Truck Platooning System Implementation using Master Slave Architecture

Akash Cuntur Shrinivasmurthy
Matriculation No : 7219642
Embedded Systems Engineering
Fachhochschule Dortmund
Dortmund, Germany
akash.cuntursshrinivasmurthy001@stud.fh-dortmund.de

Akhil Narayanaswamy
Matriculation No : 7219528
Embedded Systems Engineering
Fachhochschule Dortmund
Dortmund, Germany
akhil.narayanaswamy001@stud.fh-dortmund.de

Madhukar Devendrappa
Matriculation No : 7219639
Embedded Systems Engineering
Fachhochschule Dortmund
Dortmund, Germany
madhukar.devendrappa002@stud.fh-dortmund.de

Swathi Chandrashekaraiah
Matriculation No : 7218877
Embedded Systems Engineering
Fachhochschule Dortmund
Dortmund, Germany
swathi.chandrashekaraiah001@stud.fh-dortmund.de

Vipin Krishna Vijayakumar
Matriculation No : 7219307
Embedded Systems Engineering
Fachhochschule Dortmund
Dortmund, Germany
vipin.vijayakumar001@stud.fh-dortmund.de

Abstract—**Truck platooning is a technique where multiple trucks travel in a close formation, using advanced communication and automation systems to maintain safe distances and synchronize movements. It offers benefits such as improved fuel efficiency, mitigate traffic congestion, reduced emissions, and enhanced road safety by optimizing convoyed vehicles' aerodynamics and coordination. In the implemented truck platooning system, communication is exclusively facilitated between the leader and follower trucks, focusing on exchanging crucial information regarding the time and position of each vehicle at regular time intervals. Join requests from follower trucks trigger the leader to allocate dedicated slots and accept these requests, rejecting duplicate IDs if a request is received from an already coupled follower truck. The system incorporates a robust communication failure handling mechanism, attempting re-establishment over a 15-unit time span. If communication remains unresolved beyond this timeframe, the follower trucks are automatically decoupled to ensure safety. Both leader and follower trucks undergo rigorous time synchronization. GPU calculations, employing CUDA and PID techniques, contribute to maintaining optimal inter-vehicle distances within the platoon. This comprehensive truck platooning system represents a transformative leap in transportation, offering synchronized fleets that significantly reduce fuel consumption, enhance operational efficiency, improve road safety, and contribute to a more sustainable and efficient future.**

Keywords—Platooning, fuel efficiency, road safety, CUDA, PID

I. INTRODUCTION

In recent years, the transportation industry has witnessed significant advancements in autonomous driving technology, revolutionizing the way goods are transported over long distances. One of the promising innovations in this domain is truck platooning, a technique where multiple trucks travel in a close formation, often referred to as a platoon, with the aid of advanced communication and control systems.

By leveraging vehicle-to-vehicle (V2V) communication and advanced driver assistance systems (ADAS), truck platooning enables convoyed trucks to maintain precise distances and synchronize their movements, thereby reducing aerodynamic drag and fuel consumption. These fuel savings translate into considerable cost reductions for fleet operators and contribute to environmental sustainability by lowering carbon emissions associated with freight transportation.

A study commissioned by the European Commission in 2016 found that truck platooning could potentially lead to fuel savings of up to 10% for following trucks and around 4.5 % for leading trucks compared to conventional driving methods. These findings were part of the European Truck Platooning Challenge initiative aimed at promoting the development and deployment of platooning technology. Research conducted by the US Department of Energy in 2017 indicated that truck platooning could result in fuel savings ranging from 4% to 10% for following vehicles, depending on factors such as speed, distance, and traffic conditions. The study highlighted the potential of platooning to improve fuel efficiency in long-haul trucking operations across various road networks.

Furthermore, truck platooning has the potential to alleviate traffic congestion and enhance road capacity by optimizing the utilization of existing infrastructure. Through synchronized movements and reduced inter-vehicle spacing, platooning enables convoyed trucks to travel more efficiently, thereby reducing the likelihood of traffic bottlenecks and gridlock. With its ability to enhance safety through automated braking and collision avoidance systems, truck platooning aims to reduce the frequency and severity of accidents, thereby improving overall road safety metrics.

A report published by the ATRI in 2018 explored the impact of truck platooning on traffic congestion and roadway capacity. While specific quantitative data may vary, the report suggested that implementing truck platooning on major highways could lead to a notable increase in roadway throughput and a reduction in travel times for both commercial and passenger vehicles. These findings underscored the potential of platooning to mitigate traffic congestion and improve overall traffic flow efficiency. A study conducted by the Transport Research Laboratory in 2019 examined the effects of truck platooning on traffic congestion in urban and suburban areas. While the study focused primarily on highway scenarios, it also considered the potential spillover effects of reduced congestion from highways to surrounding road networks. Although specific data may vary by location and context, the study suggested that truck platooning could contribute to overall traffic congestion reduction by optimizing freight movements and enhancing roadway capacity.

II. Motivation

The motivation for initiating a truck platooning project stems from a comprehensive understanding of the challenges facing the freight transportation industry and the recognition of the potential benefits that truck platooning can offer. Below are some key motivating factors:

Efficiency Enhancement: Long-haul trucking operations are often associated with high fuel consumption and operating costs. Truck platooning presents an opportunity to significantly improve fuel efficiency by reducing aerodynamic drag and optimizing convoyed vehicles' coordination. This translates to cost savings for fleet operators and a more sustainable transportation ecosystem.

Traffic Congestion Reduction: Congested roadways not only lead to delays but also increase fuel consumption and emissions. Truck platooning has the potential to alleviate traffic congestion by optimizing the flow of goods on highways. By maintaining consistent speeds and reducing unnecessary lane changes, platoons can contribute to smoother traffic flow and reduced congestion for all road users.

Safety Enhancement: Truck accidents pose significant risks to drivers, passengers, and other road users. Truck platooning systems incorporate advanced safety features such as automated braking, collision avoidance, and adaptive cruise control to mitigate the risk of accidents. By maintaining safe following distances and synchronizing movements, platoons enhance overall road safety and reduce the likelihood of collisions.

Driver Well-being: Long hours of driving can lead to driver fatigue and increase the risk of accidents. Truck platooning technology can help alleviate driver fatigue by sharing the driving workload among convoyed vehicles. This allows drivers to take breaks and rest periods while the platoon continues to progress toward its destination, thereby improving driver well-being and reducing the incidence of fatigue-related accidents.

Environmental Sustainability: The transportation sector is a significant contributor to greenhouse gas emissions and air pollution. Truck platooning offers environmental benefits by reducing fuel consumption and emissions per ton-mile of freight transported. By promoting the adoption of cleaner and more efficient transportation practices, truck platooning contributes to broader sustainability goals and mitigates the impacts of climate change.

Overall, the motivation for a truck platooning project lies in its potential to address pressing challenges in the freight transportation industry, improve operational efficiency, enhance road safety, promote environmental sustainability, and drive technological innovation for a more efficient and sustainable future.

III. Sketch of Approach

The analysis of the model has taken place an important role to design the required model. The tasks are done in the following alphabetical order to complete the whole project from analyzing to testing

A. *Requirement Diagram*

Requirements refer to the specifications, features, capabilities, and constraints that a system, software application, or project must satisfy to meet its intended purpose and stakeholders' needs.

A requirement diagram is a visual representation of the requirements for a system, software application, or project. It organizes and illustrates the relationships between various requirements, helping stakeholders understand the scope of the project, identify dependencies, and prioritize features. By depicting relationships between requirements, requirement diagrams help stakeholders understand how different features and functionalities are interconnected. This facilitates better decision-making, risk assessment, and change management throughout the project lifecycle. It serves as a communication tool for conveying complex requirements in a visual format. They enable stakeholders from different backgrounds and disciplines to collaborate effectively, discuss trade-offs, and resolve conflicts or ambiguities in the requirements.

Each requirement is created with unique ID, and they got incremented when the corresponding requirement get updated. For this truck platooning project, requirements blocks are created and its structural appearance is shown in Fig. 1

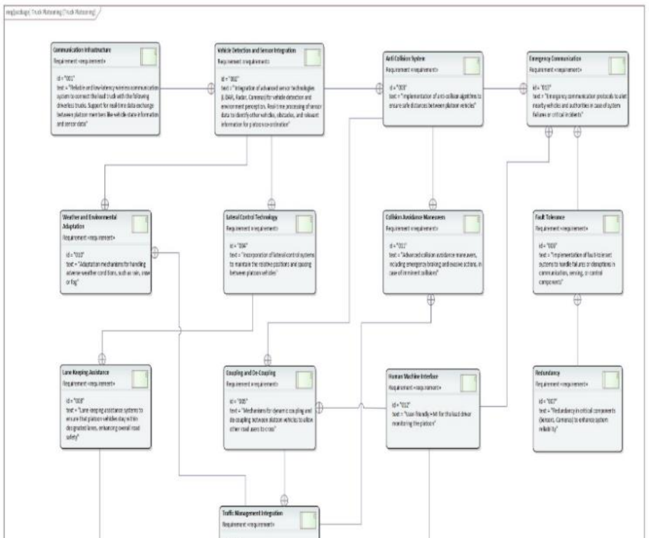

Communication Infrastructure
Vehicle Detection and Sensor Integration
Anti Collision System
Emergency Communication
Weather and Environmental Adaptation
Lateral Control Technology
Collision Avoidance Maneuvers
Fault Tolerance
Lane Keeping Assistance
Coupling and De-Coupling
Human Machine Interface
Redundancy
Traffic Management Integration

Fig. 1. Requirement Diagram of Truck Platooning

The truck platooning system is equipped with a comprehensive set of requirements to ensure its effectiveness, safety, and adaptability to various scenarios. Firstly, it features a robust communication infrastructure that provides reliable and low-latency wireless connectivity, facilitating real-time data exchange between the lead truck and the following driverless trucks. The integration of advanced sensor technologies, including LiDAR, Radar, and Cameras, forms a crucial aspect of the system. These sensors enable vehicle detection and environment perception, with real-time processing capabilities to identify other vehicles, obstacles, and relevant information essential for platoon coordination.

The system incorporates anti-collision algorithms to maintain a safe distance between platoon vehicles, promoting overall safety on the road. Additionally, lateral control technology is implemented to ensure the proper maintenance of relative positions and spacing between platoon vehicles, contributing to smooth and coordinated movement. Dynamic coupling and de-coupling mechanisms are integrated, allowing platoon vehicles to dynamically adjust their formations to enable the crossing of other road users, enhancing overall road flexibility.

Fault tolerance is a key consideration in the system, addressing potential disruptions in communication, sensing, or control components. Redundancy measures are implemented in critical components, such as sensors and cameras, to enhance overall system reliability. Lane keeping assistance systems are employed to ensure that platoon vehicles stay within designated lanes, contributing to enhanced road safety.

Furthermore, the system integrates with traffic management systems, optimizing platoon routes based on real-time traffic conditions to minimize congestion and improve overall efficiency. It is equipped with adaptation mechanisms to handle adverse weather conditions, such as rain, snow, or fog, ensuring consistent performance in various environments.

In terms of safety maneuvers, advanced collision avoidance measures, including emergency braking and evasive actions, are incorporated in the system. A user-friendly Human-Machine Interface (HMI) is designed for the lead driver who monitors the platoon, providing an intuitive and accessible interface. Additionally, the system includes emergency communication protocols to promptly alert nearby vehicles and authorities in the event of system failures or critical incidents. Overall, these requirements collectively form a comprehensive framework for a sophisticated and reliable truck platooning system.

B. *State Machine Diagram for Communication*

A State Machine Diagram is a behavioral diagram that represents the dynamic behavior of a system or an entity by depicting its states, transitions between states, and the events that trigger these transitions. State machines are particularly useful for modeling systems that exhibit different behaviors based on their internal states.
These are key components and concepts related to State Machine Diagrams,
State: - A state represents a condition or situation in which a system or an entity can exist. It could be a stable condition where the system performs specific activities until a triggering event occurs.
Transition: - Transitions represent the movement from one state to another. Transitions are triggered by events and may have associated conditions or actions that are executed when the transition occurs.
Event: - An event is an occurrence that triggers a transition from one state to another. Events can be internal or external and may include signals, changes in conditions, or user actions.
Action: - Actions are activities or operations associated with a state or a transition. These represent the behavior or functionality that occurs when a system is in a particular state or when a transition takes place.
Initial State: - The initial state indicates the starting point of the state machine, representing the state the system is in when it begins its operation.
Final State: - The final state represents the conclusion or termination of the state machine. It is the state the system enters when it has completed its activities or when a specific condition is met.
Guard Condition: - Guard conditions are optional conditions associated with transitions. These conditions determine whether a transition is taken based on the evaluation of certain criteria.
History State: - A history state indicates that the system should resume its operation from a specific point, depending on its past behavior.

State Machine Diagrams are useful for modeling complex systems where the behavior is not solely determined by the current input, but also by the system's history and past states. They are commonly employed in software engineering, control systems, and embedded systems design to represent the dynamic behavior of software, hardware, or combined software-hardware systems. State Machine Diagrams provide a clear and visual representation of how a system responds to different events and how it transitions between various states during its operation. The State Machine for Communication is given in Fig. 2 where we have considered TCP over UDP because we will get acknowledgement in TCP and this will be really helpful to detect the node failure. As we are implementing this in windows, we are using WINSOCK.

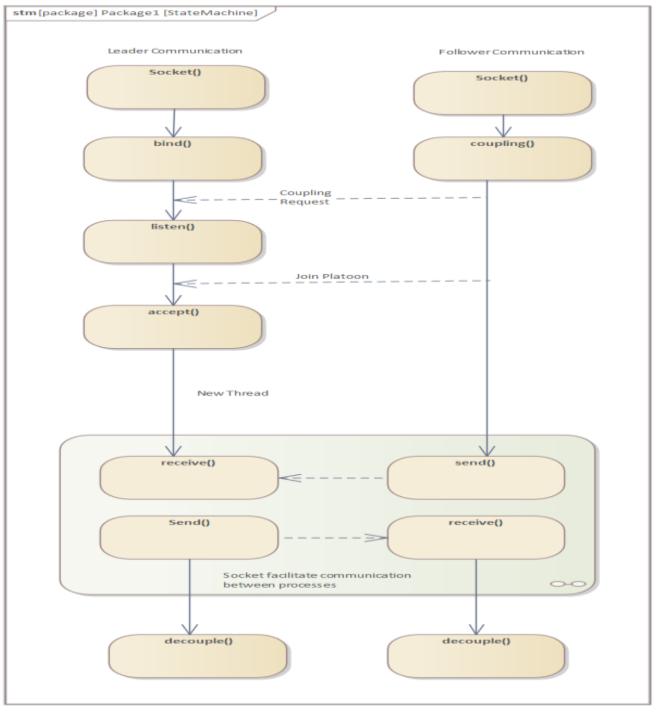


Fig. 2. State Machine Diagram of Communication

First, we have a socket, which will get connected to a port. we have used a local host which is on the port 8080. The leader will be constantly listening for the request messages from the follower. Through coupling request the follower will be sending its ID. Then leader will send back its ID to the follower. After this step the follower will send request for joining the platoon with its position and destination. If the leader and follower share the same destination, the request will be accepted and a slot will be assigned to the following truck and a new thread will be created to communicate between leader and following truck. Leader will be sending the expected position of follower truck each second. Using PID and Obstacle detection, follower will try to achieve the expected position. Follower will only send messages when it wants to leave platoon or during emergency situations. When leave platoon message is received or communication error occurs then it will lead to decoupling.

C. *Activity Diagram*

An Activity Diagram is a graphical representation that models the dynamic aspects of a system. It is used to describe the flow of activities, actions, and control flow within a system or a specific process. Activity Diagrams are part of the behavioral diagram category and are particularly useful for capturing the workflow and behavior of a system during runtime. Activity Diagrams are valuable for visualizing the dynamic aspects of a system, helping stakeholders understand the sequence of activities and their interactions. They are commonly used during the analysis and design phases of system development to model business processes, use cases, or any other dynamic aspects of a system.

An activity represents a unit of work or a set of actions performed within the system. It can range from simple operations to complex processes. Actions represent atomic steps or operations within an activity. These can include calculations, data manipulations, or any other executable behavior. Control flow arrows show the sequence and order of activities and actions. They indicate the flow of control from one element to another. Decision nodes, often represented by diamonds, are used to model decision points in the process. Depending on certain conditions, the control flow can take different paths. Merge nodes, also represented by diamonds, indicate points where multiple control flows converge back into a single flow. Fork nodes represent points in the process where a single control flow splits into multiple parallel flows, indicating concurrent activities. Join nodes indicate synchronization points where multiple parallel flows converge back into a single flow. Object flows represent the flow of objects between different activities. This can show how data or objects are passed between actions and activities. Swim lanes are used to partition the diagram to represent different actors or subsystems involved in the activities. Each swim lane typically contains the actions and activities associated with a specific role or system component. Every activity diagram has a starting node and an ending node, indicating the initiation and completion of the activity.

The activity diagram of Truck platooning is, initially the following truck initiates the communication to the lead truck, there will be an ID associated with each following truck. If the ID of the truck exists then the request is denied. The Truck ID exists if it is already connected to the leader. In that case the request is denied. If it doesn't exist the following truck will be assigned a slot (like a position with respect to the leading truck). Then the time synchronization part will be executed by the logical matrix clocks. From this stage the next processes will be functioning in a cycle with respect to certain conditions. If the leading truck decide to leave the platoon formation, then the process will be directed to the Decouple stage. Otherwise, communication status is monitored in the next step and if there is no communication error the process is redirected to the time synchronization part and the process will continue in a cycle until there is a communication error or a decoupling situation. If there is a communication error the system will try to reconnect for 15 seconds and if it is not established after this period then there will be a decoupling otherwise the system will continue from the time synchronization steps. The process is same for all the following trucks in the platoon. The diagram is shown in Fig. 3.

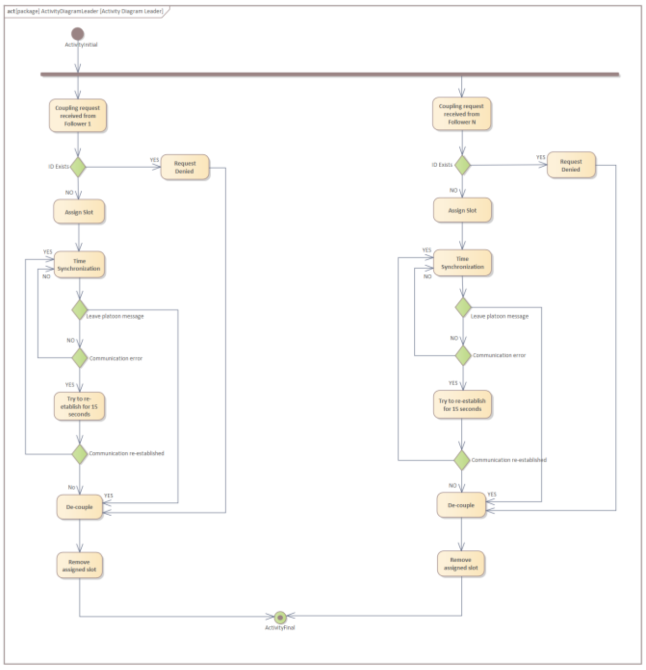

act [package] ActivityDiagramLeader [Activity Diagram Leader]
ActivityInitial
Coupling request received from Follower 1
Coupling request received from Follower N
ID Exists
YES
NO
Request Denied
Assign Slot
Time Synchronization
Leave platoon message
Communication error
Try to re-establish for 15 seconds
Communication re-established
De-couple
Remove assigned slot
ActivityFinal

Fig. 3. Activity Diagram of Leading Truck

The second activity diagram describe the dynamic aspects of following truck. For the following Truck, initially the communication is established, then if the leader is identified the request for joining platoon is sent. If the leader is not found the process is redirected to the communication establishment step. Next step is also a condition where we check whether the leader has sent a duplicate ID or not. A Duplicate ID is sent by the leader if the truck is already in the Platoon and it is sending request again for connecting. If it is not identified as a duplicate ID then Time synchronization process will be started as similar to the leading truck. The leading trick will be sending the expected position in every second to the following truck and in parallel to that the Anti-collision system and the lane change will be running. We will be using PID for the acceleration control. This will ensure the proper maintenance of speed with respect to the leading truck.

This will run in a loop until there is a communication fail. If there is a communication fail there is a waiting time of 15 seconds, at this time the trucks will try to re couple. If the communication is not established in 15 seconds, then the system will decouple. If the communication didn't face any issues, then we will check whether we have reached the destination. If the destination is reached then the system will decouple, otherwise the system will continue in the cycle from time synchronization stage. In case of a duplicate ID the system will be shifted to the decouple stage directly. The Activity Diagram is shown in the Fig. 4

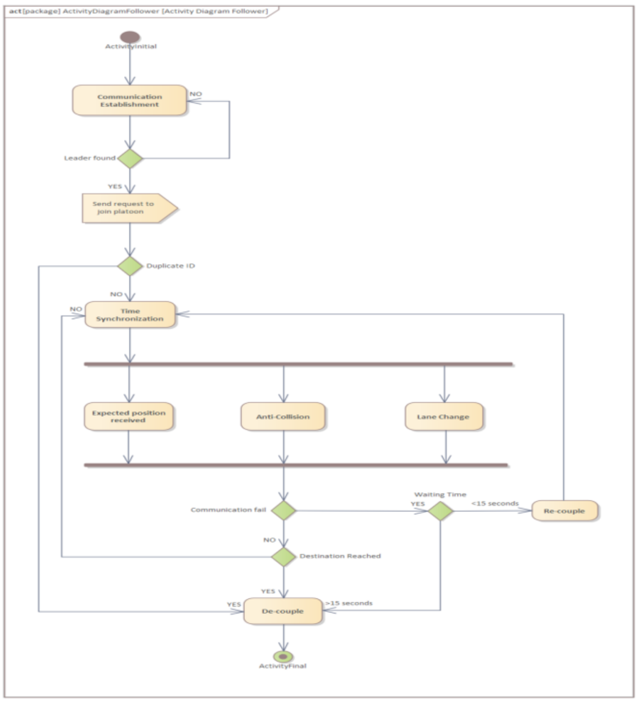


Fig. 4. Activity Diagram of Following Truck

III. IMPLEMENTATION

The truck platooning system comprises two main components: the leader truck controller and the follower truck controller. The leader truck takes on the role of managing the platoon, calculating positions, and communicating with individual followers. Followers, on the other hand, synchronize with the leader and adjust their positions accordingly.

A. *System Architecture*

The Truck Platooning System adopts a master-slave model, leveraging the leader truck as the master to direct follower trucks. This model's hierarchical structure, based on a client-server paradigm, centralizes control and simplifies coordination. With a clear hierarchy, the leader efficiently disseminates instructions, reducing decision-making complexity. Additionally, the model promotes enhanced synchronization among followers, ensuring cohesive and simultaneous operations and optimizing the overall platooning process.

In terms of communication protocols, the system relies on the TCP communication protocol facilitated by WinSock2. This choice is motivated by TCP's reliability, which makes providing ordered and error-checked information delivery critical for maintaining accurate and synchronized data exchange. The connection-oriented nature of TCP establishes a dedicated channel between the leader and followers, enhancing communication stability and reducing the likelihood of data loss or corruption during transmission. Furthermore, TCP WinSock2 incorporates efficient error-handling mechanisms, responding adeptly to network issues to ensure uninterrupted communication. The protocol's widespread compatibility across various platforms and operating systems underscores its versatility, making it an apt choice for communication in diverse environments.

B. *Truck class*

The "Truck" class, represent a generic truck in the platooning system, encompasses pivotal attributes, including ID, state (IDLE, CRUISE, ACCELERATE, DECELERATE, STOP), speed (providing the vehicle's speed), position (for locating the trucks), size (representing the length of the truck), and gap (representing the distance maintained between the trucks during platooning). These attributes collectively define the truck's identity, operational status, and dynamic parameters.

The "Leader-Truck" class, extending the Truck class, introduces methods like "addFollower()" and "removeFollower()" for efficient platoon management. Beyond the generic attributes, the Leader-Truck class dynamically handles followers within the platoon, allowing the leader to adapt to changing scenarios.

Similarly, the "Follower-Truck" class, also an extension of the Truck class, includes specialized methods like "setLeader()" and "getLeader()." These methods enhance communication and coordination within the platoon by facilitating connections with designated leaders and retrieving leader information.

C. *Leader Truck Controller*

1) *Leader Initialization*

The Leader Truck initializes with a unique identifier (ID) set to "LTRK012" and an initial position at coordinates (500, 1). This initialization signifies the distinctive identification and initial positioning of the leader truck within the broader platooning system.

2) *Logical Clock Thread*

A detached thread is started to increment the logical clock for the leader in the background, ensuring accurate time synchronization within the platoon.

3) *Winsock Setup*

Following the leader's initialization, the code proceeds to set up Winsock, the Windows Socket API. The success of this initialization is verified using the WSAStartup function. In the event of a failure, an error message is displayed, and the program exits with a failure status.

4) *Server Socket Creation*

The server socket is created using the socket function, specifying the address family as AF_INET and the socket type as SOCK_STREAM. If the creation of the server socket fails, the code prints an error message, and the program exits.

5) *Binding to the IP Address and Port*

The server socket is then bound to a specific IP address (INADDR_ANY) and port number (PORT) using the bind function. Any issues encountered during the binding process prompt the display of an error message, and the program exits.

6) Listening for Incoming Connections

Subsequently, the server initiates listening for incoming connections using the listen function. If the listening process encounters an issue, an error message is printed, and the program exits.

7) Accepting Client Connections

The code enters a continuous loop to accept incoming client connections using the accept function. For each accepted connection, a new thread is created to handle the client independently. The newly created thread is detached, allowing it to run independently of the main program.

8) Handling Follower Connections

Continuously monitoring for incoming connection requests from follower trucks, the leader initiates a new thread for each accepted connection, ensuring independent communication with each follower.

9) Follower Join Requests

Upon receiving a join request from a follower, the leader extracts the follower's ID and position. The request undergoes validation, checking for duplicate entries in the leader's list. If the follower is not a duplicate, it is added to the leader's list. In the case of a duplicate, an error message is sent to the client, and the connection terminates.

10) Position Calculation and Handling

After adding a new follower to the list, the leader prints information about the join request and calculates the position that needs to be sent to the follower. This calculation is based on the leader's position and the follower's position, ensuring accurate and synchronized positioning within the platoon.

11) Main Loop

Entering a loop to handle ongoing communication with the client, the leader:

- Periodically sends update messages to the client, including logical clock information and the expected position based on the leader's state.
- Monitors for communication errors and attempts to reconnect with the client.

12) Handling Communication Errors and Follower Removal

Within the main loop, the code monitors for issues in sending data, such as socket errors or client disconnection. It handles recoupling attempts if there's a communication failure (up to 15 seconds). If such issues persist, the code handles the error by removing the follower from the leader's list and exiting the loop. This ensures the integrity of the platooning system.

13) Socket Closure

Finally, upon exiting the main loop, the leader closes the socket using closesocket. This marks the end of the server-client interaction for the given client.

D. Follower Truck Controller

The follower truck controller complements the leader by managing individual follower trucks. This section outlines the key features and functionalities implemented for effective follower participation.

1) Follower Initialization

Each Follower Truck is initialized with a dynamically assigned unique identifier (ID) and an initial position. Additionally, a thread is created to increment its logical clock, contributing to the overall time synchronization within the platoon.

2) Communication Setup

Similar to the leader, the follower utilizes Winsock2 for communication. A client socket is created to connect with the leader, and the server details, such as IP address and port, are configured.

3) Handling Leader Greeting

Upon connecting to the leader, the follower receives a greeting message containing the leader's ID. The follower processes this message to identify the leader and checks if the received data starts with the prefix "leader." If successful, the follower couples with the identified leader and sends a "join" request to the leader.

4) Joining the Platooning System

Once identified, the follower sends a join request to the leader, including its ID and initial position. This request establishes the follower's presence in the platoon.

5) Receiving and Processing Updates

The follower enters a continuous loop to receive updates from the leader. The updates contain crucial information about the platooning system, allowing the follower to adjust its position accordingly.

The follower updates

6) Reconnection Mechanism

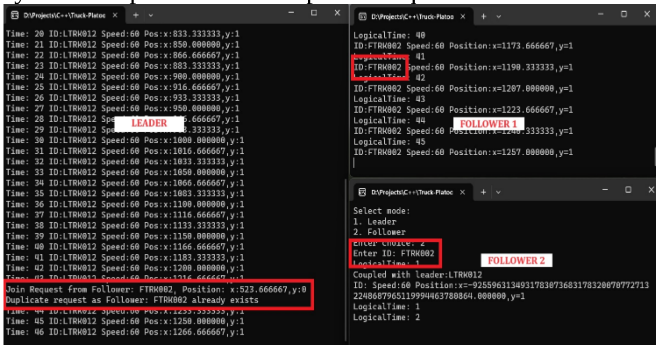
In cases of communication failure or disconnection from the leader, the follower attempts to reconnect. If reconnection fails after a certain number of attempts, the follower removes the leader, and the loop terminates once the client socket is closed.

7) Overall Flow

The function continuously listens for updates from the leader, synchronizes clocks, and attempts to reconnect in case of a failure. It provides a mechanism for handling disconnection scenarios.

E. Duplicate Request Handling

One critical scenario was the handling of duplicate requests, where a follower with an existing ID attempt to join the leader. Fig. 5 illustrates this scenario, showcasing the system's response when a duplicate request is detected.


D:\Projects\C++\Truck-Platoo
LEADER
D:\Projects\C++\Truck-Platoo
FOLLOWER 1
D:\Projects\C++\Truck-Platoo
FOLLOWER 2

```
Time: 20 ID:LTRK012 Speed:60 Pos:x:833.333333,y:1
Time: 21 ID:LTRK012 Speed:60 Pos:x:850.000000,y:1
Time: 22 ID:LTRK012 Speed:60 Pos:x:866.666667,y:1
Time: 23 ID:LTRK012 Speed:60 Pos:x:883.333333,y:1
Time: 24 ID:LTRK012 Speed:60 Pos:x:900.000000,y:1
Time: 25 ID:LTRK012 Speed:60 Pos:x:916.666667,y:1
Time: 26 ID:LTRK012 Speed:60 Pos:x:933.333333,y:1
Time: 27 ID:LTRK012 Speed:60 Pos:x:950.000000,y:1
Time: 28 ID:LTRK012 Spe            .666667,y:1
Time: 29 ID:LTRK012 Spe            .333333,y:1
Time: 30 ID:LTRK012 Speed:60 Pos:x:1000.000000,y:1
Time: 31 ID:LTRK012 Speed:60 Pos:x:1016.666667,y:1
Time: 32 ID:LTRK012 Speed:60 Pos:x:1033.333333,y:1
Time: 33 ID:LTRK012 Speed:60 Pos:x:1050.000000,y:1
Time: 34 ID:LTRK012 Speed:60 Pos:x:1066.666667,y:1
Time: 35 ID:LTRK012 Speed:60 Pos:x:1083.333333,y:1
Time: 36 ID:LTRK012 Speed:60 Pos:x:1100.000000,y:1
Time: 37 ID:LTRK012 Speed:60 Pos:x:1116.666667,y:1
Time: 38 ID:LTRK012 Speed:60 Pos:x:1133.333333,y:1
Time: 39 ID:LTRK012 Speed:60 Pos:x:1150.000000,y:1
Time: 40 ID:LTRK012 Speed:60 Pos:x:1166.666667,y:1
Time: 41 ID:LTRK012 Speed:60 Pos:x:1183.333333,y:1
Time: 42 ID:LTRK012 Speed:60 Pos:x:1200.000000,y:1
Time: 43 ID:LTRK012 Speed:60 Pos:x:1216.666667,y:1
Join Request from Follower: FTRK002, Position: x:523.666667,y:0
Duplicate request as Follower: FTRK002 already exists
Time: 44 ID:LTRK012 Speed:60 Pos:x:1233.333333,y:1
Time: 45 ID:LTRK012 Speed:60 Pos:x:1250.000000,y:1
Time: 46 ID:LTRK012 Speed:60 Pos:x:1266.666667,y:1
```

```
LogicalTime: 40
ID:FTRK002 Speed:60 Position:x=1173.666667,y=1
LogicalTime: 41
ID:FTRK002 Speed:60 Position:x=1190.333333,y=1
LogicalTime: 42
ID:FTRK002 Speed:60 Position:x=1207.000000,y=1
LogicalTime: 43
ID:FTRK002 Speed:60 Position:x=1223.666667,y=1
LogicalTime: 44
ID:FTRK002 Speed:60 Position:x=1240.333333,y=1
LogicalTime: 45
ID:FTRK002 Speed:60 Position:x=1257.000000,y=1
```

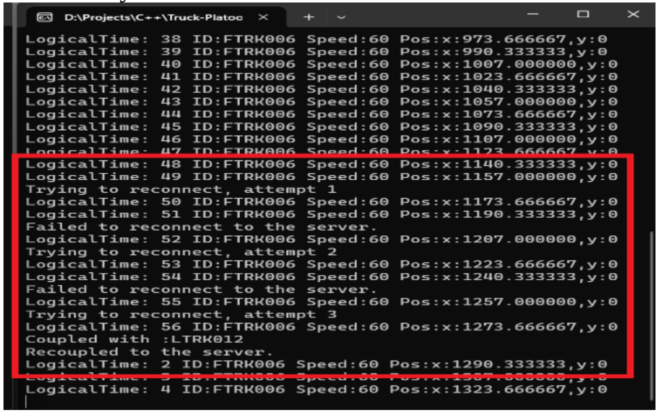
```
Select mode:
1. Leader
2. Follower
Enter choice: 2
Enter ID: FTRK002
LogicalTime: 1
Coupled with leader:LTRK012
ID: Speed:60 Position:x=-92559631349317830736831783200707727132248687965119994463780864.000000,y=1
LogicalTime: 1
LogicalTime: 2
```

Fig. 5. Duplicate Request Handling

In this scenario, FTRK002, an existing follower in the platoon, attempts to rejoin the platoon with the same ID. Upon receiving the joining request, the leader identifies this request as a duplicate. Subsequently, the leader denies acceptance of the request and outputs the message "Duplicate request: Follower FTRK002 already exists." This acknowledgement is crucial in maintaining the uniqueness of follower IDs within the platoon and preventing potential inconsistencies.

F. *Reconnection After Communication Failure (Within 15 Seconds)*

In this scenario, as depicted in Fig. 6, a follower experiences a temporary communication failure with the leader. The system detects the communication disruption and initiates a reconnection attempt. Since the reconnection occurs within the designated 15-second timeframe, the follower successfully re-establishes communication with the leader.



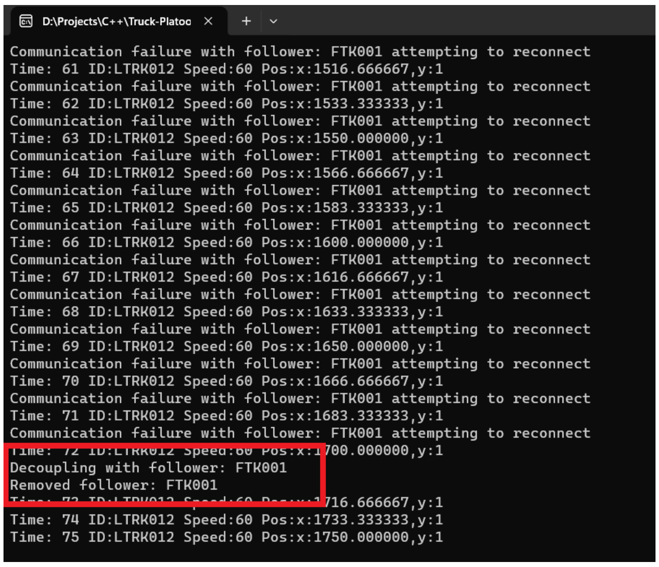
```
LogicalTime: 38 ID:FTRK006 Speed:60 Pos:x:973.666667,y:0
LogicalTime: 39 ID:FTRK006 Speed:60 Pos:x:990.333333,y:0
LogicalTime: 40 ID:FTRK006 Speed:60 Pos:x:1007.000000,y:0
LogicalTime: 41 ID:FTRK006 Speed:60 Pos:x:1023.666667,y:0
LogicalTime: 42 ID:FTRK006 Speed:60 Pos:x:1040.333333,y:0
LogicalTime: 43 ID:FTRK006 Speed:60 Pos:x:1057.000000,y:0
LogicalTime: 44 ID:FTRK006 Speed:60 Pos:x:1073.666667,y:0
LogicalTime: 45 ID:FTRK006 Speed:60 Pos:x:1090.333333,y:0
LogicalTime: 46 ID:FTRK006 Speed:60 Pos:x:1107.000000,y:0
LogicalTime: 47 ID:FTRK006 Speed:60 Pos:x:1123.666667,y:0
LogicalTime: 48 ID:FTRK006 Speed:60 Pos:x:1140.333333,y:0
LogicalTime: 49 ID:FTRK006 Speed:60 Pos:x:1157.000000,y:0
Trying to reconnect, attempt 1
LogicalTime: 50 ID:FTRK006 Speed:60 Pos:x:1173.666667,y:0
LogicalTime: 51 ID:FTRK006 Speed:60 Pos:x:1190.333333,y:0
Failed to reconnect to the server.
LogicalTime: 52 ID:FTRK006 Speed:60 Pos:x:1207.000000,y:0
Trying to reconnect, attempt 2
LogicalTime: 53 ID:FTRK006 Speed:60 Pos:x:1223.666667,y:0
LogicalTime: 54 ID:FTRK006 Speed:60 Pos:x:1240.333333,y:0
Failed to reconnect to the server.
LogicalTime: 55 ID:FTRK006 Speed:60 Pos:x:1257.000000,y:0
Trying to reconnect, attempt 3
LogicalTime: 56 ID:FTRK006 Speed:60 Pos:x:1273.666667,y:0
Coupled with :LTRK012
Recoupled to the server.
LogicalTime: 2 ID:FTRK006 Speed:60 Pos:x:1290.333333,y:0
LogicalTime: 3 ID:FTRK006 Speed:60 Pos:x:1307.000000,y:0
LogicalTime: 4 ID:FTRK006 Speed:60 Pos:x:1323.666667,y:0
```

Fig. 6. Reconnection After Communication Failure (Within 15 Seconds)

This test condition demonstrates the system's capability to handle transient communication issues and ensures the continuity of communication within the platoon with minimal disruption.

G. *Reconnection After Prolonged Communication Failure (Beyond 15 Seconds)*

In this scenario, as depicted in Fig. 7, a follower experiences a temporary communication failure with the leader. The system detects the communication disruption and initiates a reconnection attempt. However, due to the persistence of the communication issue beyond the designated 15-second timeframe, the leader successfully decouples and removes the follower from the platoon.



```
Communication failure with follower: FTK001 attempting to reconnect
Time: 61 ID:LTRK012 Speed:60 Pos:x:1516.666667,y:1
Communication failure with follower: FTK001 attempting to reconnect
Time: 62 ID:LTRK012 Speed:60 Pos:x:1533.333333,y:1
Communication failure with follower: FTK001 attempting to reconnect
Time: 63 ID:LTRK012 Speed:60 Pos:x:1550.000000,y:1
Communication failure with follower: FTK001 attempting to reconnect
Time: 64 ID:LTRK012 Speed:60 Pos:x:1566.666667,y:1
Communication failure with follower: FTK001 attempting to reconnect
Time: 65 ID:LTRK012 Speed:60 Pos:x:1583.333333,y:1
Communication failure with follower: FTK001 attempting to reconnect
Time: 66 ID:LTRK012 Speed:60 Pos:x:1600.000000,y:1
Communication failure with follower: FTK001 attempting to reconnect
Time: 67 ID:LTRK012 Speed:60 Pos:x:1616.666667,y:1
Communication failure with follower: FTK001 attempting to reconnect
Time: 68 ID:LTRK012 Speed:60 Pos:x:1633.333333,y:1
Communication failure with follower: FTK001 attempting to reconnect
Time: 69 ID:LTRK012 Speed:60 Pos:x:1650.000000,y:1
Communication failure with follower: FTK001 attempting to reconnect
Time: 70 ID:LTRK012 Speed:60 Pos:x:1666.666667,y:1
Communication failure with follower: FTK001 attempting to reconnect
Time: 71 ID:LTRK012 Speed:60 Pos:x:1683.333333,y:1
Communication failure with follower: FTK001 attempting to reconnect
Time: 72 ID:LTRK012 Speed:60 Pos:x:1700.000000,y:1
Decoupling with follower: FTK001
Removed follower: FTK001
Time: 73 ID:LTRK012 Speed:60 Pos:x:1716.666667,y:1
Time: 74 ID:LTRK012 Speed:60 Pos:x:1733.333333,y:1
Time: 75 ID:LTRK012 Speed:60 Pos:x:1750.000000,y:1
```

Fig. 7. Reconnection After Prolonged Communication Failure (Beyond 15 Seconds)

The follower is removed from the platoon, ensures the prevention of potential disruptions caused by an extended inability to reestablish communication. This test condition highlights the system's adaptive nature, prioritizing stability and preventing prolonged inconsistencies in the platoon.

H. *Future Scope*

While the implemented truck platooning system successfully integrates follower trucks, future enhancements could focus on accommodating follower trucks at the rear slots for improved platoon dynamics. Additionally, addressing lane-changing capabilities is crucial for comprehensive system functionality. Currently, the system considers both x and y planes for position but does not incorporate alterations in the y plane, limiting the ability to change lanes. Introducing lane-changing mechanisms will add a crucial dimension to the system, enhancing its versatility and adaptability to diverse road scenarios.

IV. Time Synchronization

Ensuring precise time synchronization is a fundamental requirement for the seamless operation of truck platooning systems, where a lead vehicle communicates wirelessly with a convoy of driverless trucks. The incorporation of advanced safety technologies, such as vehicle detection, anti-collision mechanisms and lateral control, underscores the critical need for temporal coherence.

Physical clocks exhibit variations in resolution, accuracy, and drift, necessitating their role in establishing temporal order within systems. These clocks play a crucial role in calculating elapsed and current time, contributing to the achievement of distributed temporal coherence, particularly in embedded domains. On the other hand, logical clocks serve a different purpose by not directly offering time or temporal relations. Instead, they focus on determining causally related events and ensuring transitivity within the system. While physical clocks cater to the temporal aspects of systems, logical clocks provide essential insights into the causal relationships between events, contributing to the overall coordination and synchronization in diverse domains.

In this section, the exploration centers around the implementation and comparison of three distinct logical clocks: Logical Matrix Clock, Vector Clock, and Lamport Logical Clock, in the context of truck platooning scenario. We delve into the characteristics, advantages and limitations of each clock, offering a comprehensive analysis that informs the decision-making process in selecting the most suitable synchronization method.

A. *Logical Matrix Clock*

The Logical Matrix Clock is an extension of Lamport Logical Clocks tailored to synchronize multiple processes. Each process maintains a logical matrix capturing the local logical clocks of all processes.

- Initialization: Each truck in the platoon, including the leading truck and driverless trucks, maintains its own logical clock vector to track local events. The logical matrix is initialized with all elements set to zero.

- Event Occurrence: When an event occurs in a truck, such as a change in speed, initiation of anti-collision measures, or coupling and de-coupling, the corresponding element in its logical clock vector is incremented.

- Matrix Update: During wireless communication between trucks, the sending truck includes its entire logical clock vector along with the message. Upon receiving the message, the receiving truck updates its own logical clock vector by taking the maximum value from each corresponding element in the received vector and its own vector. The logical matrix is updated by replacing the row corresponding to the sending truck with the received logical clock vector.

- Causality Determination: The logical matrix allows each truck to determine the causality relationships between its own events and events in other trucks. If the logical clock value at position (i, j) is greater than the logical clock value at position (k, j), it implies that event i in truck j causally precedes event k in the same truck j.

  Example:

  The leading truck (Process 1) detects an obstacle and initiates an anti-collision maneuver, incrementing its logical clock vector to [1, 0, 0]. It communicates this event to a driverless truck (Process 2) in the platoon. The driverless truck updates its logical clock vector to [1, 1, 0], and the matrix row (2, 1) is updated to [1, 0, 0].

While conceptually robust, the implementation of a Logical Matrix Clock in truck platooning system posed formidable challenges. The management and frequent updates of the matrix for each process resulted in considerable computational overhead. The complexity of maintaining such matrices in real-time and resource-constrained scenarios led to reevaluating its practicality for the implementation.

B. *Vector Clock*

Vector clocks are a logical clock synchronization algorithm used in distributed systems to capture the partial ordering of events. They are primarily employed to establish a causal relationship between events occurring in different processes within the system. The fundamental idea behind vector clocks is to assign a vector to each process, where each component of the vector corresponds to the logical time observed by a specific process.

- Initialization: Each process is assigned a vector clock initially set to [0, 0, 0, ..., 0]. Process 1 (leading truck): [0, 0, 0, ..., 0]. Process 2 (follower truck 1): [0, 0, 0, ..., 0]. Process 2 (follower truck 2): [0, 0, 0, ..., 0]. Process N (follower truck N): [0, 0, 0, ..., 0].

- Event Occurrence: When an event occurs in a process, the corresponding component in its vector clock is incremented. Let's say an event occurs in the leading truck (Process 1): [1, 0, 0, …, 0]. Simultaneously, an event occurs in one of the follower trucks, (Process 2): [0, 1, 0, …, 0], (Process 3: [0, 0, 1, 0, …, 0], (Process N): [0, 0, 0, …, 1].

- Sending and Receiving: When the leading truck sends a message to a follower truck, it includes its current vector clock. For example, Process 1 (leading truck) sends a message to Process 2 (follower truck) with the vector clock [1, 0, 0, …, 0]. Upon receiving the message, Process 2 updates its vector clock to [1, 1, 0, …, 0] and Process 3 updates its vector clock to [1, 1, 1, 0, …, 0].

- Causality Determination: Vector clocks help determine the causal relationship between events in different processes. If vector clock A is less than vector clock B (A < B), it implies that A causally precedes B. If A and B are not comparable (A || B), it indicates that A and B are concurrent events.

  Example:

  Leading truck (Process 1) has an event: [2, 0, …, 0]. Follower truck (Process 2) has a concurrent event: [0, 2, 0, …, 0]. Process 1 sends a message to Process 2 with its vector clock [2, 0, …, 0]. Upon receiving, Process 2 updates its vector clock to [2, 2, 0, …, 0].

- Advantages and Use Cases: Vector clocks assist in determining the causal relationship and ordering of events between the leading truck and follower trucks in the platoon. They provide a systematic way to track the logical progression of events, aiding in maintaining consistency and synchronization.

In summary, vector clocks in the truck platooning scenario allow for a nuanced understanding of the ordering of events between the leading and follower trucks, facilitating effective communication, coordination, and safety measures within the platoon. Though not ultimately implemented in the system as it did not involve the communication among driverless follower trucks.

C. *Lamport Logical Clock*

In contrast to the challenges presented by Logical Matrix and Vector Clocks, Lamport Logical Clocks emerged as the preferred choice for our truck platooning system. The Lamport Clock model offers a straightforward solution by assigning a unique timestamp to each event based on the logical time of the respective process.

- Clock Initialization: Each process in the distributed system maintains a logical clock, initialized to zero. The logical clock represents the ordering of events in the absence of a global clock.

- Event Occurrence: When an event occurs in a process, its logical clock is incremented by one. This increment signifies the occurrence of an event in that process. Events include internal computations, state changes, or any action that needs to be ordered.

- Send and Receive Events: When a process sends a message to another process, it includes its current logical clock value with the message. Upon receiving the message, the logical clock in the receiving process is updated to be the maximum of its current logical clock value and the logical clock value received in the message, plus one. This mechanism ensures that the logical clocks of processes are synchronized based on the occurrence of events.

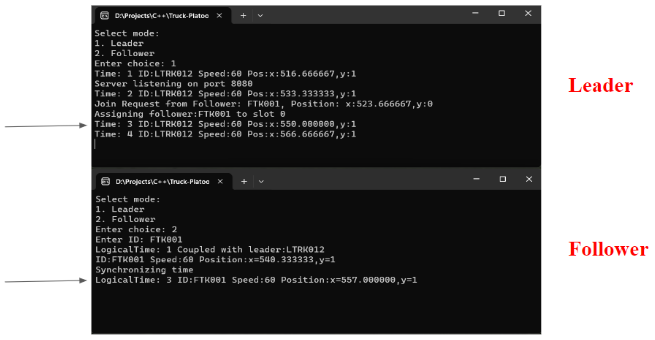
- Causality Determination**:** The key principle is that if event A precedes event B in terms of Lamport's logical clock values, then A causally precedes B. Causality is established by comparing the logical clock values associated with the events. If the logical clock of event A is less than the logical clock of event B, A causally precedes B.

Example:

Suppose process P1 has an initial logical clock value of 5. If an event occurs in P1, the logical clock is incremented to 6. If P1 sends a message to process P2, the message includes the logical clock value (6) at that moment. If P2 receives the message, its logical clock is updated to max(current value, received value + 1), resulting in an updated logical clock of 7 in P2.

- Use Cases: Lamport Logical Clocks are commonly used in distributed systems for debugging, tracing, and understanding the flow of events. They form the basis for more advanced logical clock models and distributed algorithms.
- Concurrency Control: Lamport Logical Clocks are integral to concurrency control mechanisms in distributed databases and file systems. They help resolve conflicts and maintain consistency in distributed data storage.
- Limitations: Lamport Logical Clocks do not guarantee a total order of events, meaning events that are not causally related may have the same logical clock value. The clock values themselves do not correspond to real-time units, as they only serve the purpose of establishing a relative ordering.
- Benefits: Lamport Logical Clocks provide a partial ordering of events in a distributed system, allowing processes to understand the causal relationships between events. It facilitates reasoning about the relative order of events and ensures a consistent view of causality across processes. The simplicity of implementation makes it suitable for various distributed system scenarios.

In the initial implementation of Lamport Logical Clock for the truck platooning scenario, there was a misconception in the synchronization logic. Originally, both the logical clocks of the leader truck and follower trucks were incremented simultaneously. When a coupling request is initiated by a follower truck, the synchronization occurs by comparing and aligning the clocks of the leader and follower trucks, choosing the maximum value. The result of initial implementation leads to time synchronization among leader and follower trucks as shown in Fig. 8

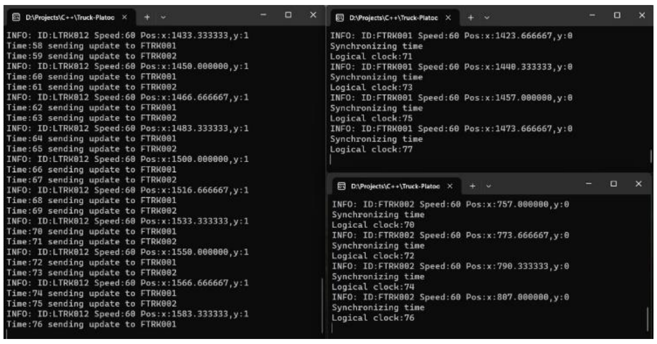


```
Select mode:
1. Leader
2. Follower
Enter choice: 1
Time: 1 ID:LTRK012 Speed:60 Pos:x:516.666667,y:1
Server listening on port 8080
Time: 2 ID:LTRK012 Speed:60 Pos:x:533.333333,y:1
Join Request from Follower: FTK001, Position: x:523.666667,y:0
Assigning follower:FTK001 to slot 0
Time: 3 ID:LTRK012 Speed:60 Pos:x:550.000000,y:1
Time: 4 ID:LTRK012 Speed:60 Pos:x:566.666667,y:1
```

```
Select mode:
1. Leader
2. Follower
Enter choice: 2
Enter ID: FTK001
LogicalTime: 1 Coupled with leader:LTRK012
ID:FTK001 Speed:60 Position:x=540.333333,y=1
Synchronizing time
LogicalTime: 3 ID:FTK001 Speed:60 Position:x=557.000000,y=1
```

Fig. 8. Timing synchronization (Initial Implementation)

However, this approach contradicted the fundamental principles of Lamport Logical Clock, as it failed to capture the causality relationships between events in a distributed system accurately. To address this, we refined our logic to adhere to Lamport's definition. In the revised approach, the leader truck consistently sends clock updates to the follower trucks at each time unit. This ensures that the logical clocks of the follower trucks are synchronized with the leader's clock. When a coupling request is initiated by a follower truck, the synchronization process involves comparing and aligning the clocks of the leader and follower trucks, with the synchronization point determined as the maximum value between the two clocks. This refined logic aligns with Lamport's logical clock principles by preserving the causal relationships between events in the platooning system, providing a more accurate and consistent time synchronization approach. The result of this implementation in the simulation environment is shown in Fig. 9



```
INFO: ID:LTRK012 Speed:60 Pos:x:1433.333333,y:1
Time:58 sending update to FTRK001
Time:59 sending update to FTRK002
INFO: ID:LTRK012 Speed:60 Pos:x:1450.000000,y:1
Time:60 sending update to FTRK001
Time:61 sending update to FTRK002
INFO: ID:LTRK012 Speed:60 Pos:x:1466.666667,y:1
Time:62 sending update to FTRK001
Time:63 sending update to FTRK002
INFO: ID:LTRK012 Speed:60 Pos:x:1483.333333,y:1
Time:64 sending update to FTRK001
Time:65 sending update to FTRK002
INFO: ID:LTRK012 Speed:60 Pos:x:1500.000000,y:1
Time:66 sending update to FTRK001
Time:67 sending update to FTRK002
INFO: ID:LTRK012 Speed:60 Pos:x:1516.666667,y:1
Time:68 sending update to FTRK001
Time:69 sending update to FTRK002
INFO: ID:LTRK012 Speed:60 Pos:x:1533.333333,y:1
Time:70 sending update to FTRK001
Time:71 sending update to FTRK002
INFO: ID:LTRK012 Speed:60 Pos:x:1550.000000,y:1
Time:72 sending update to FTRK001
Time:73 sending update to FTRK002
INFO: ID:LTRK012 Speed:60 Pos:x:1566.666667,y:1
Time:74 sending update to FTRK001
Time:75 sending update to FTRK002
INFO: ID:LTRK012 Speed:60 Pos:x:1583.333333,y:1
Time:76 sending update to FTRK001
```

```
INFO: ID:FTRK001 Speed:60 Pos:x:1423.666667,y:0
Synchronizing time
Logical clock:71
INFO: ID:FTRK001 Speed:60 Pos:x:1440.333333,y:0
Synchronizing time
Logical clock:73
INFO: ID:FTRK001 Speed:60 Pos:x:1457.000000,y:0
Synchronizing time
Logical clock:75
INFO: ID:FTRK001 Speed:60 Pos:x:1473.666667,y:0
Synchronizing time
Logical clock:77
```

```
INFO: ID:FTRK002 Speed:60 Pos:x:757.000000,y:0
Synchronizing time
Logical clock:70
INFO: ID:FTRK002 Speed:60 Pos:x:773.666667,y:0
Synchronizing time
Logical clock:72
INFO: ID:FTRK002 Speed:60 Pos:x:790.333333,y:0
Synchronizing time
Logical clock:74
INFO: ID:FTRK002 Speed:60 Pos:x:807.000000,y:0
Synchronizing time
Logical clock:76
```

Fig. 9. Time synchronization using Lamport Logical Clock

In the context of truck platooning scenario, while Lamport Logical Clocks have demonstrated efficacy, there is room for enhancement. Logical Matrix Clocks, despite their conceptual robustness, present challenges in real-time implementation, primarily due to the computational overhead associated with managing and updating matrices for each process. Future research could focus on optimizing algorithms or devising strategies to mitigate the computational burden, making Logical Matrix Clocks more feasible for real-world applications. Similarly, Vector Clocks, although not implemented in our system, could be subject to future investigation. Advances in algorithms addressing the complexity of vector size management and concurrent event updates could make Vector Clocks more suitable for scenarios with limited computational capabilities. Lamport Logical Clocks, although lightweight, may encounter challenges in scenarios where the system undergoes rapid changes or experiences bursts of communication events. Exploring hybrid approaches that leverage the strengths of multiple logical clock methods may lead to improved synchronization mechanisms tailored to the unique requirements of the truck platooning system. Additionally, the integration of predictive algorithms or machine learning models could be considered to anticipate the timing of critical events, allowing for proactive adjustments in the logical clocks. Moreover, advancements in hardware technologies, such as the adoption of more powerful processors or specialized timekeeping units, could further refine the time synchronization capabilities, especially in resource-constrained embedded systems like those employed in truck platooning. Continuous research and development efforts in the field of distributed systems and time synchronization algorithms will play a pivotal role in refining the temporal coordination of events within the truck platooning system, ensuring robust and reliable operation in dynamic and real-world scenarios.

V. GPU Implementation

A. Maintaining Distance

A critical aspect of truck platooning is the ability to maintain a consistent and safe distance between trucks traveling in a convoy. This part focuses on a key component of a larger truck platooning simulation project - the implementation of a GPU-accelerated Proportional-Integral-Derivative (PID) controller dedicated to maintaining an optimal distance between trucks. Given the computational demands of real-time distance calculation and adjustment in a dynamic environment, we have utilized GPU acceleration to enhance the controller's performance. The aim is to ensure that each truck in the platoon maintains a predefined gap from its predecessor, factoring in the constant speed of the leading truck and the predetermined length of each vehicle.

1) PID Controller Design

The PID controller implemented in our truck platooning simulation is a pivotal component designed to ensure precise distance maintenance between trucks. By leveraging the power of GPU acceleration, the PID controller can efficiently process the necessary calculations to maintain a constant gap between the trucks. The Proportional-Integral-Derivative (PID) controller is a widely-used control loop feedback mechanism. In our simulation, the PID controller's role is to calculate the required adjustment in speed to maintain a set distance between each truck. This adjustment is based on the difference, or error, between the truck's current position and its expected position in the platoon. The PID controller operates based on three components: Proportional, Integral, and Derivative. Each component plays a crucial role:

- Proportional: This component produces an output value that is proportional to the current error value. It determines the reaction to the current distance error.
- Integral: This component is concerned with the accumulation of past errors. It seeks to eliminate residual steady-state error by adjusting the truck speed based on the historical cumulative error.
- Derivative: This component predicts future error based on the rate of change. It contributes to the system's stability by dampening the proportional response.

2) PID Constraints

These constants were carefully chosen to optimize the control of the trucks' distances in the platoon. Each component of the PID controller plays a distinct role in achieving this:

- Proportional Gain (K_p = 0.3): This constant determines the response based on the current distance error. A K_p value of 0.3 ensures a aggressive response, meaning the acceleration of the follower truck increases significantly when it is far from its expected position. This helps in quickly reducing large gaps.
- Integral Gain (K_i = 0.02): The integral gain is responsible for the accumulation and correction of past errors. A lower value of 0.02 for K_i indicates that the controller will gradually adjust for accumulated errors over time, preventing drastic speed changes due to small, persistent errors.
- Derivative Gain (K_d = 0.2): This gain predicts future error based on the current rate of change. A K_d of 0.2 provides a balance, dampening the system's response to rapid changes in error, which helps in stabilizing the platoon, especially when adjusting speeds.

The combination of these constants is designed to ensure that the PID controller efficiently maintains the desired distance between trucks, with a focus on quicker acceleration when the gap increases, while also ensuring stability and smooth operation of the platoon.

3) Control system architecture

During the initial design phase of our truck platooning simulation, a pivotal decision was made regarding the placement of the PID controllers. The original concept involved centralizing the control mechanism within the leader truck, which would receive the current positions of the followers and then issue acceleration or deceleration commands. This approach, however, presented significant risks and limitations, primarily concerning communication latency and reliability. The latency inherent in wireless communication could introduce delays in the control signals reaching the follower trucks, which would be particularly problematic at high speeds or in situations requiring quick adjustments. Additionally, in the event of communication failures, the follower trucks would be rendered without control instructions, potentially leading to safety hazards.

To mitigate these risks, we decided to decentralize the PID control by embedding a controller within each follower truck. This ensures that each vehicle independently maintains its position in the platoon, relying on local data to adjust its speed. Such a design enhances the robustness of the system by eliminating dependency on continuous communication for control. It also empowers each truck with the ability to respond immediately to dynamic changes, ensuring a safer and more reliable platoon formation. This architecture decision highlights our proactive approach to system reliability and safety, ensuring that each follower truck is self-sufficient in maintaining the desired gap with minimal reliance on the leader truck's communications.

4) Need for GPU acceleration

The implementation of a GPU-accelerated PID controller is a strategic choice in our truck platooning simulation, primarily driven by the need for high-speed, real-time processing capabilities. Given the substantial computational demands of continuously calculating and adjusting the speeds of multiple trucks to maintain precise distances, the parallel processing prowess of GPUs offers a significant advantage. Unlike traditional CPUs that handle complex tasks sequentially, GPUs excel in conducting numerous simple tasks simultaneously, thereby ensuring rapid and efficient computations. This acceleration is crucial for real-time responsiveness in a dynamic system where delays can lead to inefficiencies and safety risks.

5) Implementation using CUDA

The CUDA-based implementation of the PID controller within our truck platooning simulation is a testament to the system's need for real-time processing. Each follower truck within the simulation is endowed with a local PID controller, ensuring robust and immediate response to the dynamic environment. The CUDA kernel *pidKernel* is invoked where it performs the PID computation using the constants. These constants were determined through iterative tuning to ensure that the acceleration and deceleration are appropriately responsive to the distance errors, hence maintaining a precise gap between the trucks. The kernel calculates the proportional, integral, and derivative terms based on the current error, which is the difference between the truck's current and desired positions.

The CUDA architecture allows for these calculations to be processed in parallel across multiple trucks. Each truck is assigned a unique thread within the GPU, allowing simultaneous and independent PID computations. This design is crucial for scaling the simulation to accommodate a larger platoon without degradation in performance. After the PID computations, the kernel updates each truck's speed accordingly. Memory management is handled through CUDA's *cudaMalloc* and *cudaMemcpy* functions, which manage the allocation and transfer of data between the host and device memory spaces. Once the speed adjustments are computed, the updated speeds are transferred back to the host, allowing for the simulation to proceed with the new speed.

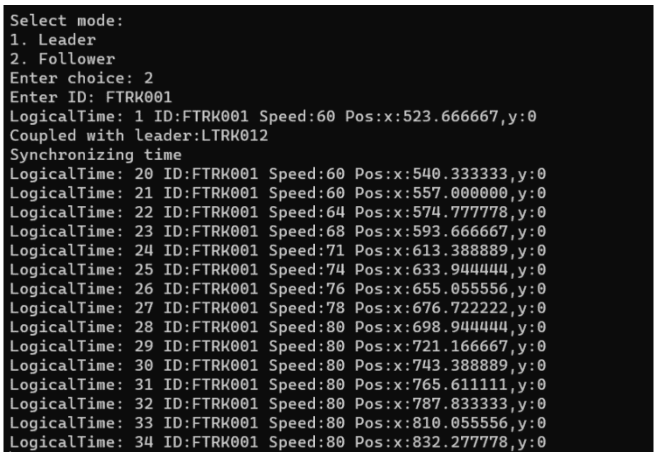
6) Simulation Assumptions and Constraints

The algorithm for maintaining the desired position of each truck in the platoon is designed to calculate the exact location where a follower truck should be at any given time. The desired position is determined by the leader truck's current position and a predetermined gap that each truck must maintain. The gap includes the length of a truck 5 meters plus an additional 2-meter distance for safety. For instance, the desired position for the first follower Slot 0 is the leader's position minus 7 meters, the second follower Slot 1 is the leader's position minus 14 meters, and so on. The trucks onboard systems continuously calculate their current position based on their speed and time elapsed, comparing it to the desired position to determine the positional error. If the follower truck is behind its expected position, the PID controller outputs a positive control signal, indicating acceleration. If the follower truck is ahead, the output is negative, signalling deceleration. The truck's onboard system then translates this control signal into throttle adjustments or brake application to change the speed accordingly, always within the constraints of the minimum and maximum speed limits. The trucks are programmed to operate within a speed range of 40 km/h to 80 km/h, with the leader truck moving at a constant speed of 60 km/h. The minimum speed limit ensures the platoon moves efficiently without blocking traffic, while the maximum limit prevents trucks from speeding, which could be unsafe and against traffic regulations.

7) Simulation Results

The simulation log in Fig. 10 captures the dynamic response of the truck FTRK001, which holds slot 0 in the platoon, immediately following the leader. Initially, FTRK001 maintains a cruising speed of 60 km/h. This initial behaviour indicates that the truck is at the correct distance behind the leader, as the PID controller is not making any adjustments to the speed, mirroring the leader's velocity to maintain the set gap. There is a noticeable incremental increase in the speed of FTRK001. This gradual acceleration suggests that the truck is responding to a calculated distance error, likely being slightly farther from the leader than the desired gap.

To correct this discrepancy, the PID controller engages, increasing the throttle input to accelerate the truck and reduce the gap. The log entries show the truck's speed increasing from the initial 60 km/h to a consistent 80 km/h, which is the upper limit of the speed regulation constraints. The truck's position on the x-axis, which represents its forward movement, increases accordingly with the increase in speed. Upon reaching the maximum speed of 80 km/h, FTRK001 sustains this speed, as upper speed is set in the simulation as 80km/h. This plateau at the maximum speed demonstrates the controller's compliance with the speed constraints while still striving to minimize the gap error. The truck's performance, as indicated by the log, illustrates the PID controller's successful execution of its task within the given operational parameters - maintaining the set distance behind the leader truck without exceeding the defined speed limits.



```
Select mode:
1. Leader
2. Follower
Enter choice: 2
Enter ID: FTRK001
LogicalTime: 1 ID:FTRK001 Speed:60 Pos:x:523.666667,y:0
Coupled with leader:LTRK012
Synchronizing time
LogicalTime: 20 ID:FTRK001 Speed:60 Pos:x:540.333333,y:0
LogicalTime: 21 ID:FTRK001 Speed:60 Pos:x:557.000000,y:0
LogicalTime: 22 ID:FTRK001 Speed:64 Pos:x:574.777778,y:0
LogicalTime: 23 ID:FTRK001 Speed:68 Pos:x:593.666667,y:0
LogicalTime: 24 ID:FTRK001 Speed:71 Pos:x:613.388889,y:0
LogicalTime: 25 ID:FTRK001 Speed:74 Pos:x:633.944444,y:0
LogicalTime: 26 ID:FTRK001 Speed:76 Pos:x:655.055556,y:0
LogicalTime: 27 ID:FTRK001 Speed:78 Pos:x:676.722222,y:0
LogicalTime: 28 ID:FTRK001 Speed:80 Pos:x:698.944444,y:0
LogicalTime: 29 ID:FTRK001 Speed:80 Pos:x:721.166667,y:0
LogicalTime: 30 ID:FTRK001 Speed:80 Pos:x:743.388889,y:0
LogicalTime: 31 ID:FTRK001 Speed:80 Pos:x:765.611111,y:0
LogicalTime: 32 ID:FTRK001 Speed:80 Pos:x:787.833333,y:0
LogicalTime: 33 ID:FTRK001 Speed:80 Pos:x:810.055556,y:0
LogicalTime: 34 ID:FTRK001 Speed:80 Pos:x:832.277778,y:0
```

Fig. 10. Log of Follower truck's speed adjustment by PID Controller

8) Challenges and Solution

Initially, the project structure included *Truck.h* and *Truck.cpp* files, which are typical for C++ projects. However, integrating CUDA required these to be transitioned to *Truck.cuh* and *Truck.cu* extensions, respectively, to be compatible with nvcc, the CUDA compiler. This change was essential for enabling the GPU acceleration of the PID controller calculations. The standard makefile approach for building the project proved to be complex when integrating CUDA code. To address the integration and compilation challenges, we transitioned to using CLion, an IDE that supports C++ and CUDA development. CLion facilitated a more seamless combination of C++ and CUDA code, streamlining the build process.

A practical challenge we faced was the lack of consistent access to NVIDIA GPUs across the development team. CUDA development requires NVIDIA hardware, and not all team members had the necessary equipment, which initially hindered our ability to develop and test the CUDA-accelerated components of our system uniformly. To circumvent this hardware limitation, we deferred the CUDA-specific development to the final stages of the project. This allowed team members without NVIDIA GPUs to contribute to other aspects of the project without being blocked by hardware constraints.

9) Future Scope

A current limitation in the system is the method of slot assignment for new trucks joining the platoon. Presently, a new truck is always assigned to the last slot. This approach, while straightforward, is not optimized for scenarios where a nearer slot may become available, potentially leading to inefficient platooning dynamics. A proposed enhancement to the system is the development of an intelligent slot assignment algorithm. This would allow for dynamic allocation of slots, where a new truck could be assigned to the nearest available slot rather than defaulting to the last position. Implementing this would optimize the platoon formation, reduce unnecessary travel distances for joining trucks, and improve the overall efficiency of the platooning system.

VI. Summary

The project delves into the realm of truck platooning system in the transportation landscape. The proposed system hinges on effective communication between leader and follower trucks, ensuring the continuous exchange of crucial time and position information. The coordination process involves slot assignment and request acceptance, with robust protocols to handle duplicates. The system employs resilient communication strategies, attempting reconnection in case of failure and initiating de-coupling if recovery remains elusive. The integration of time synchronization, CUDA, and PID in GPU calculations facilitates optimal distance maintenance between trucks. The implementation is realized using CLion on Windows, whose results match the expected behavior. The study highlights the profound impact of truck platooning on fuel efficiency, operational efficacy, road safety, and environmental sustainability.

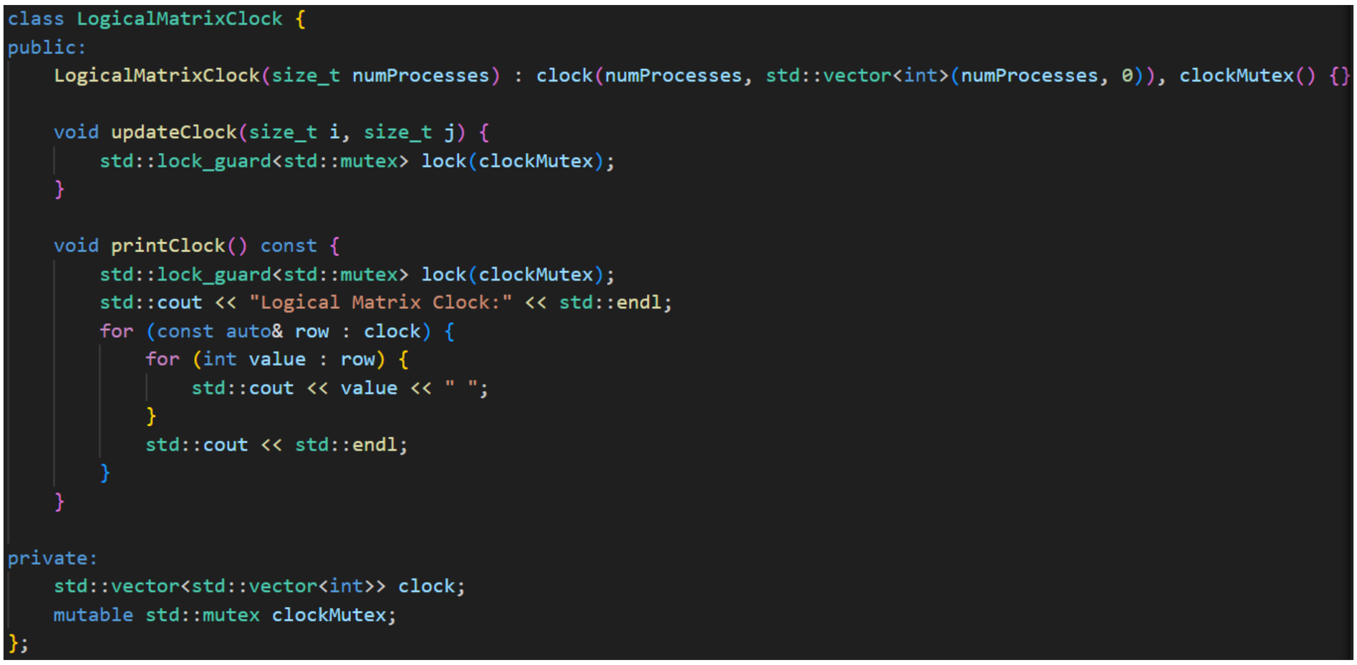
VII. References

[1] https://github.com/doctest/doctest

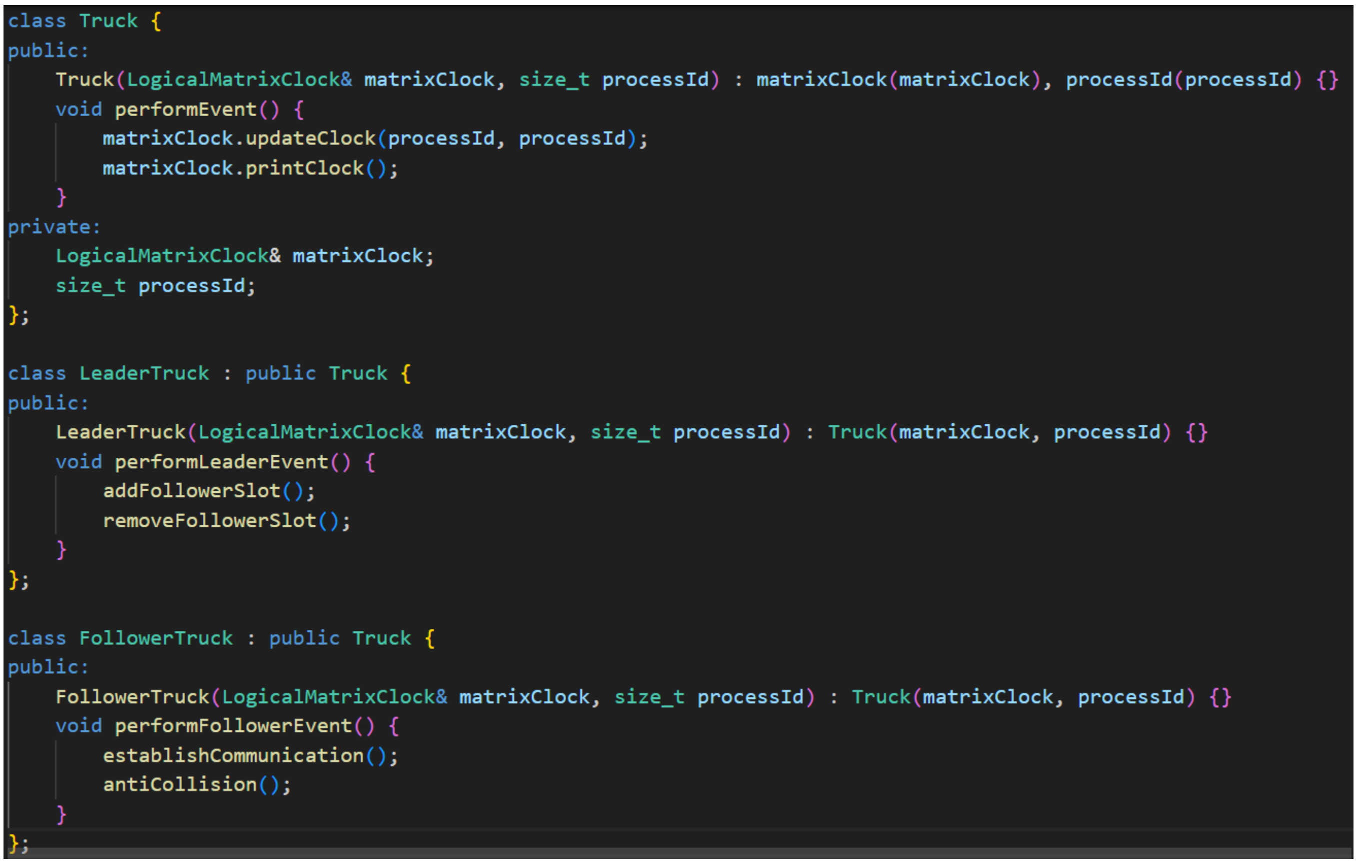
[2] https://developer.nvidia.com/cuda-toolkit

Appendix A

Time Synchronization using Logical Matrix Clock Code

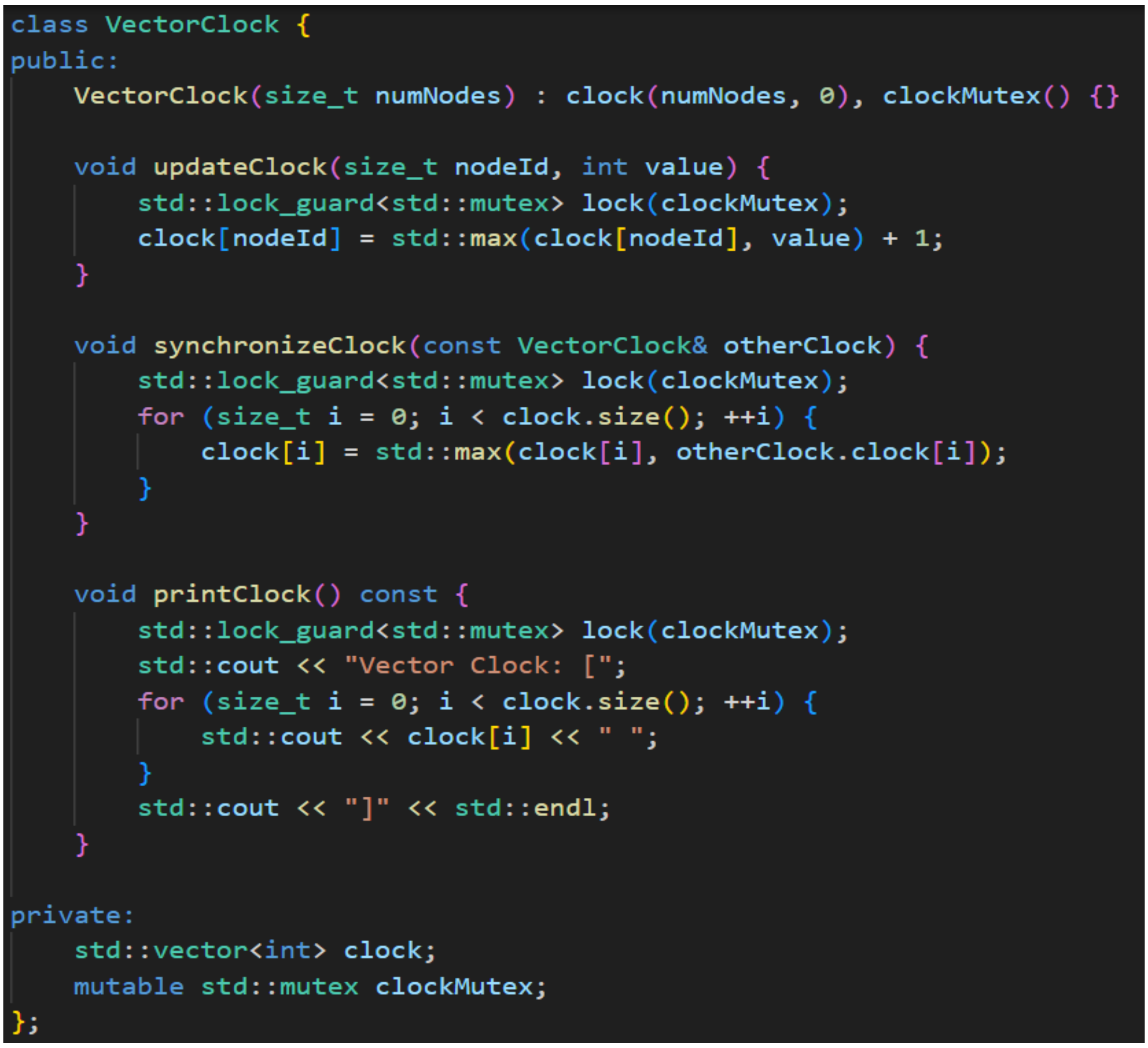


```cpp
class LogicalMatrixClock {
public:
    LogicalMatrixClock(size_t numProcesses) : clock(numProcesses, std::vector<int>(numProcesses, 0)), clockMutex() {}

    void updateClock(size_t i, size_t j) {
        std::lock_guard<std::mutex> lock(clockMutex);
    }

    void printClock() const {
        std::lock_guard<std::mutex> lock(clockMutex);
        std::cout << "Logical Matrix Clock:" << std::endl;
        for (const auto& row : clock) {
            for (int value : row) {
                std::cout << value << " ";
            }
            std::cout << std::endl;
        }
    }

private:
    std::vector<std::vector<int>> clock;
    mutable std::mutex clockMutex;
};
```

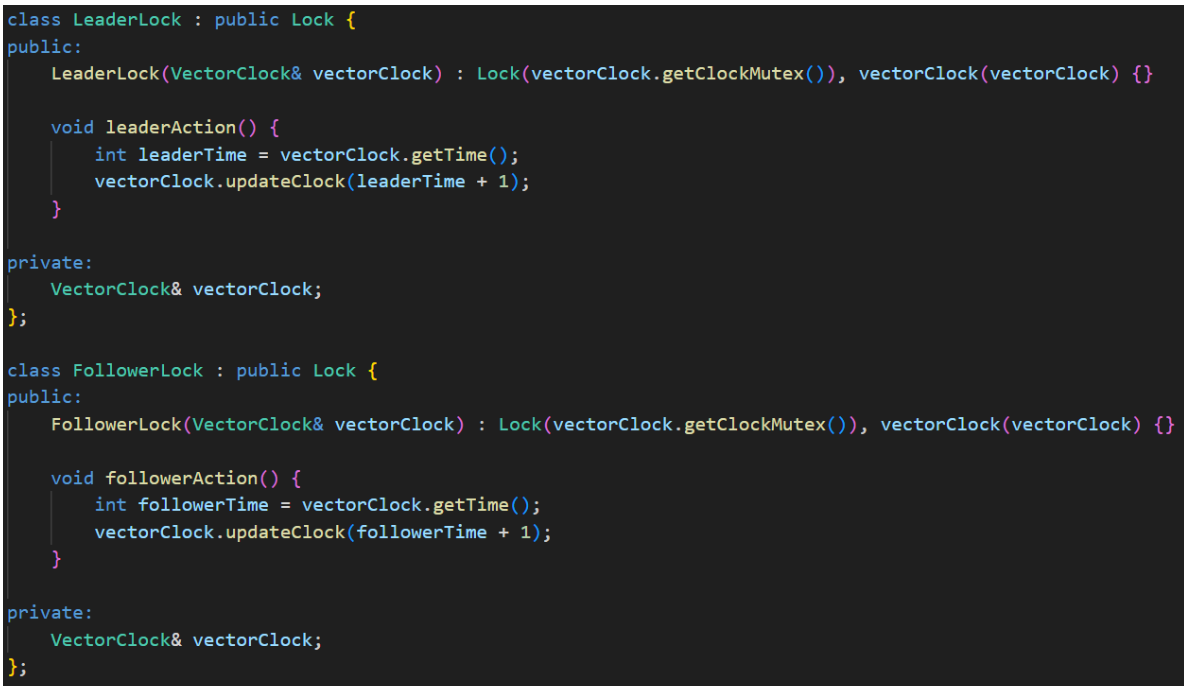


```cpp
class Truck {
public:
    Truck(LogicalMatrixClock& matrixClock, size_t processId) : matrixClock(matrixClock), processId(processId) {}
    void performEvent() {
        matrixClock.updateClock(processId, processId);
        matrixClock.printClock();
    }
private:
    LogicalMatrixClock& matrixClock;
    size_t processId;
};

class LeaderTruck : public Truck {
public:
    LeaderTruck(LogicalMatrixClock& matrixClock, size_t processId) : Truck(matrixClock, processId) {}
    void performLeaderEvent() {
        addFollowerSlot();
        removeFollowerSlot();
    }
};

class FollowerTruck : public Truck {
public:
    FollowerTruck(LogicalMatrixClock& matrixClock, size_t processId) : Truck(matrixClock, processId) {}
    void performFollowerEvent() {
        establishCommunication();
        antiCollision();
    }
};
```

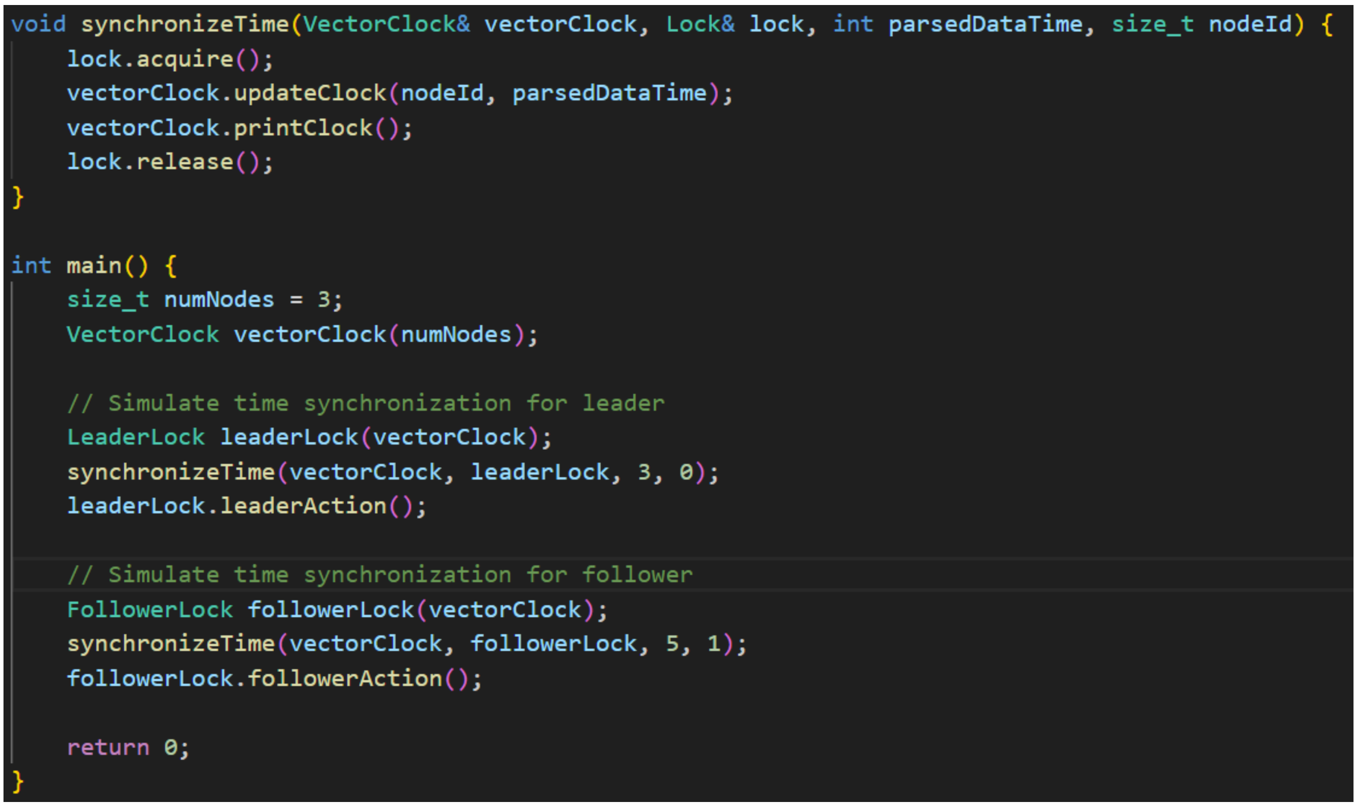
Time Synchronization using Vector Clock Code



```cpp
class VectorClock {
public:
    VectorClock(size_t numNodes) : clock(numNodes, 0), clockMutex() {}

    void updateClock(size_t nodeId, int value) {
        std::lock_guard<std::mutex> lock(clockMutex);
        clock[nodeId] = std::max(clock[nodeId], value) + 1;
    }

    void synchronizeClock(const VectorClock& otherClock) {
        std::lock_guard<std::mutex> lock(clockMutex);
        for (size_t i = 0; i < clock.size(); ++i) {
            clock[i] = std::max(clock[i], otherClock.clock[i]);
        }
    }

    void printClock() const {
        std::lock_guard<std::mutex> lock(clockMutex);
        std::cout << "Vector Clock: [";
        for (size_t i = 0; i < clock.size(); ++i) {
            std::cout << clock[i] << " ";
        }
        std::cout << "]" << std::endl;
    }

private:
    std::vector<int> clock;
    mutable std::mutex clockMutex;
};
```



```cpp
class LeaderLock : public Lock {
public:
    LeaderLock(VectorClock& vectorClock) : Lock(vectorClock.getClockMutex()), vectorClock(vectorClock) {}

    void leaderAction() {
        int leaderTime = vectorClock.getTime();
        vectorClock.updateClock(leaderTime + 1);
    }

private:
    VectorClock& vectorClock;
};

class FollowerLock : public Lock {
public:
    FollowerLock(VectorClock& vectorClock) : Lock(vectorClock.getClockMutex()), vectorClock(vectorClock) {}

    void followerAction() {
        int followerTime = vectorClock.getTime();
        vectorClock.updateClock(followerTime + 1);
    }

private:
    VectorClock& vectorClock;
};
```


```cpp
void synchronizeTime(VectorClock& vectorClock, Lock& lock, int parsedDataTime, size_t nodeId) {
    lock.acquire();
    vectorClock.updateClock(nodeId, parsedDataTime);
    vectorClock.printClock();
    lock.release();
}

int main() {
    size_t numNodes = 3;
    VectorClock vectorClock(numNodes);

    // Simulate time synchronization for leader
    LeaderLock leaderLock(vectorClock);
    synchronizeTime(vectorClock, leaderLock, 3, 0);
    leaderLock.leaderAction();

    // Simulate time synchronization for follower
    FollowerLock followerLock(vectorClock);
    synchronizeTime(vectorClock, followerLock, 5, 1);
    followerLock.followerAction();

    return 0;
}
```

APPENDIX B
Git Overview

A. Lines of code:

main.cpp - 468
Truck.cuh - 56
Truck.cu - 243
LeaderTruck.h - 22
LeaderTruck.cpp - 35
FollowerTruck.h - 24
FollowerTruck.cpp - 22
Client.h - 22
Client.cpp - 49
Server.h - 25
Server.cpp - 66

B. Number of submits per person:

Akash Cuntur Shrinivasmurthy - 4
Akhil Narayanaswamy - 1
Madhukar Devendrappa - 4
Swathi Chandrashekaraiah - 4
Vipin Krishna Vijayakumar - 3

C. Structure:

ea - .eapx files containing Requirements Diagram, State Machine Diagram and Activity Diagram
code - contains the source code

D. Contribution by each member:

1. Akash Cuntur Shrinivasmurthy:
    - PID implementation with CUDA
    - Software Coordinator
    - Paper: GPU implementation
2. Akhil Narayanaswamy:
    - Base classes for Truck, FollowerTruck, and LeaderTruck
    - Paper: Implementation
3. Madhukar Devendrappa:
    - Logical clock implementation
    - Requirements Diagram
    - Paper: Time Synchronization, Summary
4. Swathi Chandrashekaraiah:
    - Client code
    - State Machine Diagram
    - Paper: Abstract, Introduction & Motivation
5. Vipin Krishna Vijayakumar:
    - Server code
    - Activity Diagram
    - Paper: Activity Diagram

AFFIDAVIT

We, Akash Cuntur Shrinivasmurthy (7219642), Akhil Narayanaswamy (7219528), Madhukar Devendrappa (7219639), Swathi Chandrashekaraiah (7218877), Vipin Krishna Vijayakumar (7219307) herewith declare that we have composed the present paper and work ourself and without use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The paper and work in the same or similar form has not been submitted to any examination body and has not been published. This paper was not yet, even in part, used in another examination or as a course performance.

Dortmund, 31.01.2024

[illegible]

Akash Cuntur Shrinivasmurthy

[illegible]

Akhil Narayanaswamy

[illegible]

Madhukar Devendrappa

[illegible]

Swathi Chandrashekaraiah

[illegible]

Vipin Krishna Vijayakumar