Penn State Harrisburg

COMP 512 - Advanced Operating Systems

P2P Based Platoon Simulation Model

April 29, 2019

Author
Varshil Anavadia
In Jung Kim

Professor
Dr. Linda Null

Contents

List of Figures

1 Executive Summary

This document attempts to address the problem of platooning, also known as flocking of automobiles to increase road capacity [12]. That is developing a system to decrease time to commute, decrease fuel consumption, detect and avoid collisions and consequently reduce traffic congestion. However, as we discuss later that dealing with such a system involving a dynamic environment and changing variables in real time is challenging. Also, the lack of a completely autonomous car with so-called *'brain'* and *'eyes'* adds to the difficulty.

In addition, this document describes the attempts taken to approach the problem with the goal of maintaining a constant headway within the platoon based on distributed systems concepts. Our primitive approach tackles this problem in a single lane and demonstrates how **V2V** communication contributes towards achieving this goal. Specific details about design, implementation and results have also been discussed in the document.

Finally, this document concludes with the obstacles encountered during the project, steps taken to resolve them and possible future works.

2 Introduction

2.1 Problem Definition

Many automotive companies are working towards developing fully automated (driver-less) vehicles. Information about safety and benefits of driver-less vehicles can be found in this article by the National Highway Traffic Safety Administration discusses [2]. An autonomous car [13] strongly contributes to building an automated driving system as it requires a reliable **V2V** communication link capable of efficient and robust operations in a real-time dynamic environment. The main purpose of developing an **ITS** [11] like platooning is to reduce traffic congestion and to avoid collisions on the road. Other benefits include shorter commute time to destination and fuel efficiency.

Developing a robust communication protocol with very low transmission latency that operates efficiently in a dynamic environment is tough. The challenge is that it is hard to manipulate certain conditions and variables like number of vehicles on the road, make of these vehicles, timings of traffic signals, unanticipated interruptions while driving, the behaviour of the driver, road conditions, traffic and safety laws and many more. Also, these variables are harder to predict and control with vehicles operating with a manual transmission as this class of vehicles simply operates on mechanical operations and is not built to consider these variables.

Our aim is to simulate an intelligent platoon for a single lane that is based on a **P2P** architecture which maintains a constant headway between any two vehicles in the platoon and avoids a collision. The basis of constructing the simulation is to use basic **TCP** socket communication. Our main focus is to reduce any latency that exists in the communication architecture to ensure that vehicles convey information of value in the least amount of time.

2.2 Previous Approaches

Some approaches have been taken to develop efficient and robust platooning systems in the past. Benalla, Achchab, and Hrimech (2016) design an intelligent distributed system using multiagent systems for guiding emergency platoon to their destination [6]. Darbha, Konduri, and Pagilla (2018) analyze the benefits of **V2V** communication and conclude that employable headway decreases and the platoon is much stable when information from r predecessors is utilized for communication [10]. According to Brown et al. (2000), a reliable communication link between vehicles is a key element to **AVCSS** [4]. Their simulation concludes that not only the platoon remains stable but also the performance of the communication protocol is unaffected even in the case of communication outages. Brief performance evaluation of platooning and **CACC**, two upcoming **ITS**, has been provided by Vinel, Lan and Lyamin (2015) in their work [1]. It should be noted that while the background study mentioned above may not be entirely relevant to our application, it does provide better insight to guide us through the implementation.

2.3 Inspiration

This project takes inspiration from a GitHub simulation which explores the idea of a conceptual algorithm for self-driving cars with acceleration and braking capabilities [3].

2.4 Proposed Solution

Our approach to solving this problem is to use distributed computing concepts, mainly the **P2P** architecture to develop a communication mechanism to maintain the platoon. In addition to **P2P** based communication, we have also used client/server architecture. The server is only used to set up the platoon by accepting client connections and to visualize the simulation once it begins. Hence we have a number of computers simulating one or more vehicles and communicating with each other (clients or cars) and a server visualizing platoon by receiving position information from each client. Our reasoning for selecting a **P2P** based architecture for **V2V** communication is to ensure that message passing among cars is comparatively faster as it is crucial for all necessary information to be communicated in the least amount of time possible within the platoon.

2.5 Terminology

For the purpose of this document please note that the usage of the terms *car(s)* and *vehicle(s)* are synonymous to the terms *client(s)* or *peer(s)*. Moreover, the term *lead car* refers to the first car in the simulation window. The lead car has an ID number assigned by the server equal to 1. The term *non-lead car(s)* refers to any car following the lead car in the simulation window. The non-lead car(s) has an ID number always greater than 1 depending on the total number of cars in the platoon. Information about how ID numbers are assigned is discussed below. The term *front car*, *immediate front car*, or *car in the front* refers to the car in the simulation window which has an ID lower by one compared to the car under consideration. The front car is in front of the car under consideration towards the direction of the platoon. A similar argument can be made for the *back car*, *immediate back car*, or *car in the back* which has an ID greater by one compared to the car under consideration.

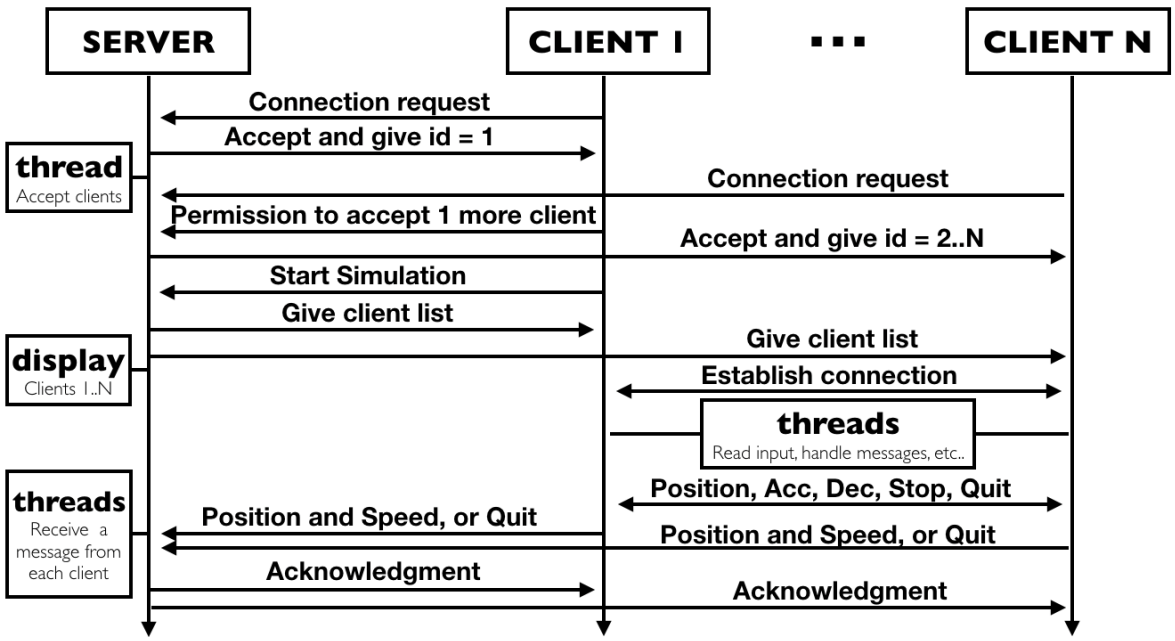


Figure 1: Message passing between server and clients

3 Design and Implementation

3.1 Architecture and Design Decisions

Within the project, we view each vehicle in the platoon and the server as a process running on a computer. In order to mimic the real-life situation, the server cannot directly perform an action to the simulation but may accept an additional client or start the simulation only in accordance with the lead vehicles permission. The user of the lead car may press '**c**' or '**C**' to accept a client or '**s**' or '**S**' to start the simulation. Once the lead car decided to start the simulation, the server sends all clients the list of clients and their information (i.e. ID, address, and port). Given the connection information about the adjacent vehicles, each client then establishes the **P2P** socket connection with its neighbouring clients. We have chosen the **P2P** based architecture over the client/server based architecture for the client-client communication as having message passing through the server will be inefficient due to the bottleneck and therefore will require more time compared to **P2P**.

Figure 1 describes the message passing protocol between the server and each client. The communication between the server and each client is synchronous, as the server acknowledges every successful receive from each client. On the other hand, the client-client communication does not require an acknowledgement from a successful receive, as the messages are sent and received frequent enough that lost or delay of a message does not affect the simulation. Thus, within the client-client communication, frequency is prioritized over accuracy.

Figure 2 provides a detailed view of the entire system. Note that threads are used and assigned for distinct tasks to maximize the concurrency. For any client, it will determine and set up the connection with its immediate front and back vehicles.

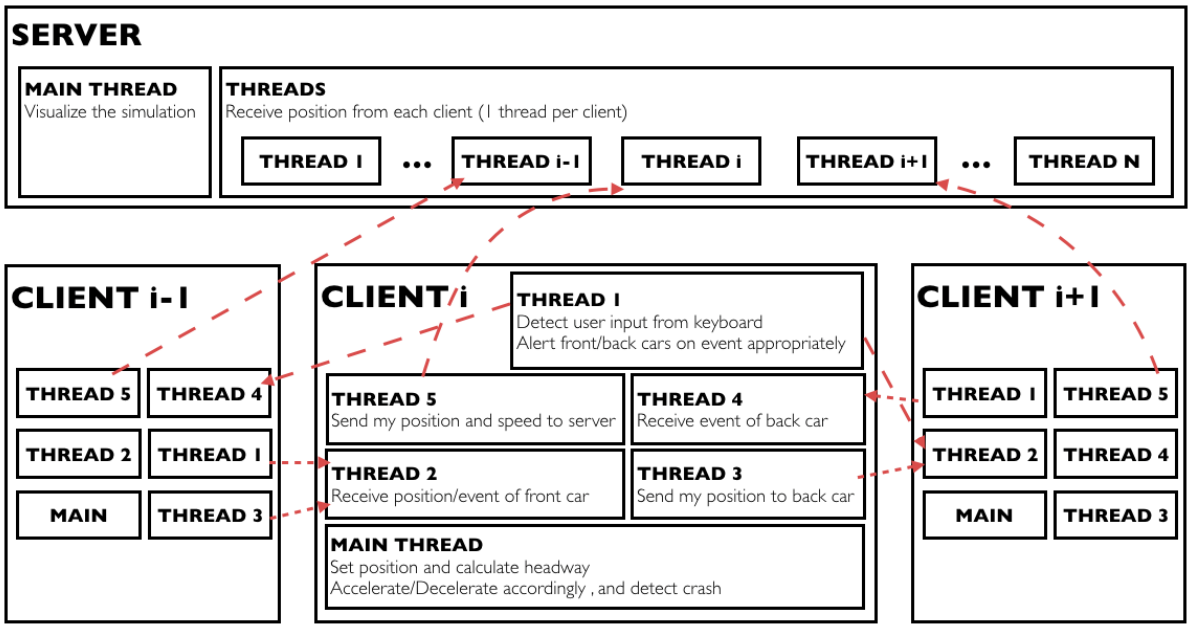


Figure 2: Overview of server and client threads

Each client has six threads including the main thread, where the main thread does not perform any communication but rather is continuously updating position and speed, and detecting the crash according to the global variable of an immediate front vehicle's position. The simulation may end due to the following reasons: a collision of two or more vehicles, a failure of a process, and a proper quit request from a client-side user. Before the main thread exits the program due to the end of the simulation, it messages the server that the simulation is over. Note that all non-main threads have the daemon attribute set to true and that naming of each thread is from the actual implementation.

Thread 1 detects a non-blocking user input from the keyboard, where '**a**' is deceleration, '**s**' is stop, '**d**' is acceleration, and '**q**' is quit. Note that input keys are assigned in terms of the arrows (not the abbreviation of each functionality) and that the user input is not case sensitive. The stop or quit request will be sent to both the front and back car. The acceleration or deceleration request from the lead car directly affects the speed of platoon. However, as the front car has priority over a conflict, acceleration or deceleration of a non-lead car may not affect the speed of platoon sometimes. If a non-lead car attempts to accelerate or decelerate, not violating the appropriate boundary of the headway, it will immediately update its speed accordingly without affecting the speed of platoon. On the other hand, if a non-lead car attempts to accelerate or decelerate from its user input, which causes its headway to be out of the appropriate boundary, it will request its front car to update speed appropriately and update its own speed as well. However, if the request to the front car is not applied (reverted) and thus its headway is out of the boundary, the main thread will automatically update its speed back to how it was.

Thread 2 continuously receives a message from the immediate front car. Note that the lead car will not have Thread 2. The message received from the front is either a position

Figure 3: Immediate Predecessor Communication Structure [10]

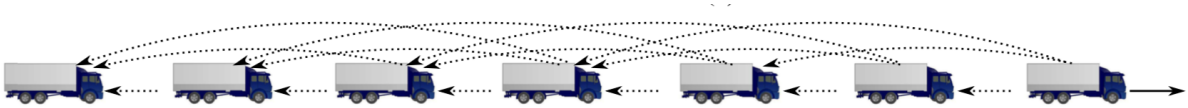


Figure 4: 'r' Predecessor Look Ahead [10]

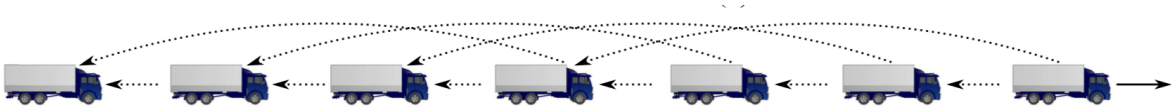


Figure 5: Immediate and 'r'-th Predecessor [10]

of the front car or a request. The request must be either stop or quit, and such requests are propagated to its back car. If the message is a position of the front car, Thread 2 updates the global variable of the immediate front cars position.

Thread 3 accesses the global variable of its current position and continuously sends the position to its immediate back car. Thread 4 receives a request from its back car, which may be accelerating, decelerating and requesting to stop or quit. All requests are directly applied and thus its speed is updated accordingly. However, the speed may be reverted back from the main thread according to the headway of its front car. Note that the last car will not have Thread 3 and Thread 4, as it does not have a car on the back.

Thread 5 accesses the global variables of its current position and speed, and continuously sends them to the server. Each non-main thread of the server is performing a synchronous receive of the position and speed from each client. The main thread of the server visualizes the simulation using pygame.

3.2 Logic of Vehicles

3.2.1 Immediate Predecessor

Darbha, Konduri, and Pagilla (2018) investigates some communication structures for **V2V** communication in the platoon [10]. Figure 3, 4 and 5 shows three such structures. The solid arrow represents the direction of travel and the dotted arrows represent flow of information. For the purpose of the project, we have implemented a modified 'Immediate Predecessor' structure. The 'Immediate Predecessor' defines communication between platooning vehicles in a linear fashion. Here only the lead car controls the platoon and appropriate information is propagated down the platooning vehicles like a 'domino effect'. Communication setup for our project differs in the sense that any car (lead or non-lead) can control the platoon and information is propagated bidirectionally.

The other two communication structures, "*'r'* predecessor look ahead" in figure 4 and "immediate and *'r'*-th predecessor" in figure 5 propagate information to an *'r'*-th predecessor(s). This reduces transmission delay and time taken for the last vehicle in the

platoon to receive information. This is beneficial when the number of platooning vehicles is incredibly large and some vehicles are outside the radius/range of the vehicle that initiated a command.

3.2.2 Linear Car Movement

Acceleration and deceleration of cars in our project occurs in a linear fashion. There is no emphasis on the actual physics-based movement of vehicles involving actual concepts of acceleration, velocity, mass, torque and force. This is done for ease of application and focusing on message passing rather than imitating the movement of actual vehicles. Platoon increases and decreases its speed with a factor of 0.1 and 0.2, respectively. We have tested different values during the simulation and come to terms on this particular value as it provided for smooth graphics.

3.2.3 Front Car Priority

In the event of conflicting operations, the front car is always prioritized. This is because once the simulation has begun acceleration or deceleration of a vehicle occurs continuously in the main thread after calculating the headway whereas acceleration or deceleration in Thread 1 occurs only once when a user initiates them.

For example, consider a situation where a particular car (say car *'v'*) wishes to accelerate (by user input) or has received a message to accelerate from the car behind it (car *'w'*) at the same time it received a message to decelerate from the car in front of it (car *'u'*). The operation by car *'u'* is prioritized in this case. Thus, car *'v'* decelerates and sends an appropriate message to car *'w'*.

Consider another example where car *'v'* accelerates (on user input). If the headway for car *'v'* is smaller than the desired headway, then it would send a message to car *'u'* to accelerate. On receiving this message car *'u'* checks its own headway. If the headway for car *'u'* is also smaller than the desired headway, then it cannot accelerate and therefore it decelerates to maintain the previous platoon speed. This information is received by car *'v'* and as a result, it reduces back to its original speed to avoid a collision.

3.3 Continuous Message Passing

The message passing using sockets in Python does not automatically include the length of a message. When a floating point value (i.e. position or speed) are sent continuously through the socket, a message is received as two or more values merged together. Thus, we have used *json* and *struct* to pack the message in the following format: [length of the message, message]. Furthermore, since Thread 2 of a client receives two different types of a message, a position (a floating point value) and a request (a single character), every message, regardless of the type, must be uniform in formatting.

3.4 Tools

3.4.1 Python 3.6.7

The project was entirely developed using Python 3.6.7. Since it was pre-installed on SUN lab machines, it was ready to use. There was a small learning curve as Python provided all necessary modules for developing the project. The three main modules we have used are *pygame* [7] for visualization, *socket* [8] for **TCP** socket communications, and *threading* [9] for concurrent execution of multiple tasks. Other modules used include *sys*, *traceback*, *json*, *time*, *os*, *struct*, *random*, *errno*, *lock*, *termios*, *tty*, and *math*.

3.4.2 Octave

We have used Octave [5] to plot and numerically evaluate the result of each simulation. As records are made per a message received on the server, use of a scientific computing language that can process large number of data was needed.

3.4.3 Google Drive

We have used Google Drive for sharing resources. Such resources expand from the source code to any submissions, references, and documentation regarding the simulations. It was also used for version control.

4 Results and Evaluation

The sections below analyze twelve successful simulations each of which has different constraints like minimum headway, maximum headway, speed factor and max speed. Note that all simulations are made on SUN lab machines and on different computers via **ssh**. The desired headway for these simulations was 150 (graphical unit of distance or pixel length). The maximum acceptable headway was 151 and the minimum acceptable headway was 150. The reason to have such tight constraints is because of lack of actual physics-based logic of vehicles and continuous message passing. The maximum permissible speed factor of the platoon was 1.1.

4.1 Obtaining Results

For each run of simulations, we have recorded position and speed information in a text file. We have also constructed graphs for speed of the platoon (speed factor), the position of all car, headway of each car versus the number of messages received per each car. This information was stored on the server side after receiving from all clients and before visualizing on the simulation window.

4.2 Evaluation

This section examines and evaluates the data collected during the simulations and provides necessary insights on the working of the project. Simulations 3, 4, 6, 9, 12 were presented during class. As we mentioned in the proposal, each simulation is evaluated on the headway for each car, minimum and maximum headway observed during the simulation.

Simulation Number	Number of Cars	Duration (approx)	#Messages (per client)	Minimum Headway	Maximum Headway
1	4	2 min 5 sec	13,799	144.93	160.09
2	4	1 min 30 sec	14,258	146.90	175.31
3	4	1 min 30 sec	9,697	145.72	155.37
4	2	1 min 20 sec	8,423	149.47	153.45
5	8	1 min 20 sec	8,144	145.90	175.00
6	8	1 min 30 sec	8,410	146.61	158.39
7	7	1 min 10 sec	4,690	144.94	169.81
8	6	1 min 20 sec	8,346	146.01	166.75
9	6	1 min 20 sec	8,349	146.43	158.97
10*	6	1 min 10 sec	7,411	144.18	163.76
11*	9	1 min 30 sec	9,523	147.49	153.62
12	5	30 min	215,585	144.89	171.70

Table 1: Simulation Information

* indicates simulations run with clients on the same computer

4.2.1 Length of Simulation

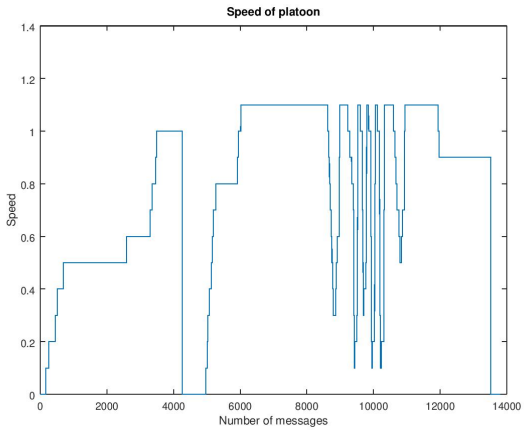
Table 1 shows information on the number of platooning vehicles, duration of the simulation, number of messages received by the server per client and the minimum and maximum headway observed during the course of the simulation. Clearly, simulation 12 has the longest duration and also a notably high maximum headway among the twelve simulations. This maximum headway, however, was not a result of the long duration of the simulation, but rather was resulted from sudden changes in the speed of platoon (from user input). In fact, the maximum headway among the twelve simulations occurred during simulation 2, which has the duration of one and a half minute. From this evaluation, we deduce that the length of simulations does not correlate with the headway.

4.2.2 Co-relation of Speed and Position with Headway

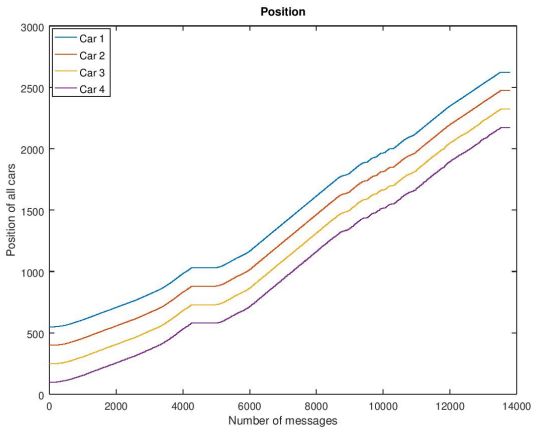
Figure 6 visualizes the speed of the platoon, position of all cars and headway of all cars against the number of messages received by the server per client. Figure 6a shows that the speed of the platoon was constantly increasing and decreasing over the course of the simulation. Highs and lows near 10,000 messages correspond to sudden increase and decrease in the speed. Also, around 4,000 messages, the speed is 0 indicating that the platoon is at a stop.

Figure 6b shows the position of all cars during the course of the simulation. Again it is clear that the plateau around 4,000 messages represents the time of the simulation when the cars were at a halt, and hence the position of all cars is constant. The increase in the position as the cars move forward is shows by the curves.

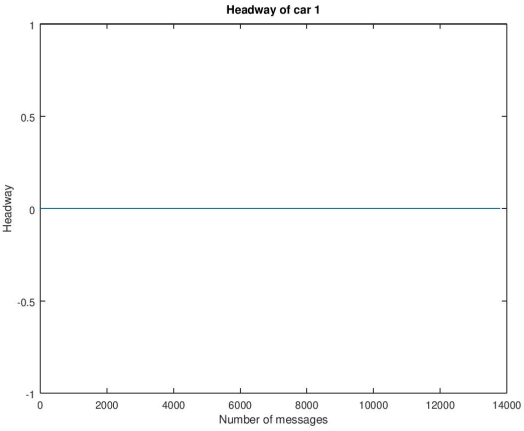
Recall that headway is always with respect to the front car. Figure 6c shows that since the lead car has no car in the front, its headway is always zero. Figures 6d, 6e and 6f represent the headway of the non-lead cars. Again, the flat plateaus in these graphs represent the time of the simulation when the cars were at a halt. Table 3 summarizes the average



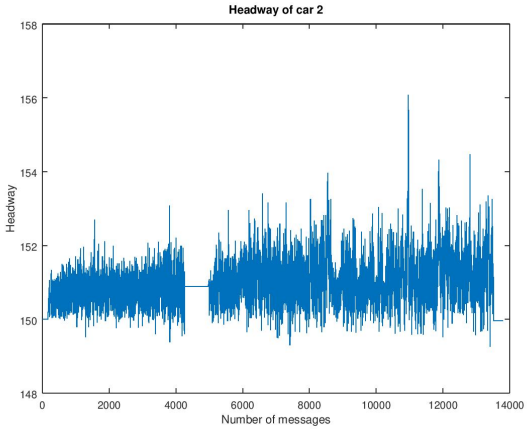
(a) Speed of platoon



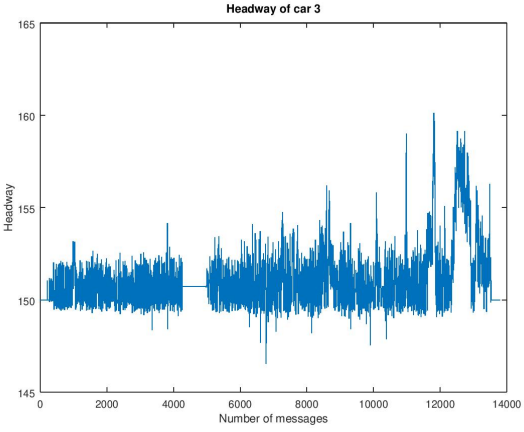
(b) Position of platoon



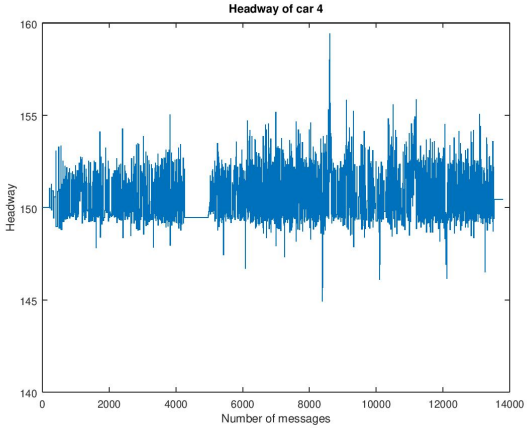
(c) Headway of car 1



(d) Headway of car 2



(e) Headway of car 3



(f) Headway of car 4

Figure 6: Plots of simulation number 1

Duration: 2 min 5 sec, Number of cars: 4

Simulation Number	Car 1	Car 2	Car 3	Car 4	Car 5	Car 6	Car 7	Car 8	Car 9
1	0.00	150.99	151.04	150.39	-	-	-	-	-
2	0.00	150.84	150.87	151.65	-	-	-	-	-
3	0.00	150.82	150.61	150.25	-				
4	0.00	150.90	-	-	-	-	-	-	-
5	0.00	150.82	151.70	150.67	150.17	149.60	149.10	148.24	-
6	0.00	150.78	150.90	150.10	150.40	150.33	149.44	149.19	-
7	0.00	150.72	152.22	150.00	149.89	149.72	148.95	-	-
8	0.00	150.88	150.85	151.50	154.62	149.75	-	-	-
9	0.00	150.77	150.68	150.39	150.03	149.55	-	-	-
10*	0.00	150.77	150.41	149.90	149.48	148.55	-	-	-
11*	0.00	150.53	150.25	150.06	149.92	149.75	149.61	149.45	149.17
12	0.00	150.99	150.86	150.63	150.74	-	-	-	-

Table 3: Average Headway Information during the Simulations

* indicates simulations run with clients on the same computer

headway distances for all cars. It is evident from these graphs that the average headway of the non-lead cars is fairly maintained around the desired headway value. However, notice the spikes outside the normal range in these graphs. These spikes correspond to the time in the simulation when there was a constant and sudden change in the platoon speed. Even though the platoon speed observed sudden changes, any increase or decrease in the headway was compensated over time. This is evident in figure 6f where the headway of the last car reaches almost 160 when the platoon speed suddenly decreases and then increases. However, subsequently, this car adapts to the desired headway by the end of the simulation.

4.2.3 Locality of Vehicles

Note that the clients of simulation 10 and 11 were made on the same SUN lab machine, whereas the clients of all other simulations were made on several different remote machines. Irrespective of the locality of cars running on the same or different machines we have noticed that the headway of the platoon is maintained and is close to our desired headway. Thus, from this evaluation, we deduce that locality of vehicle does not correlate much to the headway.

4.3 Average Headway

Table 3 summarizes the average headway observed by each car during the simulation. Recall that the headway is always with respect to the car in front and so the headway of the lead car is always zero. In the proposal, we have claimed to validate the performance of each simulation based the average headway. Clearly, this table shows that the average headway for any car is fairly close to the desired headway, satisfying acceptable boundary (standard deviation of 2). The maximum and minimum average headway have also been noted in the table.

5 Distributed Issues

5.1 Concurrency

The system maximizes its concurrency by utilizing threads. Each client has six threads including the main thread. The non-main threads handle an excessive number of message passing either with its neighbour clients or with the server. The server has N+1 threads including the main thread, where N is the number of clients. Each non-main thread continuously receives the message from its assigned client and the main thread visualizes the simulation accordingly. The system does not provide concurrency transparency as the clients are aware of the other concurrent (neighbouring) users.

5.2 Failure Handling

A failure of any process involved in the simulation is immediately noted by all other processes. The current implementation of the system has N points of failures, where N is the total number of processes involved in the simulation (the server and all clients). When a process detects a failure (i.e. disconnection) of a connected process, it immediately sends an appropriate message to other clients to stop and quit the simulation and exits the program. Thus, the system does not provide fault transparency, a process is aware of the failure of one another.

5.3 Scalability

The server may accept any number of clients (vehicles in simulation), however, any non-lead client must wait for the lead clients permission. Due to the widths of the display and the car image, the current implementation can hold up to nine vehicles per simulation, which can be easily increased or decreased by changing the size of car image.

5.4 Synchronization

Both the clients and the server make use of global variables for concurrent access and update of each thread. The three main usages of the global variables are to have a read-only fixed constant for all threads, to detect the end of the simulation and to update the speed and position of vehicles. Any update on global variables is synchronized using locks, the Python synchronization primitive.

6 Future Considerations

The sections below describe some improvements that can be made towards the current state of the system. Note that the followings are not yet implemented due to the limited time and scope of the project but are necessary to have a realistic and stable platoon simulation.

6.1 Better Scalability

Realistically, automobile platoon simulation should involve multiple lanes of vehicles, each of which can not only shift lanes but also merge or split into multiple different

platoons. Having multiple lanes of vehicles each performing a separate platoon can be easily done. However, implementing the four additional operations for each vehicle will be cumbersome, and furthermore, may result in less consistent headway distances between each vehicle.

6.2 Sophisticated Car Logic

Clearly, our current implementation of platoon simulation uses a linear fashion of car movements and thus the vehicles visualized during a simulation are slightly unrealistic. Implementation of actual physics into the car movement (i.e. position, speed, velocity, and acceleration) will resolve the bumping issue of the vehicles. Furthermore, advancement in vehicle-to-vehicle communication may improve the average headway distances overall. As of our current implementation, it uses immediate predecessor protocol, which has shown to be less performant than the other two communication protocols discussed above.

6.3 Failure Handling

To improve reliability, the system should proceed to simulate despite the failure of one or more vehicle processes. Currently, the simulation stops and exits immediately after a failure from any process are detected. For the future, the simulation should only stop and exit when the server process fails. The solutions to maintain the constant headway within a failure of one or more client process are either to split them into two separate platoons or to implement an appropriate acceleration scheme when a failure is detected.

6.4 State of Platoon

In the real world, vehicles should be able to freely join and leave the platoon. Since the server receives both the position and the speed of each vehicle, it can easily find the state of platoon–whether platoon is ongoing or on stop state. Using such information, the server should be able to accept more client connections, again in accordance with the lead cars permission.

7 Conclusion

While our project is by no means perfect and does not describe the real-time scenario, it does address the problem at hand. Even though there were a number of issues faced during the project, this project goes to show that it is possible to develop a communication protocol to implement platoons with an ITS. In summary, this project is a proof of concept for implementing real-life platoons and addresses the problem of communication in a platoon.

References

[1] Vinel A., Lin Lan L., and Lyamin N. "Vehicle-to-vehicle communication in C-ACC/platooning scenarios". In: *IEEE Communications Magazine* 53 (8) (2015), pp. 192–197. URL: https://ieeexplore.ieee.org/abstract/document/7180527.

[2] National Highway Traffic Safety Administration. *Automated Vehicles for Safety*. 2019. URL: https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety.

[3] Dan Van Boxel. *Collaborative project on self-driving car simulation*. 2016. URL: https://github.com/dvbuntu/oddities.

[4] A.C. Brown et al. "Vehicle to vehicle communication outage and its impact on convoy driving". In: *Proceedings of the IEEE Intelligent Vehicles Symposium 2000 (Cat. No.00TH8511)* (2000), pp. 528–533. URL: https://ieeexplore.ieee.org/document/898399.

[5] John W. Eaton. *Octave*. URL: https://www.gnu.org/software/octave/.

[6] Benalla M., Achchab B., and Hrimech H. "A Distributed Intelligent System for Emergency Convoy". In: *International Journal of Interactive Multimedia and Artificial Intelligence* 4 (2016), pp. 42–45. URL: https://pdfs.semanticscholar.org/f534/b972294f20bb77b08d7a8500574a18cc55db.pdf.

[7] *Pygame Documentation*. URL: https://www.pygame.org/docs/.

[8] *Python Socket Module: Low-level networking interface*. URL: https://docs.python.org/3.6/library/socket.html.

[9] *Python Threading Module: Thread-based parallelism*. URL: https://docs.python.org/3.6/library/threading.html?highlight=threading#module-threading.

[10] Darbha S., Konduri S., and Pagilla P.R. "Benefits of V2V Communication for Autonomous and Connected Vehicles". In: *IEEE Transactions on Intelligent Transportation Systems* (2018), pp. 1–10. URL: https://ieeexplore.ieee.org/document/8463512.

[11] Wikipedia. *Intelligent Transportation System*. 2019. URL: https://en.wikipedia.org/wiki/Intelligent_transportation_system.

[12] Wikipedia. *Platoon (Automobile)*. 2019. URL: https://en.wikipedia.org/wiki/Platoon_(automobile).

[13] Wikipedia. *Self-driving Car*. 2019. URL: https://en.wikipedia.org/wiki/Self-driving_car.

Glossary

AVCSS Advanced Vehicle Control and Safety Systems. 4

CACC Cooperative Adaptive Cruise Control. 4

ID Identification Number. 4, 5

ITS Intelligent Transportation System. 3, 4

P2P Peer-to-peer. 3–5

ssh Secure Shell. 9

TCP Transmission Control Protocol. 3, 9

V2V Vehicle-to-vehicle. 3, 4, 7