BDA450 Project Report

Gabrielle McCabe, Anushree Nadkarni, Jean Leclerc, and Hyunsu Shin (Group C)

Your choice of your simulation paradigm:

In the case of a traffic merging simulation, we realised that it was important to keep track of certain statistics, but also that each car has their own behaviour. Because of this, we decided to implement a mixed approach that combines both discrete time and agent-based simulation. In our simulation, discrete time allows us to look at what's happening for every time step. This is particularly useful because for every step, we're able to keep track of certain metrics such as the speed of each car, which lane they're in, their acceleration, its distance with other cars, etc. Once we have this, we can then determine aggregate statistics such as how long each car spent in the area, what average speed it is travelling at, the overall speed of traffic flow, etc. Simultaneously, since each car also has their own behaviour, we decided to model them as agents. For example, cars in our simulation have different current lanes and destinations, different top speeds and desired following distance, as well as different merging behaviours (either early or late mergers). Thus, treating each car as an agent helps us greatly in modelling their behaviour throughout the simulation.

The architecture and design of your simulation:

The car class contains the behaviour of every car in our simulation. Each car has their own ID (for tracking purposes), speed, acceleration, minimum desired space between it and the car in front, lane and destination. Each car also contains information on whether they're an early merger or late merger depending on the simulation run. Since we went with an agent-based approach mixed with discrete time, the car class contains a step() method that dictates the behaviour of the cars at every time step. This step method differentiates between cars that are upstream and cars that are arriving through the on-ramp.

For upstream cars, we check if the car is in the same lane as its destination and make it merge with that lane if not. Upstream cars also switch to a faster lane if their current lane is going too slow (in our case this means +10km/h more than the other lane's speed.)

For on-ramp cars, two different options are possible: early mergers and late mergers. For early merging cars arriving through the on-ramp, they start checking the right lane if they can merge as soon as they reach the acceleration ramp. If the desired merging location is occupied, they continue to move along the acceleration ramp and try

again at the next time step. If they reach the end of the acceleration ramp before merging, they stop behind the other waiting cars (if there are any) until they have an opening to finally merge. For late merging cars arriving through the on-ramp, cars start checking if a merge to the right lane is possible only once they've passed the late merge x coordinate (which is set to 50m before the end of the acceleration ramp). This is a global variable in our code called X_COORD_START_LATE_MERGE. If the space is occupied, they continue to move along the end of the acceleration ramp and try again at the next time step. If they reach the end of the acceleration ramp, they stop behind the other waiting cars until they have an opening to finally merge.

For our simulation to work effectively, we created an environment that allows us to control the behaviour of the cars. In our code this is the Area class, and any car that is active in the simulation is managed through this class. Every time a car enters or leaves the area, it must do so through this class. Furthermore, the constraints on the movement of the cars is also implemented inside this class through the attemptmove method which checks for valid moves within the Area and ensures that cars can only move forward.

Similarly, the behaviour of every car during one time step is also executed through this class inside the run_step method. The run_step method calls the step() method of the car class for every car in the area, in an order such that the cars in front move before the cars behind. To accomplish this, we created the method sort_by_x which returns a list of the cars in the area in order of x-position. In addition, once a car arrives in a lane, the arrival time of the next car in that lane is also determined inside the run_step method.

The Area class also contains other methods such as get_next_x which returns the distance between a car and the car in front of it or get_lane_speeds which returns the average speeds of each lane.

For the methods inside the Area class to work, they must rely on the AreaGrid class which stores important information for every active car in the simulation, such as their x coordinates and which lanes they're in (y). With this information, we can then define methods such as get_list which returns a list of all agents. This get_list method is then utilised in the Area class for methods such as get_next_x, get_lane_speeds, get_car_in_front, etc.

The AreaGrid class also contains the isoccupied, add_item and remove_item methods, as well as the move_item method which removes the current x and y coordinates of an agent and updates it with the new coordinates.

Finally, the MergingSimulation class initialises the area, iterates the time steps, runs the simulation for a specified amount of time, and calculates the necessary output statistics. In addition, the run_sim method allows the user to choose if they want graphics to be displayed or not.

Choices of input distributions

For our selection of input distributions, we devised real-world statistics based on actual driver behaviours observed on highways. Following the project criteria, we established a speed limit of 100 km/h for the highway. Additionally, we set drivers' behaviour not to exceed 120 km/h, as surpassing this limit would constitute both a violation of the law (exceeding the speed limit by 20 km/h results in demerit points in the City of Toronto) and erratic driving. We also defined the lower speed parameter to be equal to or greater than 80 km/h, as driving at excessively low speeds on a highway can impede traffic flow and is considered unusual behaviour. To randomly assign a desired speed to each car, we extracted values from a normal distribution. If the value exceeded 120 km/hr or below 80 km/hr we chose another value. We used a normal distribution because we wanted to model most people driving the speed limit of 100 km/hr and an equal proportion of people travelling both above and below the speed limit; and the normal distribution fit this model best.

Furthermore, we incorporated a lognormal distribution for the desired space between cars. Maintaining a minimum two-second distance between vehicles is advised for safe driving. We used a lognormal distribution because we wanted to model most people having a desired following distance of 2 seconds, only a few having less than 2, and many having more than 2; and the lognormal distribution fit this model best.

Additionally, we employed an exponential distribution for the interval arrival time to simulate cars entering the highway section we are modelling. We used an exponential distribution because we wanted to model most interarrival times between cars being short since we are simulating high traffic volume; and the exponential distribution fit this model best.

Verification/ validation/ calibration:

Verification:

Throughout the project, tests were conducted to ensure that our code was running and performing correctly. To examine if a car was maintaining a safe distance from the car in front of it, we tested out our code by setting a following car's top speed to a maximum while the rest of the cars were significantly slower and observed if it would decelerate appropriately to maintain a gap. Another test we conducted was to assess whether a car would move to a faster lane, we did this by increasing the speeds of cars entering from a particular lane by a significant amount and monitored to see if cars on the remaining lanes moved to the faster one. We also wanted to make sure that our cars were leaving in the correct destination lane, therefore, to test if our cars would switch lanes to reach their designated destination, we set up a scenario where a car's starting and destination lanes were opposite and observed whether lane changes were happening.

Validation:

For the validation stage, we made sure to follow the project guidelines as they informed us about real-world traffic behaviours. However, we also added a couple of more driving characteristics and behaviours such as ensuring that cars stopped at the end of the acceleration ramp if there wasn't any space for them to merge and that drivers switched to their correct destination lane as soon as possible since the highway stretch covered in the simulation is only 550 m.

Calibration:

During the calibration phase, we focused on making sure our results made sense and matched real-world situations. We wanted to ensure that we were collecting data during high traffic periods, therefore, we first tested out our simulation for the duration of a typical rush hour in Toronto. A typical rush hour lasts for about 2 hours, therefore we wanted to run the simulation for 3 hours to ensure that we reach a steady state in our simulation. 3 hours is 7200 seconds - which when we tested this out it took longer than expected to run therefore, when reached the 5 minute mark of our simulation running we stopped the simulation. We then realised we had to decide whether we wanted to run this simulation for a longer period of time or have multiple runs of this simulation.

Hence, we started testing out shorter run timings and examining the results. We started off with 1000 seconds which is about 0.27 hours. This took over a minute to run,

the results did display speed reaching a steady state, however, the lines were not very smooth. Then we tried running it for 1800 seconds, which is 30 minutes, which took around 3 minutes to run and our results displayed a much smoother curve and we observed a steady state being reached around the 400 second time step. We wanted to test our simulation multiple times, therefore we decided to test slightly lower time steps, we tried 1000 time steps gain just to compare our results with the 1500 seconds simulation and as mentioned before, the results did reflect speed reaching a steady state but the graph for the 1000 time steps wasn't as smooth and we would only have 600 time steps to collect data. Therefore, we reached a compromise, we would run the simulation for 1400 seconds which is around 0.38 hours, this not only allowed us to have more time steps (1000 time steps) to collect data after reaching steady state, but it also enabled us to run the simulation five times without having to wait as long for our simulation to run.

We will be running our simulation five times for early merger and five times for late merger, each simulation will run for 1400 time steps.

Limitations of your simulation:

In our simulation, cars can have decimals for their distance and speed values. However, our area is composed of pixels and cars can only be visualised at integer positions, meaning we can't show the precise location of the cars. Because of this, the output graphics show a rounded approximation of the car position, not their precise position. This means that we can't fully rely on the graphics for validity.
Another limitation is that our cars change lanes instantly, by changing their y coordinate. In the real world, cars progressively merge onto another lane, changing in both x and y directions.

In our simulation, at each timestep our cars check if they're in the correct lane, then check if they're in the slower lane. If they're in the slower lane there is a 10% chance that they switch to the faster lane. However, at the next timestep, there is no check to see if that car passed the slow car. This means that a car could switch to a faster lane, then switch back to the lane that leads to their destination without having passed the slow car.

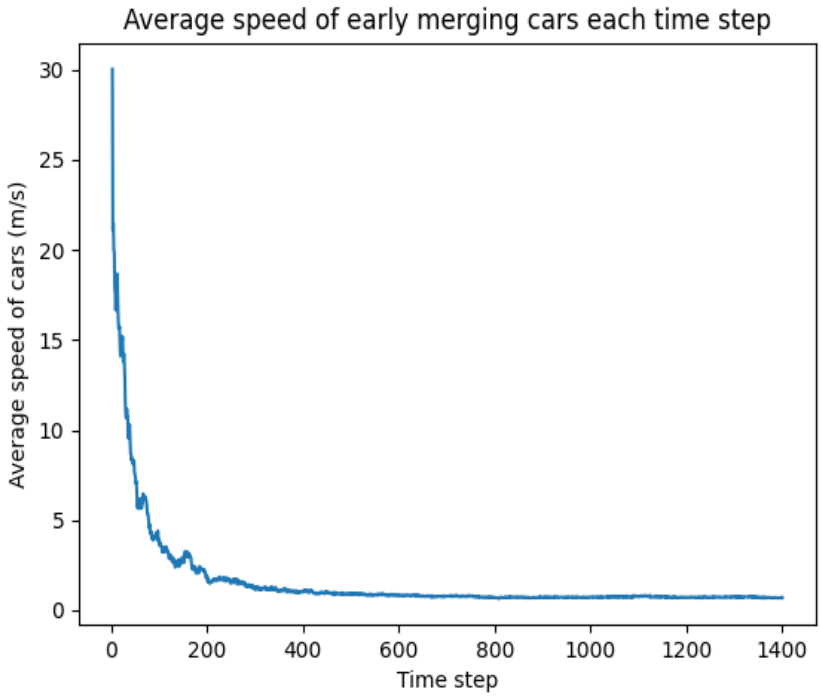
Another limitation is that we start our simulation with no cars and then populate the area as time goes on, as opposed to the real world where highways already have cars on them.

Additionally, all the late mergers in our simulation start looking to merge at the same location, which is unrealistic compared to the real world where late merging can occur at different positions.

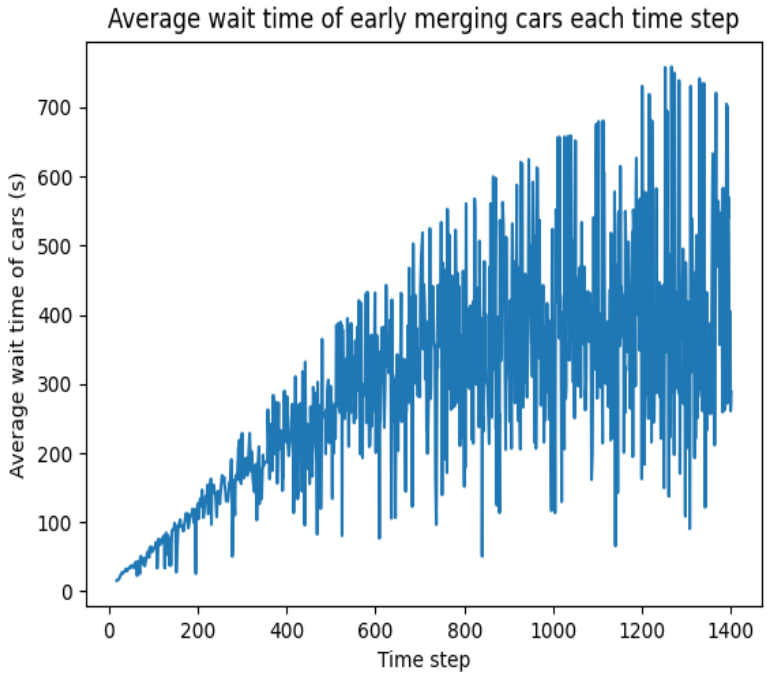
Finally, our simulation takes a while to execute (it takes around 10-15 mins on our school laptops without even the graphics).

Output analysis:

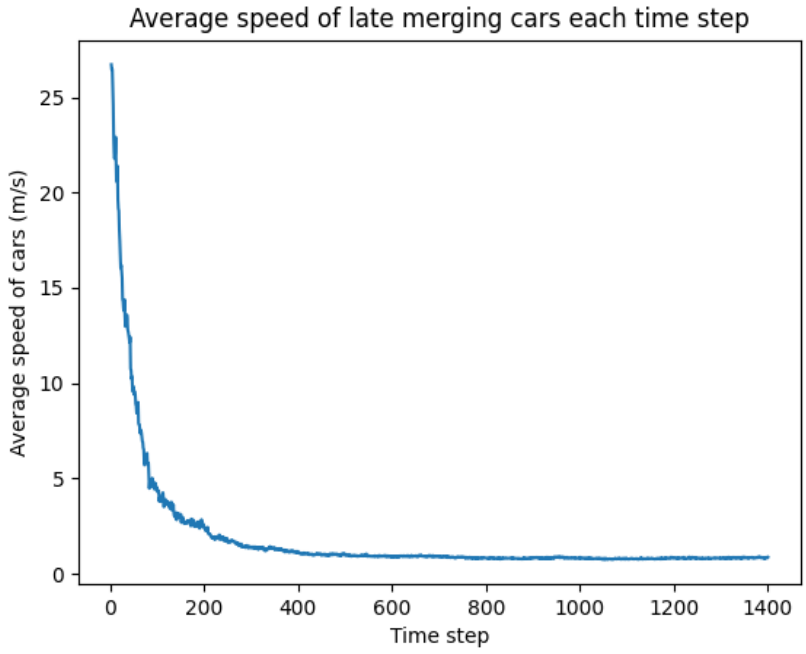
Observing when the simulation reaches steady state.



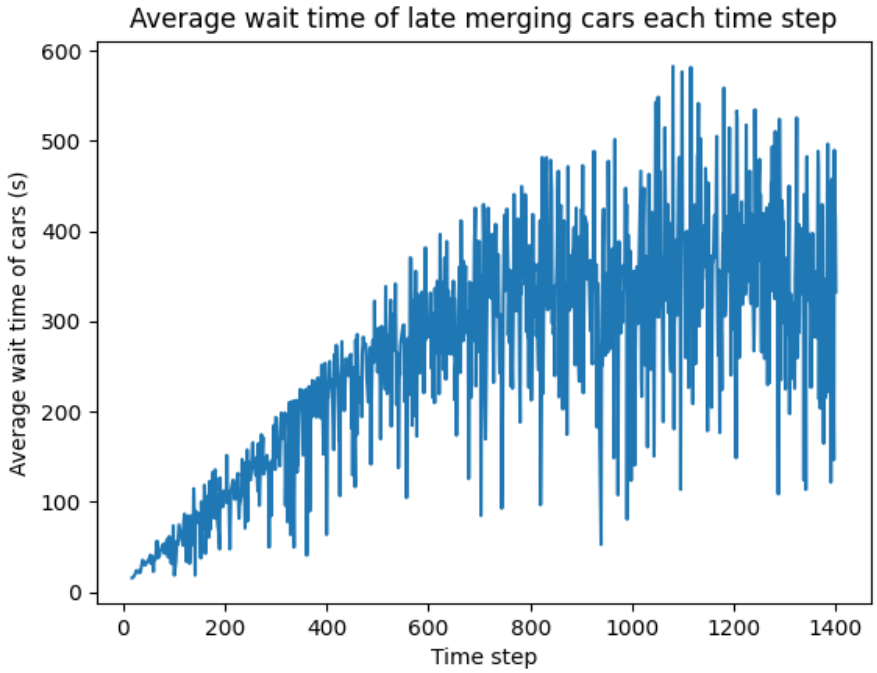
From the above graph, we can see that the steady state begins at time step 400.



From the above graph, we can see that the steady state begins at time step 600.

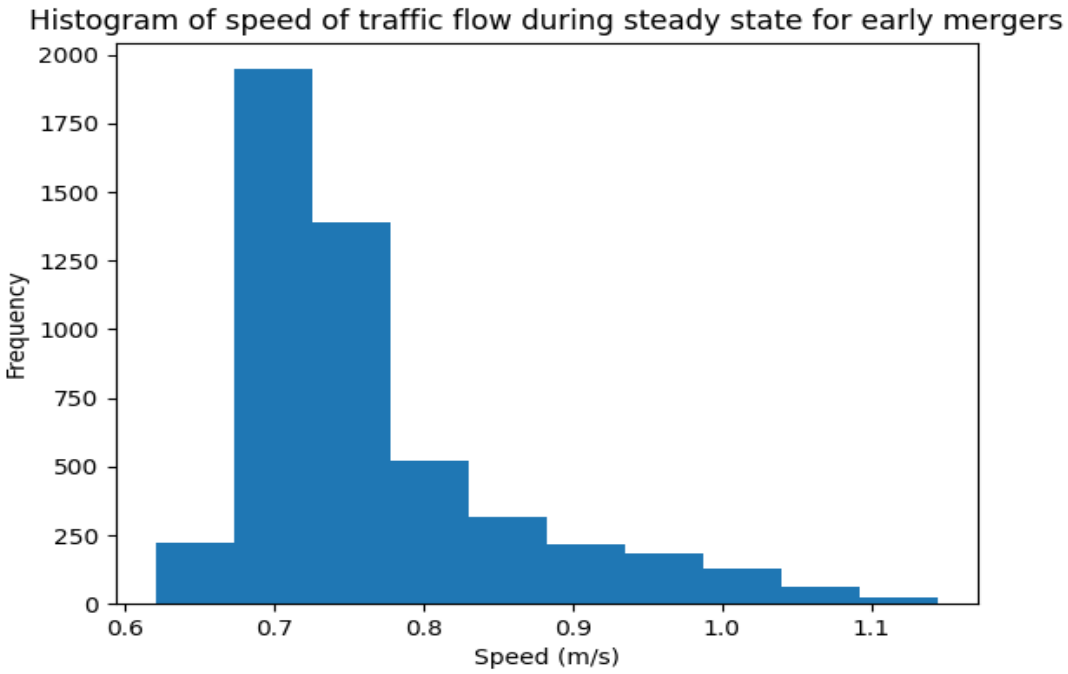


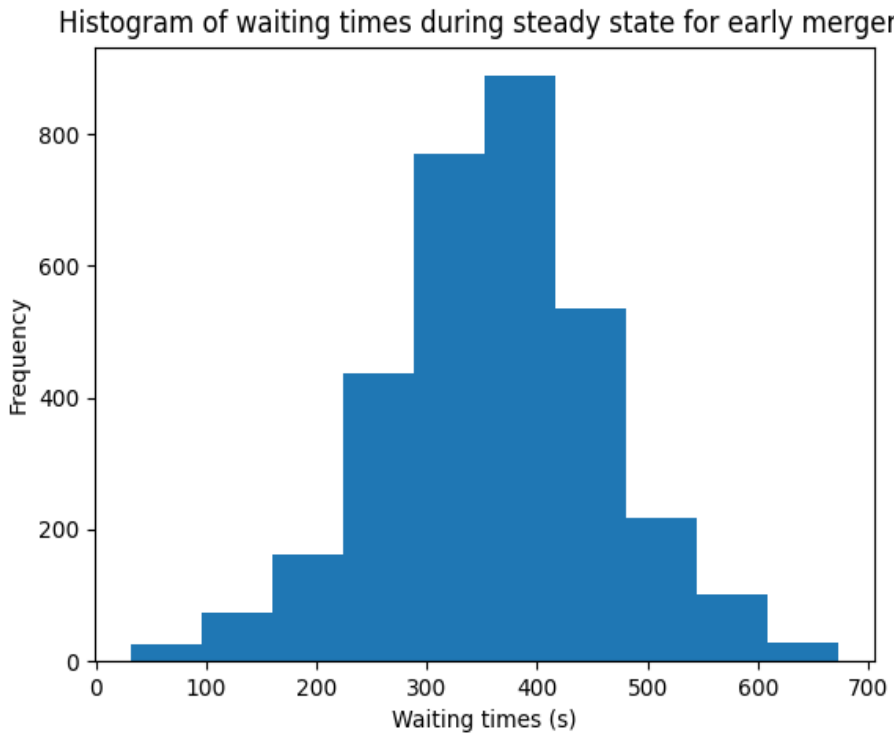
From the above graph, we can see that the steady state begins at time step 400.

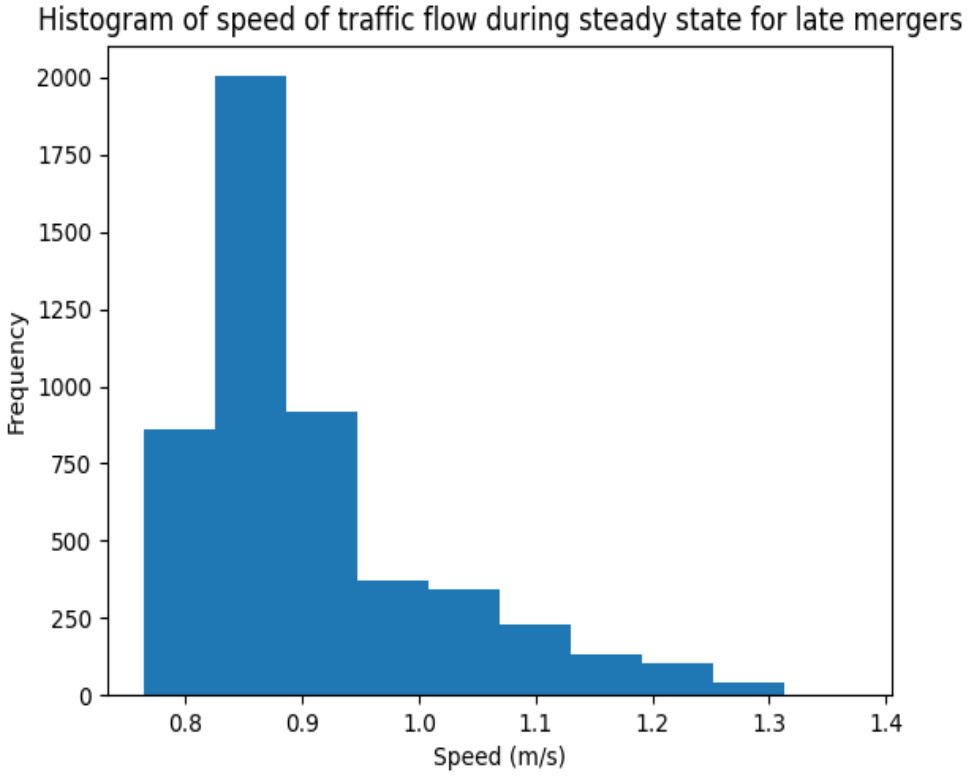


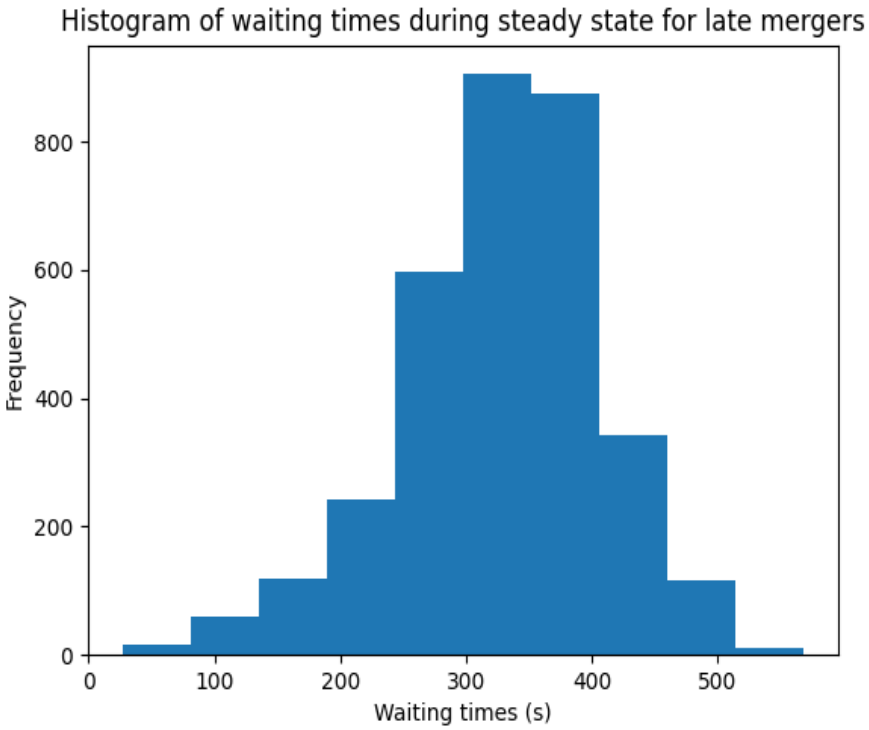
From the above graph, we can see that the steady state begins at time step 600.

Simulation output:




Histogram of waiting times during steady state for early mergers
Frequency
0
200
400
600
800
0
100
200
300
400
500
600
700
Waiting times (s)


Histogram of speed of traffic flow during steady state for late mergers
Frequency
0
250
500
750
1000
1250
1500
1750
2000
0.8
0.9
1.0
1.1
1.2
1.3
1.4
Speed (m/s)



Statistics for early mergers during steady state:
For average speed (i.e. speed of traffic flow) (m/s):
Mean: 0.7650
Standard deviation: 0.0105
95% confidence interval limits: 0.7504, 0.7795

For waiting times (s):
Mean: 360.9375
Standard deviation: 2.7412
95% confidence interval limits: 357.1321, 364.7429

For throughput (cars/hr):
Mean: 5005.0286
Standard deviation: 66.9678
95% confidence interval limits: 4912.0623, 5097.9948

Statistics for late mergers during steady state:
For average speed (i.e. speed of traffic flow) (m/s):
Mean: 0.9085
Standard deviation: 0.0136
95% confidence interval limits: 0.8896, 0.9273

For waiting times (s):
Mean: 327.9230
Standard deviation: 7.0169
95% confidence interval limits: 318.1820, 337.6640

For throughput (cars/hr):
Mean: 5058.5143
Standard deviation: 65.1825
95% confidence interval limits: 4968.0264, 5149.0022

Conclusion:

According to our simulation results, we found that the commonly cited “fact” that the late merge is more efficient for cars entering the highway via an on ramp than the early merge is actually true. The mean speed of traffic is faster, the mean waiting times are less, and the mean throughput is higher when cars perform a late merge as opposed to an early merge.