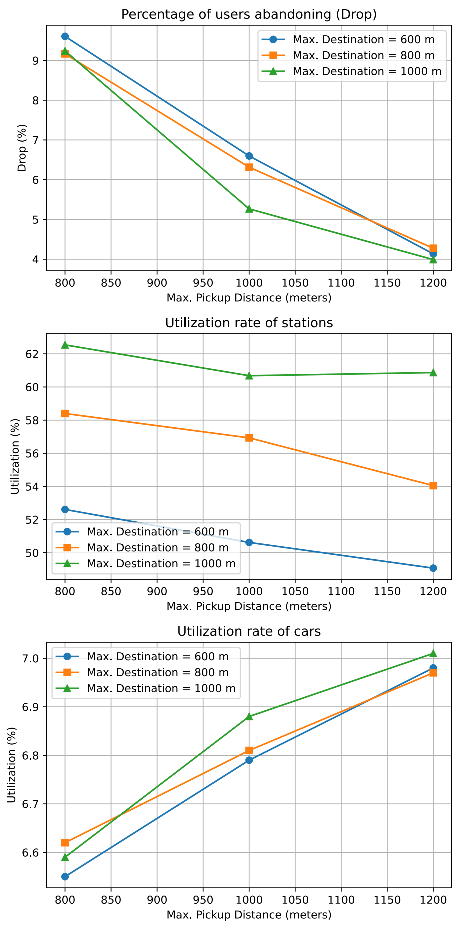

Fig. 4. KPIs for various configurations of maximum pickup distance and maximum destination charging distance ($\#cars = 400$ and $\#stations = 150$).

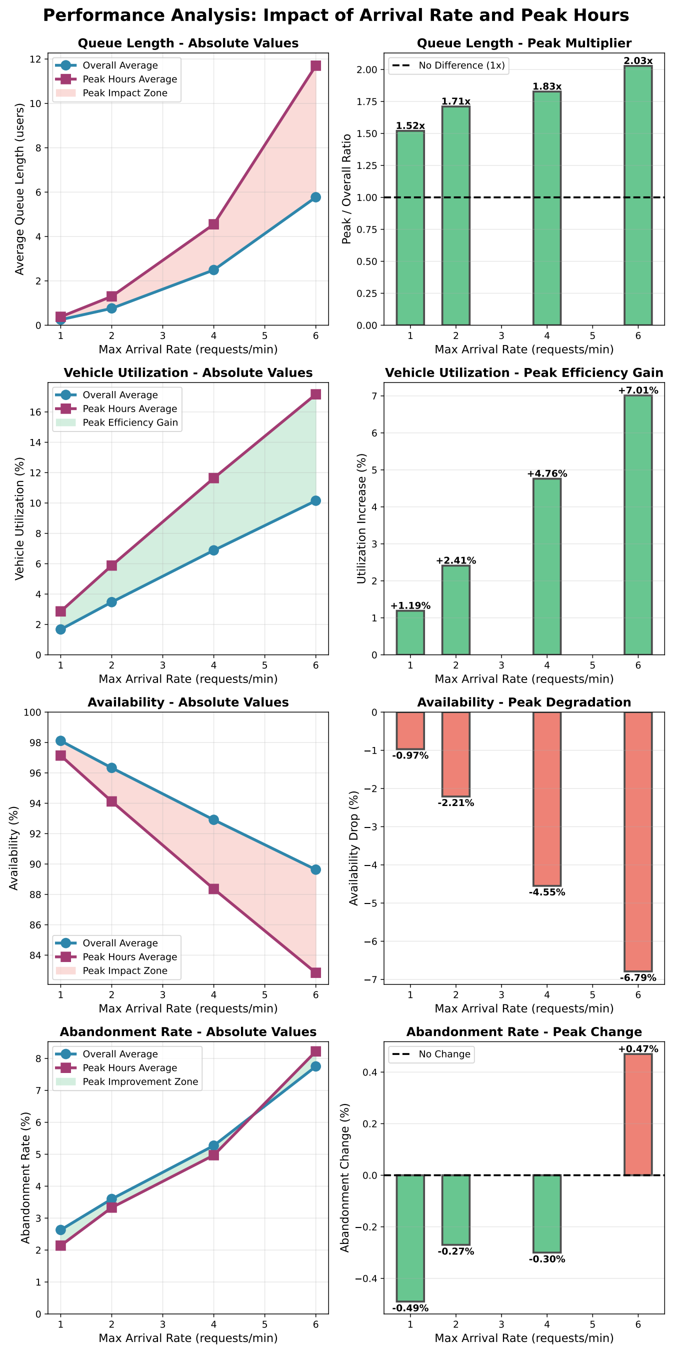

Fig. 5. Peak vs. Overall measures with respect to maximum arrival rate.

decreases by about the same amount, also the waiting queue is much larger. Less obvious is the result obtained for abandonment rate, which demonstrates the system's resilience in the face of a dynamically changing rate of requests during the day. The graphs also clearly show that the vehicle utilization and availability, together with the user abandonment rate are linearly dependent with respect to the maximum arrival rate of users, while the average queue length exponentially increases with respect to it.

II. Transient Phase Detection and Confidence Intervals

The second main task of the delivery is twofold: to develop and implement an automated algorithm to detect the end of the transient phase in a stationary system, and to compute confidence intervals in a non-stationary system. The following subsections describe the solutions for each problem.

A. Transient Phase Detection Algorithm

The proposed algorithm to detect the end of the transient (or warm-up) phase in a stationary system is defined in Algorithm 1. The code implementation of the algorithm can be found in `transient_detection.py`.

The *TransientDetection* class contains some hyperparameters that need to be tuned according to the specific system for which the transient has to be detected. These parameters include:

- Window size: length of the time interval over which the average quantity is computed.
- Window stride: length of the time interval between 2 consecutive windows.

Algorithm 1 Detection of the End of the Transient Phase in a Stationary System

```
Require: Measured quantity $X(t)$, window duration $T$, stride $S$, number of stored averages $N$, threshold $P$
Ensure: Estimated end time of transient phase

1: Compute the sequence of averages of $X(t)$ over a sliding window of duration $T$ and stride $S$.
2: Store the last $N$ computed averages: $\{\bar{X}_1, \bar{X}_2, \ldots, \bar{X}_N\}$.
3: Compute the mean squared variation between consecutive averages:
       $$V = \frac{1}{N-1}\sum_{i=1}^{N-1}(\bar{X}_{i+1} - \bar{X}_i)^2$$
4: if $V < P$ then
5:     The transient phase ends at the start time of the first stored interval.
6: else
7:     Continue computing further intervals.
8: end if
```

- Number of intervals: how many averaging windows to consider when calculating the variance.
- Variance threshold: minimum variance needed to consider the transient concluded.

B. Transient Detection in a Queuing System

The previously presented algorithm is applied to detect the end of the transient in a simple "First Come, First Served" queuing system of type M/M/1 (i. e., exponentially distributed arrival and service times, with just one server) with a finite capacity of 1000. Three different scenarios are evaluated ($\lambda \sim$ arrival rate, $\mu \sim$ service rate, $N_0 \sim$ initial occupancy of the queue):

1) $\lambda = 0.8\mu$ and $N_0 = 1000$ (full);
2) $\lambda = 1.2\mu$ and $N_0 = 0$ (empty);
3) $\lambda = \mu$ and $N_0 = 0$ (empty).

The quantity $X(t)$ measured by the algorithm is, obviously, the queue length and the values selected for hyperparameters are reported in Table III. The results obtained for each of the 3 cases are respectively shown in Figures 6, 7 and 8.

TABLE III
TRANSIENT DETECTION HYPERPARAMETER CONFIGURATION FOR QUEUING SYSTEM

Hyperparameter	Selected Value
Simulation time	10,000
Window size	100
Window stride	10
Number of intervals	20
Variance threshold	0.1

The first thing to notice is that only in scenario 1 and 2, the system reaches a steady-state, while in scenario 3 the queue length continue to oscillate until the end of the simulation,

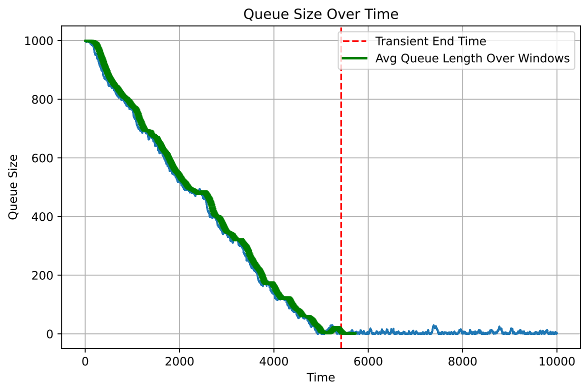

Fig. 6. Queuing system transient detection for $\lambda = 0.8\mu$ and $N_0 = 1000$.

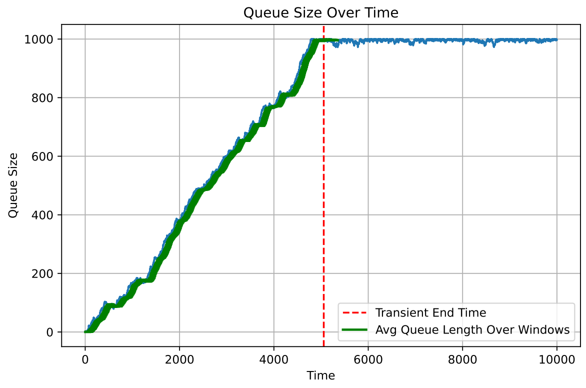

Fig. 7. Queuing system transient detection for $\lambda = 1.2\mu$ and $N_0 = 0$.

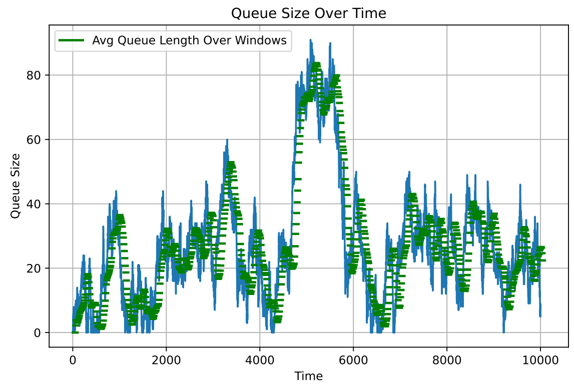

Fig. 8. Queuing system transient detection for $\lambda = \mu$ and $N_0 = 0$.

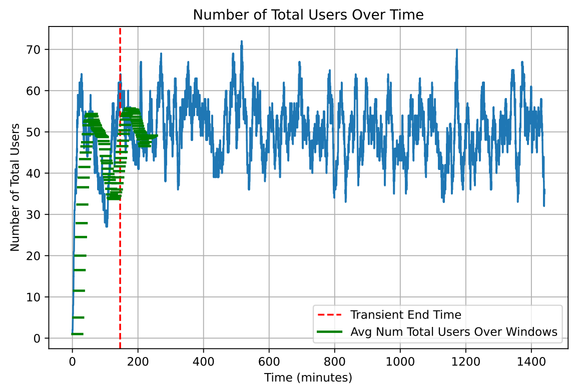

Fig. 9. Car-sharing system transient detection with respect to total users.

therefore the transient phase ending is not detected. The red vertical lines representing the end of the warm-up phase are very close to the flattening of the curve, so the algorithm performs well in this simple scenarios.

C. Transient Detection in the Car-sharing System

To detect the end of the transient phase in the car-sharing system, some modifications to the simulator are needed to force steady-state conditions. In fact, the system presented in the previous section has a cyclic behavior over the day, due to the dynamically changing requests rate and travels distribution (the direction of travel in the morning is opposite to that in the evening).

To accomplish a stationary condition, the following test is done under the assumptions that user request rate is fixed throughout the day (always the maximum rate), and all trips have both starting and ending point retrieved from a common probability density distribution over the map, that does not change. Additionally, the speed of the cars is artificially kept constant, not considering congestion.

The transient detection hyperparameters selected for the car-sharing system are shown in Table IV, while the metric considered is the total number of users in the system, both waiting for a car and already traveling (similar results are observed considering only served users). Figure 9 shows the obtained results.

TABLE IV
TRANSIENT DETECTION HYPERPARAMETER CONFIGURATION FOR CAR-SHARING SYSTEM

Hyperparameter	**Selected Value**
Simulation time (min)	1440 (1 day)
Window size (min)	30
Window stride (min)	2
Number of intervals	40
Variance threshold	0.3

Even if the conditions under with the simulation run are stationary, the total number of users is subject to a significant amount of noise, as can be seen in the image. Nevertheless, the proposed algorithm is still capable to detect when the average number of users stop increasing and stabilize around 50. Also longer simulations have been tested (e. g., one week, one month) and the warm-up phase always terminates in the first few hours.

D. Confidence Interval Estimation

To compute confidence intervals, a specific class is implemented in `confidence_interval.py`. This simple class stores a list of data points concerning the measure for which the C. I. has to be calculated, then different methods are defined to compute the sample mean and standard deviation used to retrieve the interval. Three hyperparameters can be tuned depending on the use case:

- Confidence level: usually called $1 - \alpha$, necessary to determine the z-score (only the standard normal distribution is considered; no Student's t-distribution).
- Maximum interval width: the maximum percentage deviation of the interval from the mean value, so that it can be considered definitive.
- Minimum sample count: the minimum number of data points required to compute a confidence interval.

Two different non-stationary scenarios are considered for the confidence interval estimation:

1) Cycle-stationary system, with slowly varying traffic and constant car speeds (i. e., no congestion).
2) Cycle-stationary system, with highly variable traffic and dynamic car speeds.

The first case is the most stable, but it is also the most unrealistic. To reduce traffic variability, the user request rate during no-peak day hours is set to 80% of maximum rate and the night rate is set to 60% of maximum rate. To avoid dependency on the warm-up phase, the first day of simulation is not considered for C. I. calculation. The independence between collected data points is obtained using the batch means method, i. e., the simulation is split in different non-overlapping intervals and, for each interval, the estimation of the metric under study is computed.

Since the user request rate assumes 3 distinct values throughout the day, then 3 different C. I. have to be calculated, one for each arrival rate. The metric considered in the following test is the total number of users in the system and the C. I. hyperparameters, along with the batch size, are reported in Table V.

TABLE V
HYPERPARAMETER CONFIGURATION FOR CONFIDENCE INTERVAL ESTIMATION

Hyperparameter	**Selected Value**
Batch size (min)	30
Confidence level	95%
Maximum interval width	5%
Minimum sample count	10

The simulation is run for one week, the final confidence intervals are shown in Table VI.

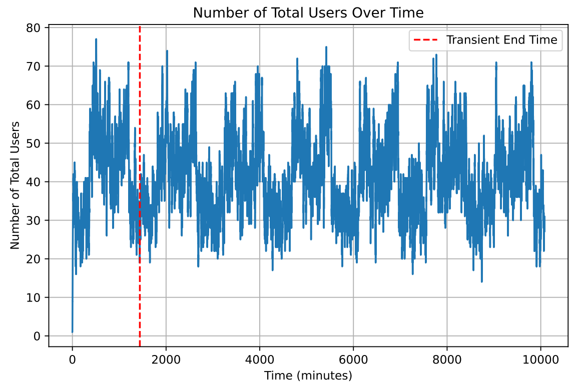

Fig. 10. Total users in the system over time with no congestion and slowly varying traffic.

Figure 10 plots the total number of users in the system during the simulated week. Even if the graph is very noisy, the different days can be identified by looking at the peaks, that correspond to the commute hours. Also the time windows with diverse arrival rates can be distinguished, in particular the night hours, that are characterized by a significantly lower number of users.

TABLE VI
CONFIDENCE INTERVALS PER ARRIVAL RATE PHASE (95% CI) - QUASI-STATIONARY CASE

Rate Mult.	Mean Users	CI Lower	CI Upper	CI Width
0.6	32.75	31.74	33.36	1.63
0.8	42.29	40.92	43.01	2.09
1.0	50.70	49.40	51.91	2.50

For the second case, the most realistic, user request rate over the day is set to the values presented in the section I-A (Model Design); additionally, the cars' speed is changed based on congestion, i. e., lower speed when a lot of cars are moving. Also in this case the metric under study is the total number of users in the system, and the first simulated day is excluded from C. I. calculation. The hyperparameters used are the same as well (see Table V), but to obtain final confidence intervals of the desired width a longer simulation is needed, since the behavior of the system is, obviously, more unstable.

The simulation is run for almost 50 days, the definitive confidence intervals are shown in Table VII.

Figure 11 plots the total number of users in the system for the first week of simulation. Differently than before, the distinction between the days and, also, the time windows with diverse arrival rate is very clear.

As a final consideration, to improve the accuracy of the C. I. estimation, a better transient detection approach could be adopted. Two possible solutions, not implemented in code, are:

1) Compute an average statistic over the all repeating unit (in this case, a day) and check the variance of this coarse-grained measure.
2) Use the algorithm proposed in section II-A separately for each time interval that behaves in the same way (in this case, the time windows with common request rate), then calculate a weighted-average of the different transient end times obtained, that should not differ significantly if the system is cycle-stationary.

TABLE VII
CONFIDENCE INTERVALS PER ARRIVAL RATE PHASE (95% CI) - REALISTIC CASE

Rate Mult.	Mean Users	CI Lower	CI Upper	CI Width
0.2	12.26	11.90	12.51	0.61
0.6	33.86	33.55	35.27	1.71
1.0	52.10	50.81	53.41	2.60

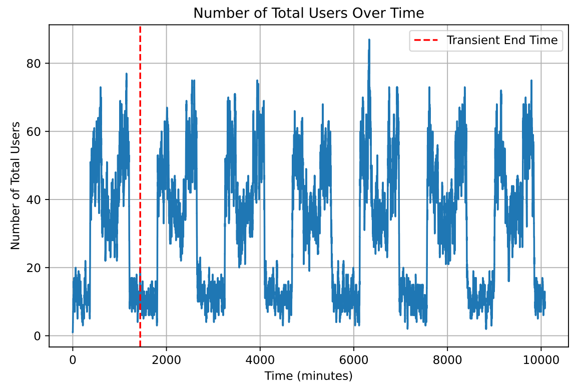

Fig. 11. Total users in the system over time with congestion and highly varying traffic.