Truck Platooning System

Vibhashree Vijayendra Hippargi
Dept. of Computer Science
FH Dortmund
Dortmund, Germany
vibhashree.hippargi001@stud.fh-dortmund.de

Supadma Kadabi
Dept. of Computer Science
FH Dortmund
Dortmund, Germany
supadma.kadabi002@stud.fh-dortmund.de

Sachin Kumar
Dept. of Computer Science
FH Dortmund
Dortmund, Germany
sachin.kumar003@stud.fh-dortmund.de

Hamida Aliyeva
Dept. of Computer Science
FH Dortmund
Dortmund, Germany
hamida.aliyeva004@stud.fh-dortmund.de

Abstract **- Main objective of this paper is to review and explain requirements for distributed and parallel systems, while introducing the development process for truck platooning technologies. Main objective of this paper is to review and explain requirements for distributed and parallel systems, while introducing the development process for truck platooning technologies. GPU programming is done by OpenCL and client-server implementation is done by a Java multi-threading system.**

Keywords - Platooning, Distributed, Parallel, GPU programming, Multi-core,* OpenCL*, Multi-threading

I. INTRODUCTION

author - Hamida Aliyeva

The first autonomous vehicle can be traced back to the 1980s, a project funded by the United States Defense Advanced Research Projects Agency called NavLab. It looked like a mail delivery truck. and bore little resemblance to Google's self-driving car (without a safety driver) that was first shown on Arizona roads in 2017. Since past Autonomous Vehicle (AV) development is constantly changing. Most of the systems that make up AV are typically built on artificial intelligence (AI), which is trained by identifying patterns in real-world settings and using those patterns to guide future actions that result in a specific outcome. In order to achieve a concrete level of autonomy, cars are equipped with on-board sensors, computing and communication devices, as electronics and software are increasingly embedded in the auto-mobile sector. Thanks to those technologies these intelligent vehicles are enabled to operate in a platoon model on the highway, where a closely spaced series of vehicles follow the same leader, maintain proximity, communicate with each other, and operate under fully automated longitudinal and lateral control. Platooning has various advantages. Platooning has a great number of advantages. For instance, improved fuel efficIn the past, vehicular platooning has been extensively studied from both a management and a network perspective, resulting in the creation of a platoon-based system. Requirements of designing such a system are written below:

1. To maintain platoon string stability through control algorithms and spacing policy.
2. To incorporate effective platoon management and real-time data dissemination.

Yet, it is rarely examined from the perspective of software architecture. Platooning is an example of a value-added service that calls for architectural changes from an existing automobile to a future smart car, a change from a feature phone to a fixed-function embedded system, and a change to a single software framework that can connect different services to the electronic units of the vehicles inside a platoon, especially for heavy-duty trucks, results from lower air drag. In addition, less vehicle gaps and speed disturbances in the traffic contribute to increased road capacity and more comfortable travel. Also, newly created autonomous vehicles have the ability to sense the environment and take off on their own. Therefore, platooning software can be added to such vehicles as a value-added service.
one.

This article clearly explains the important terms in platoon and describes the main points of our client-server structure. Source code is also presented at the end.

II. CLIENT SERVER ARCHITECTURE

author -Vibhashree Hippargi

Distributed systems which communicate with each other over the network are usually addressed as Client-Server architecture model.In our truck platooning system, we employ two tier client server architecture since there is just one logical server which is being connected by 'n' number of clients. The below figure shows the pictorial representation of a server that is connected to 3 clients through a network. They communicate with each other by exchanging messages (msg in the diagram).

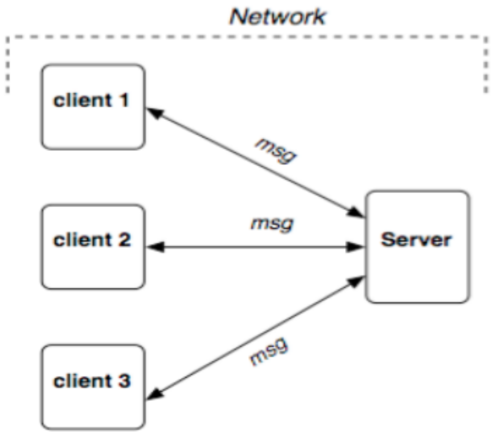

Fig 1:The structure of Client Server model

We call the server here as a "Leader" truck which is handled by a human driver and the platooned trucks as "Followers" which do not contain human drivers. They are automated. Slave trucks follow the leader truck and act according to its instructions.

There are two types of Clients. Thin client and Fat client.

Thin client is the one which mostly depends on the server and cannot work offline. Thick clients are the one which can do most of the computation by itself and does not depend much on the server.

In the truck platooning system, the client is a thin client because a significant amount of decision making and processing is being done by the leader truck (server). It is the one which controls important aspects such as speed, acceleration, deceleration, braking etc.of the platooning trucks. Ofcourse, the platooned trucks consist of sensors and computing power for monitoring and sending their current information to the server, but majority of the processing is done by the server.

Layers in client server system:

Fig 2: Thin Client Server model

Presentation Layer: This layer is contained in the platooned trucks (thin client). It deals with sending and receiving information to and from the server.

Application Processing Layer: This is the layer that contains the main logic or the application. It is present on the server. It processes the requests received from the slaves and makes decisions about what needs to be done which is sent back to the clients.The instructions could be such as adjusting the speed of the trucks in the platoon based on its current speed or distance between the adjacent trucks etc.

Data Handling Layer: This layer is used to manage connections between the server and the client. Here, TCP protocol is used for reliable connection between them.

Database Layer: This layer is present within the server.It stores and manages the data that is needed by the server which it uses to give instructions to the client.For example, it may store information about the data sent by the platoons about its location, weather, any object detection happening etc.

III. STATE DIAGRAM

The steps that the leader truck(master) and the platooned trucks follow are displayed below in the form of a state machine.

Here, the leader truck initiates the connection and waits for any of the platooned trucks to join the connection inorder to start the platooning.The leader truck is the one who sends the instructions on what the platooned trucks has to do based on the information received by the same.

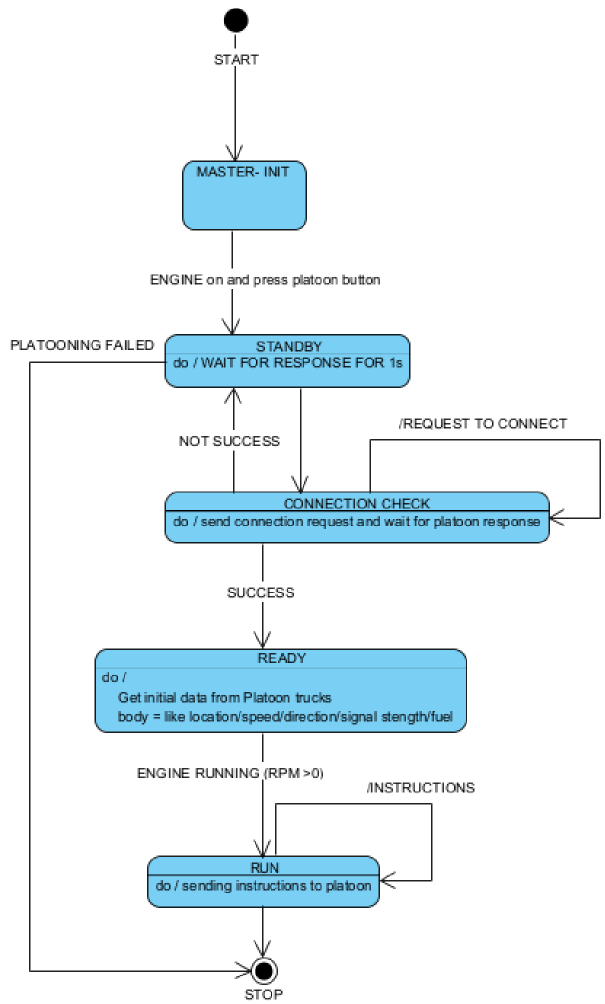

Fig 3: Leader truck

On the other hand, the responsibility of the platooning trucks is different from the leader. They have to sense and send their information regarding speed, location etc to the server truck based on which they get instructions and then the platooned trucks have to act accordingly. The below figure shows the responsibilities.

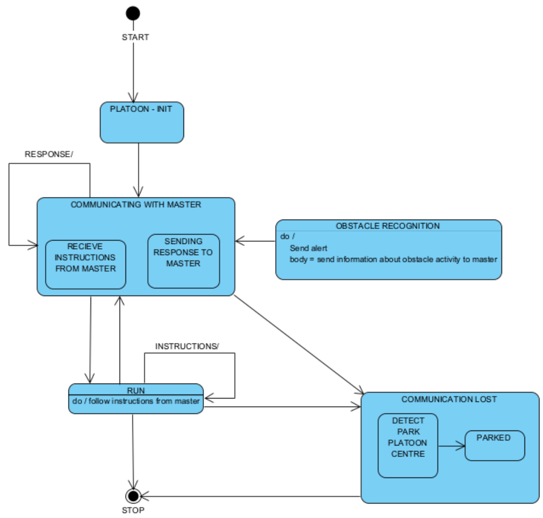

Fig 4: Platooned Trucks

The other distinct characteristics the platooned trucks possess are the obstacle recognition (also called as object detection)and the fault management.

When any of the clients detect an object in front of it such as another road user that is coming in between the platoon, it sends the object detected information to the server. Server instructs the platoon about the speed and distance it has to maintain to make sure to accommodate the vehicle and not to hit it.

When the signal strength of a particular platooned truck goes low and cuts off, it gets parked at the predefined location near to it.By this way, the entire system is not disturbed and only the affected truck withdraws from the platooning.

IV. SEQUENCE DIAGRAM

author - Supadma Kadabi

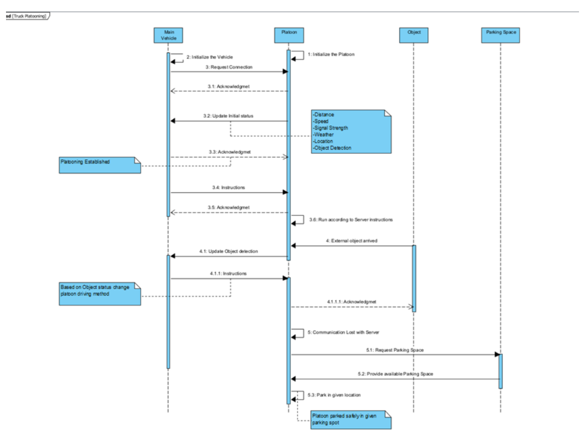

Fig 5: Sequence Diagram

The sequence diagram is used to model the communication between the blocks in time order. These diagrams are used to specify the interactions between the elements of the system. In the given sequence diagram we can briefly understand the working of the proposed Platooning system. The Main vehicle initiates the communication by starting to listen to the available platoons. Once the Main vehicle is ready Platoons start to respond to the Main vehicle and update it with its initial state. Then the Platoons continue to travel as per the instructions given by the Main Vehicle. The Platooning system works as per instructions and informs the Main vehicle whenever any object is encountered. Here, objects can be referred to any vehicles on the road. The Main vehicle will then decide on how the platoon has to tackle the situation and send appropriate instructions. There is a special circumstance when the communication between the main vehicle and platoon is lost, in which case the Platoon will refer to Parking assist and park the vehicle in a nearby parking spot.

V. TCP/IP MODEL

author - Vibhashree Hippargi

A protocol is a set of rules that is to be followed for proper communication between the devices on a network.It ensures that the communication is carried out in a consistent and predictable way. TCP/IP model is a conceptual framework that defines how the devices send and receive messages with each other over a network.

TCP/IP model consists of 4 layers each of which is responsible for a specific aspect of network communication. The layers are as shown below.

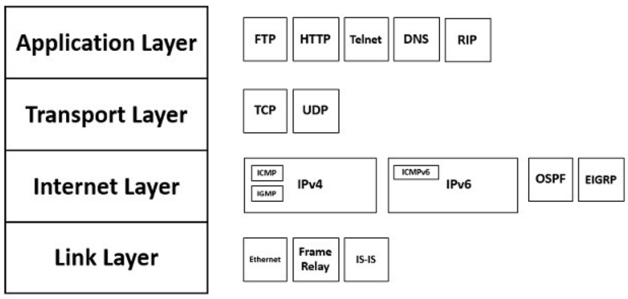

Fig 6: TCP Model

Transport Layer: This layer is responsible for managing end-to-end communication between applications.The data transfer is reliable and transparent.This layer includes protocols like TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). In our project, we employ TCP instead of UDP because of the following reasons.

Connection Oriented - TCP first establishes connection between the devices and only on successful connection, it transfers and receives the data. Also, the transmission and reception happens in the same order.Hence TCP is called a connection-oriented protocol.

UDP is connectionless, which means that it does not establish a connection before transmitting data. This means that there is no handshaking process to ensure that the receiver is ready to receive the data. So, it cannot be 100% guaranteed that the data being transmitted will reach the receiver and also it may so happen that the packets received are out-of-order.

Reliable delivery - TCP is a reliable protocol that guarantees the delivery of data without errors or loss. It achieves this by using sequence numbers and acknowledgments to ensure that all packets are received in the correct order. If a packet is lost or damaged during transmission, TCP will retransmit it until it is successfully received.

UDP on the other hand, does not provide reliable delivery of data like TCP. It does not have a mechanism for retransmitting lost or damaged packets, which can result in data loss or corruption.

Hence, because UDP lacks the reliability and features of TCP, TCP is generally preferred for applications that require reliable delivery, ordering of packets, and flow and congestion control. Since the truck platooning system is out with other road users, the precision and timing has to be of utmost priority and the instructions sent by the server have to reach the clients in time and correctly. If not, then it may lead to accidents or unpleasant situations.

VI. GPU IMPLEMENTATION

author - Sachin Kumar

The GPU (Graphical Processing Unit) is another processor unit like CPU (Central Processing Unit). It plays an important role where the tasks have to be performed in parallel e.g. 3D rendering. While the CPU is used to perform the general tasks, managing with the scheduler, but the GPU is optimized to perform parallel processing. There are various frameworks like CUDA, OpenCL, OneAPI, HIP, etc. Out of all the frameworks and apis, the OpenCL provides an approach where the large number of GPUs are targeted from different vendors. OpenCL (Open Computing Language) is standard for cross-platform and parallel processing. It is a low level api for heterogeneous programming. In OpenCL, the kernels are defined which are the functions which execute on the GPU threads.

The code implementation of the using this approach is standard which is efficient and reliable in the Distance Safety Checking, where the vehicle has to decide whether it should accelerate or decelerate.

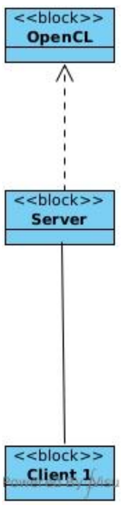

Fig 7. Block Diagram of the Platooning System

The topology used is based on the Server and Client, where the client platoons connect with the server, which is the main platoon and then server uses the kernel in order to process the data in parallel in the GPU.

The structure of the code is defined using three classes: -
1. Server

```
class Server {
public:
    Server() {…
    ~Server() {…

    void run() {…

private:
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt;
    int addrlen;
    char buffer[1024] = { 0 };
    char* hello;
    ClCode *clCodeObj;
};
```

Fig 8: *Server* Class Structure

The Server class uses the C++ socket library to create the socket and binds the server to the port 8080. It has a *run* function which waits for the clients to connect and then gets the data of vehicles.

2. Client

```
class Client {
public:
    Client() {…
    ~Client() {…
    void run() {…
private:
    int sock, valread, client_fd;
    struct sockaddr_in serv_addr;
    char* hello;
    char buffer[1024] = { 0 };

};
```

Fig 9: *Client* Class Structure

The Client class also uses C++ socket library to create the socket and then use a *run* function which connects with the socket server and then sends the data to the server and also, the server sends the data back to client in order to recognize that the data is received successfully and send the next data.

`

3. ClCode

```
class ClCode {
    public:
    ClCode();
    ~ClCode();
    void distanceCompare(std::size_t numberOfVehicles,
    int vehicleDistances[], int safetyDistances[], bool status[]);
    private:
    std::vector<cl::Platform> all_platforms;
    cl::Context *context;
    cl::Device default_device;
    cl::Program *program;
};
```

Fig 10: *ClCode* Class Structure

The ClCode class uses the OpenCL library to create the kernel, which is a function used to execute the logic on the GPU threads.

```
if(all_platforms.size()==0){
    std::cout<<" No platforms found. Check OpenCL installation!\n";
    exit(1);
}
std::cout << "size of platforms " << all_platforms.size() << std::endl;
cl::Platform default_platform=all_platforms[0];
std::cout << "Using platform: "<<default_platform.getInfo<CL_PLATFORM_NAME>()<<"\n";

//get default device of the default platform
std::vector<cl::Device> all_devices;
default_platform.getDevices(CL_DEVICE_TYPE_ALL, &all_devices);
if(all_devices.size()==0){
    std::cout<<" No devices found. Check OpenCL installation!\n";
    exit(1);
}
default_device=all_devices[0];
std::cout<< "Using device: "<<default_device.getInfo<CL_DEVICE_NAME>()<<"\n";
```

Fig 11: Checking GPU Platform

The above code is used to check if OpenCL is installed on the system and then check which Platform is available and what device is available for computation on the GPU.

```
"   void kernel distance_safety(global const int* vehicleDistance, global const int* safetyDistance, global bool* status){
"           if(vehicleDistance[get_global_id(0)] > safetyDistance[get_global_id(0)]) {                    "
"               status[get_global_id(0)] = true;                                          "
"           }                                                                     "
"           else {                                                                "
"               status[get_global_id(0)] = false;                                          "
"           }                                                                     "
"   }                                                                             ";
```

Fig 12: Kernel to compare distance and safety distance

The above code is used as a kernel to execute the data on the GPU threads. Once, the distance data of all the vehicles is received, then it calls the ClCode instance *distanceCompare* function which sends the data to the GPU kernel in order to perform the logic where it needs to return whether the vehicle should accelerate or decelerate in order to maintain the safety distance.

```
// create buffers on the device
cl::Buffer buffer_vehicleDistances(*context,CL_MEM_READ_WRITE,sizeof(int)*numberOfVehicles);
cl::Buffer buffer_safetyDistances(*context,CL_MEM_READ_WRITE,sizeof(int)*numberOfVehicles);
cl::Buffer buffer_status(*context,CL_MEM_READ_WRITE,sizeof(bool)*numberOfVehicles);


//create queue to which we will push commands for the device.
cl::CommandQueue queue(*context,default_device);

//write arrays A and B to the device
queue.enqueueWriteBuffer(buffer_vehicleDistances,CL_TRUE,0,sizeof(int)*numberOfVehicles,vehicleDistances);
queue.enqueueWriteBuffer(buffer_safetyDistances,CL_TRUE,0,sizeof(int)*numberOfVehicles,safetyDistances);
```

Fig 13: Buffer creation and data writing

The sequence of the execution of the code follows a certain standard in which firstly, the buffers on the device are created, which are used to communicate between the host (CPU) and the device (GPU). After this, the data from the host is written into the buffer data type variables.

```
//run the kernel
cl::Kernel kernel_add=cl::Kernel(*program,"distance_safety");
kernel_add.setArg(0,buffer_vehicleDistances);
kernel_add.setArg(1,buffer_safetyDistances);
kernel_add.setArg(2,buffer_status);
queue.enqueueNDRangeKernel(kernel_add,cl::NullRange,cl::NDRange(numberOfVehicles),cl::NullRange);
queue.finish();

printf("queue finish\n");
//read result C from the device to array C
queue.enqueueReadBuffer(buffer_status,CL_TRUE,0,sizeof(bool)*numberOfVehicles,status);
```

Fig 14: Adding arguments to kernel and execution

The arguments are set on the kernels and then enqueue the kernel in order to execute the kernel in the GPU threads. After the execution, the buffer result data is read back to the host data type variable in order to further process the data.

The execution of the programs server and client is shown in the below figures, which uses the command line arguments in order to send the data of the vehicles from the client to the server and then from the server to the GPU device.

```
Number of vehicles: 5
ok
Enter distance for vehicle 0: 40
ok
Enter distance for vehicle 1: 60
ok
Enter distance for vehicle 2: 70
ok
Enter distance for vehicle 3: 40
ok
Enter distance for vehicle 4: 90
Vehicle 0 should Accelerate.
Vehicle 1 should Decelerate.
Vehicle 2 should Decelerate.
Vehicle 3 should Accelerate.
Vehicle 4 should Decelerate.
```

Fig 15: Client CLI

The client socket connects with the server and also sends the distance values to the server. The server sends back the message in order to recognize the data has been received.

```
size of platforms 1
Using platform: NVIDIA CUDA
Using device: NVIDIA GeForce RTX 3060 Laptop GPU
server is listening
number of vehicles 5
vehicle value received
vehicle value received
vehicle value received
vehicle value received
vehicle value received
queue finish
```

Fig 16: Server CLI

VII. HARDWARE SELECTION

author - Hamida Aliyeva

Proper hardware must be implemented for truck platooning to successfully work. The reason is that the hardware components of the truck platoon are highly dependent on the ability of the hardware components to interact with each other, collect and interpret information, and execute directives accurately and collaboratively. Below you can find the hardware components that every truck platooning should have.

Firstly, GPS sensors are essential truck platooning components which help trucks in platoon to pinpoint their position and follow a defined route. In order to keep the trucks in the platoon on the right path and preserve a safe distance from one another GPS sensors are needed. In general, the working principle of GPS sensors is to receive signals from GPS satellites around the Earth. In order to find the location of the truck distance between GPS satellites and the truck is calculated by the GPS sensors. Truck's speed and direction is also determined by the sensors as they compare its location with time. Each truck has GPS sensors and they must always be in communication with one another and specially with the lead truck. As a result, the leader truck will send commands to the following trucks. Moreover, this communication will be mutual, meaning that the lead truck will also be able to receive information from the following trucks, such as their speed and position. Finally, it is also possible to use GPS sensors to detect upcoming changes in the route, like hills or curves. This will help to regulate the vehicle's speed and position correspondingly.

Secondly, another essential part of the truck platooning is radar sensors. They enable the vehicles in the platoon to recognize and avoid hazards while maintaining a secure following distance. In order to give accurate direction, velocity and location of other vehicles and obstacles in the area around radar sensors are utilized. Working principle of radar sensors is to emit electromagnetic waves and calculate the time from waves to bounce back once hitting surrounding objects.Object's location and relative speed is calculated with the help of radar sensors. They have a wide frequency range in which they can operate, such as millimeter wave or microwave frequencies, which enables them to deliver accurate and reliable information in various environmental situations. Radar sensors are used in trucks to maintain proper distance between trucks in a platoon. With the help of radar sensors the leader truck is able to identify obstacles and change the speed and direction if necessary. In addition, radar sensors will allow following trucks to identify speed and position of the leader truck and modify theirs correspondingly. Moreover, radar sensors let trucks detect upcoming changes in the route, like hills or curves and modify their behaviors appropriately. Finally, when using radar sensors it will be possible to synchronize truck behavior, thus this will result in higher level of accuracy and coordination in truck platooning.

In addition, camera sensors are another essential part in the truck platooning.They enable trucks to observe and assess their surroundings. In order to detect and recognize barriers, signs, and other important elements, cameras offer a precise visual picture of the area surrounding the trucks. Camera sensors contain lenses and sensors that they use to capture images and videos of the surrounding area. They are able to generate information by measuring and identifying color and light that later can be used to produce the environment's digital representation. Cameras are typically employed in truck platooning so that an autonomous driving system can use the visual data produced to calculate the movement of the truck. Camera sensors are used in truck platooning to identify and categorize impediments like other cars, people, and road signs. Precise visual images of the area around the truck can be recognized by camera sensors. These images are used to define impediments and their locations. Finally, camera sensors in truck platooning systems can also be utilized in order to supply with the necessary information about the path ahead, like traffic signs, lane dividers and read markings.

V2V (Vehicle-to-Vehicle) modems are lastly essential truck platooning components. Trucks are communicated with one another via V2V modems and their behaviors are coordinated by these modems as well. Wireless communications techniques are used by V2V modems in order to let trucks tell each other about their direction, location velocity and other vehicle identifying details. The working principle of V2V modems is to use a special short range communication protocol that has a concrete frequency range and allows high-speed transmission between vehicles. A special type of wireless connection is used in modems in order to interchange data and make instant communication with one another. V2V modems are used in truck platooning in order to allow communication and behavior coordination amongst the trucks. The following vehicles in the platoon can use the lead truck's ability to relay information about its speed, position, and direction to modify their behavior and keep a safe following distance. Finally, the trucks' ability to communicate with one another about roadblocks, traffic, and other pertinent information via the modems can help the platoon operate more effectively and safely.

VIII. JAVA IMPLEMENTATION

author - Supadma Kadabi

In the given client-server-based model, the client is a thin-client and it has only a presentation layer. The server however contains all the remaining layers. The proposed solution has used socket programming in JAVA for implementation on the CPU system.

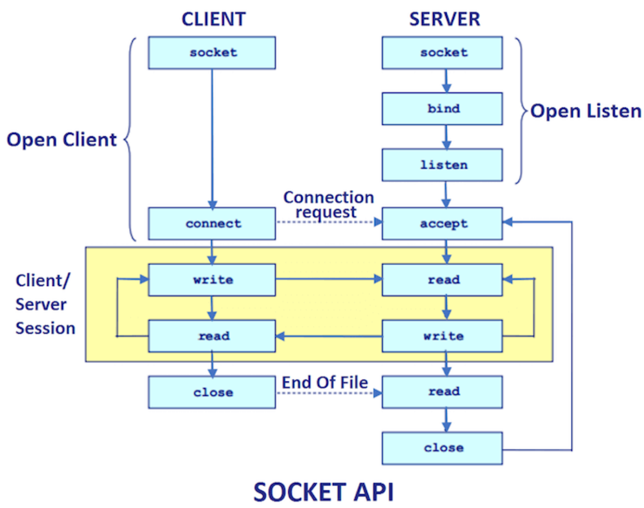

Fig 17: Socket Programming

The Socket is a special communication link that supports two-way communication between server and client. Socket programming is a programming technique where we can connect two sockets written in different languages, for example, a server socket can be written in C programming and Client can be written in Java. The Socket connection can be either connection oriented or connection less. Different sockets identify each other using the socket address. Socket address consists of protocol type, IP Address and port number. Socket programming has a wide range of applications, it can be used to connect remote and local computers, between different networks etc. Sockets can be used as stand alone processes or network applications. In the given figure 17, the server socket has main functions like bind, listen and accept. Once a client socket is initiated it requests for communication with the server. Once socket communication is established, the data transfer occurs using buffers.

```java
ServerSocket _server;
private static final int PORT = 9090;
int _clientCount;
public ServerHelper() {
    _clientCount = 0;
    try {
        _server = new ServerSocket(PORT);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Fig 18: ServerSocket and server instance

Java Programming:

The java programming language provides advanced classes and libraries which are user friendly and less prone to attacks. It helps the beginners in developing complex code with ease. Socket programming in java is connection based. It has classes like *ServerSocket, Socket* which contains in-built methods for most of the required functionalities. This programming language is more advanced and safer for beginners to start with coding. The *Socket* class of Java contains in-built functions like *bind(), listen()* combined in the *accept()* method. In the proposed system, the data transfer is done serially using buffers.

Implementation of Server:

In this system multiple classes are used to perform the different functionalities. The server contains different classes like *Server, ServerThread, ServerHelper* which combined perform all the basic functionalities of the Main Vehicle.

The *ServerHelper* class mainly controls the client connection and disconnections. It also displays required comments for understanding the system status.

```java
class ServerHelper {
    // private int disconnect_client;
    Socket socket;

    public ServerHelper() {···

    void run() {
        try {
            while (true) {
                socket = _server.accept();

                ServerThread serverThread = new ServerThread(socket, _clientCount);
                _clientCount = _clientCount + 1;
                System.out.println("***********************************" + '\n' + "     New connection Alert" + '\n'
                        + "***********************************");
                System.out.println("Number of client connected: " + _clientCount);
                System.out.println("[SERVER] Active threads:" + ServerThread.activeCount());
                System.out.println("--------------------------------" + '\n' + "--------------------------------");

                serverThread.start();
                if (ServerThread.activeCount() == 0) {
                    socket.close();
                    _server.close();
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
```

Fig 19: Class *ServerHelper*

The *serverTheread* class contributes in interacting with all the computational classes for instructing the client with next steps to be taken. It is the main interface between the platoons and the Main Vehicle.

The *monitorPlatoonData* is used to monitor each variable updated by the platoon. It performs all the comparisons using multiple methods like *monitor_distance(), monitor_signal_strength(), monitor_speed(), Object_detection_status(), Weather_monitoring().* These methods further interact with Velocity_euqations class for calculating the values at the lowest level possible. The function Weather_monitoring() helps the server to understand the environment and road conditions, using this method it updates the Platoons about the weather and the caution to be taken while driving.

The *Veloctiy_equations* uses the basic displacement equations to calculate the values of acceleration and time using simple variables like *accerleration_fin, time, ini_vel, fin_vel, distance* etc. It also converts the measurement from m/sec to km/hr before and after calculations for better understanding and application purposes.

```java
public class Velocity_equations {

    // rounding off the final out come to 2 decimal places
    private static final DecimalFormat dec = new DecimalFormat(pattern: "0.00");
    public double acceleration;
    public double acceleration_fin;
    public double time;

    public String vel_calculation(int ini_vel, int fin_vel, double distance) {
        // convert velocity to m/s, so we divide the given kmph by 3.6 to m/s
        // v^2 -u^2 = 2as, V-> Final Velocity, u -> initial Velocity a -> acceleration,
        // s -> distance
        // Standard distance is 40m, Platoon is at 50m, then it has to cover 10 m with
        // given acceratation
        double ini_vel_m = (ini_vel / 3.6);
        double fin_vel_m = (fin_vel / 3.6);
        acceleration = ((Math.pow(fin_vel_m, b: 2)) - (Math.pow(ini_vel_m, b: 2))) / (2 * distance);
        // convert accelration back to kmph^2
        acceleration_fin = acceleration * 3.6;

        return (dec.format(acceleration_fin));
    }

    public String time_calculation(int ini_vel, int fin_vel) {
        // calculate the time for which the acceleration needs to be maintained
        // v = u + at, a-> calculated from previous function, t -> time
        double ini_vel_m = (ini_vel / 3.6);
        double fin_vel_m = (fin_vel / 3.6);
        time = (fin_vel_m - ini_vel_m) / (acceleration);

        return (dec.format(time));
    }
}
```

Fig 20: Class *Veloctiy_equations*

There are individual classes defined in the system for Main vehicle and Platoon which handle all the variables that are required for monitoring the system. These classes are defined as **serializable** because they are used to transfer the values between the classes serially using buffers. The Main vehicle variables are defined in the Class *Leader,* the variables *are distance, speed, signal_strength, weather* and *location*. These variables help the system to compare each platoon status and make decisions as required. The datatype *Location* is also a class that has two variable *lat* and *lng* which defines the latitude and longitude of the vehicle.

```
public class leader implements Serializable {

    /*...
    private double distance;
    private int speed, signal_strength;
    private String weather;
    private Location location;
```

Fig 21: Class *leader*

Similarly, the platoon is also defined as a class *Platoon,* it contains certain special variables like *Object_detected_inmtrs, object_detection* along with standard variables available in *Leader* class. The extra variables of this class help the platoons in monitoring the objects/ external traffic to update the server and get its instructions. This feature is discussed again in further topics.

```
public class platoon implements Serializable {

    private double distance;
    private int speed, signal_strength;
    private Location location;
    private boolean quit;
    private int Object_detected_inmtrs;
    private boolean object_detection;
```

Fig 22: Class *platoon*

Implementation of Client:

The client class in this system describes the Platoons, it initiates all the platoon variables and sends them to the Main vehicle on regular intervals. Here it is realized by clicking the enter button for easy usage. The platoon connects to main function using IP address and port. In this system the standard IP address “127.0.0.1” and standard port “9090” is used for connection. In this function multiple print statements are also involved for understanding purposes. The first input given to the system in this simulation use a function called *Random* which generates random values at every instance. However, after initialisation the inputs are incremented at every instance to observe the system behavior. It is recommended to input realtime data to get realtime output. It has other functionalities like verifying signal strength and object detection.

Object detection: This is handled by variables *Object_detected_inmtrs, object_detection* and the method *monitor_object_distance().* The platoon constantly verifies if the object that is an intruder car is approaching itself and updates the Main vehicle when it is less than 10m away from it.

Signal Strength: There are two types of signal strength information in this system, one where the Main vehicle verifies if platoon is still able to connect to it and the other is platoon checking for its ability to connect with Main vehicle. When the signal strength is less than platoon expected strength, the platoon checks for nearby parking locations and updates the parking location to the server.

IX. RESULTS AND DISCUSSION

author - Supadma Kadabi

The output of this system can be analyzed in terms of five cases, based on the situations of the system. these cases can be described as below:

A. Normal Conditions: When the platoon is following the Main Vehicle and the values are maintained as per Main Vehicle, it acknowledges the platoon to continue in the same state

```
---------------------------------
---------------------------------
[CLIENT] Connected to server on port: 9090 ip: 127.0.0.1
---------------------------------
---------------------------------
>Hit Enter to send data to Server--

Sending details to Server....
Distance: 30.0
Signal Strength: 80
Speed: 50
Location: 33.0 160.0
Object Distance: 20
Quit: false
------------------------------
[CLIENT] Server sent:
[SERVER] information to Client[1]       Server Speed: 50       BEAUTIFUL DAY DRIVE SAFE
------------------------------
Waiting for Server response...
[CLIENT] Server sent:
[SERVER] instructions to Client[1]      Distance okay       Signal strength okay    Speed okay     Road Safe no object
---------------------------------
---------------------------------
```

Fig 23: Output Normal conditions

B. Speed Value less than Server: When the platoon is slower than the Main vehicle, it updates the platoon to increase the speed by giving it the exact values for acceleration and time

```
[CLIENT] Connected to server on port: 9090 ip: 127.0.0.1
---------------------------------
---------------------------------
>Hit Enter to send data to Server--

ending details to Server....
istance: 50.0
ignal Strength: 50
peed: 30
ocation: -15.0 124.0
bject Distance: 20
uit: false
------------------------------
CLIENT] Server sent:
SERVER] information to Client[1]       Server Speed: 50       BEAUTIFUL DAY DRIVE SAFE
------------------------------
aiting for Server response...
CLIENT] Server sent:
SERVER] instructions to Client[1]      You are far from next vehicle 20.0   Move faster      Signal strength okay    You are 20 slower than leade
, accelerate by 11.11  for the given time 1.80        Road Safe no object
---------------------------------
---------------------------------
```

Fig 24: Platoon slower than Main Vehicle

C. Speed Value more than Server:When the platoon is traveling at a higher speed and is approaching the next vehicle, the Main vehicle instructs it to slow down.

```
[CLIENT] Connected to server on port: 9090 ip: 127.0.0.1
---------------------------------
---------------------------------
>Hit Enter to send data to Server--

Sending details to Server....
Distance: 10.0
Signal Strength: 50
Speed: 60
Location: 68.0 178.0
Object Distance: 20
Quit: false
------------------------------
[CLIENT] Server sent:
[SERVER] information to Client[1]       Server Speed: 50       BEAUTIFUL DAY DRIVE SAFE
------------------------------
Waiting for Server response...
[CLIENT] Server sent:
[SERVER] instructions to Client[1]      You are close to next Vehicle 20.0  slow down        Signal strength okay    You are -10 faster than leader.Slow down!
 decelerate by -7.64  for the given time 1.31          Road Safe no object
---------------------------------
---------------------------------
```

Fig 25: Platoon faster than Main Vehicle

D. Object Detected: When Platoon updates the Main vehicle about object/external vehicle approaching the platoon, the Main vehicle instructs the Platoon to slow down and maintain the standard distance from the vehicle.

Fig 26: Object detected

E. Signal Strength lost: When the signal strength is less than Main vehicle standards, it sends a warning message to the Platoon.

Fig 27: Signal Strength less than Main vehicle Std.

When the Platoon strength is less than standard signal strength expected by platoon to connect to the Main vehicle, it parks in the nearby available parking spot and updates the location.

Fig 28: Platoon parked itself and updated the location

X. CONCLUSION:

author - Supadma Kadabi

The proposed system is successful in establishing the socket connections between one server socket and multiple client sockets using TCP protocol. The server socket a.k.a. Main vehicle was able to monitor the following Platoons. The GPU programming was successfully implemented to monitor speed of the platoons using OpenCL programming.

XI. REFERENCES

1. R. L. R. Maata, R. Cordova, B. Sudramurthy and A. Halibas, "Design and Implementation of Client-Server Based Application Using Socket Programming in a Distributed Computing Environment," 2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), Coimbatore, India, 2017, pp. 1-4, doi: 10.1109/ICCIC.2017.8524573.
2. https://www.javatpoint.com/socket-programming
3. Zhu, J., et al. "Automotive Radar Sensors in Silicon Technologies." Proceedings of the IEEE, vol. 102, no. 9, pp. 1426-1442, 2014.
4. Hai Zhang -"Architecture of Network and Client Server model" 25 Juli 2013.
5. Li, S., Li, X., and Yu, S. "Camera sensors in autonomous vehicles: A review." Computer Vision and Image Understanding, vol. 203, pp. 1-25, 2020.
6. Federal Highway Administration. "Understanding V2V: What it Is and What it Does." [Online]. Available: https://www.fhwa.dot.gov/advancedresearch/pubs/17053/17053.pdf.

XII. ACKNOWLEDGMENT

We would like to thank **Prof. Dr. Stefan Henkler** for his lecture and inputs during the exercise session which were helpful in understanding the concepts about the Embedded Systems.

XIII. APPENDIX

Git:
Main branch - used throughout the development
https://github.com/VibhaHippargi/DPSTruckPlatooning/tree/main

Final branch – contains relevant files only
https://github.com/VibhaHippargi/DPSTruckPlatooning/tree/final

ANNEXURE

Contribution % of all team members :

Vibhashree Hippargi – 25%
Sachin Kumar – 25%
Supadma Kadabi – 25%
Hamida Aliyeva – 25%

GPU Programming

Code for Server Socket

server.cpp

```
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#include "../include/cl_code.hpp"
#define PORT 8080

class Server {
public:
		Server() {
				clCodeObj = new ClCode();
				opt = 1;
				addrlen = sizeof(address);
				hello = "Hello from server";
				// Creating socket file descriptor
				if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
						perror("socket failed");
						exit(EXIT_FAILURE);
				}

				printf("server is listening\n");
				// Forcefully attaching socket to the port 8080
				if (setsockopt(server_fd, SOL_SOCKET,
											SO_REUSEADDR | SO_REUSEPORT, &opt,
											sizeof(opt))) {
						perror("setsockopt");
						exit(EXIT_FAILURE);
				}
				address.sin_family = AF_INET;
				address.sin_addr.s_addr = INADDR_ANY;
				address.sin_port = htons(PORT);

				// Forcefully attaching socket to the port 8080
				if (bind(server_fd, (struct sockaddr*)&address,
								sizeof(address))
						< 0) {
						perror("bind failed");
						exit(EXIT_FAILURE);
				}
		}
		~Server() {
				// closing the connected socket
				close(new_socket);
				// closing the listening socket
				shutdown(server_fd, SHUT_RDWR);
```

```
		// delete pointer
		// delete hello;
		delete clCodeObj;
	}

	void run() {
		if (listen(server_fd, 3) < 0) {
			perror("listen");
			exit(EXIT_FAILURE);
		}
		if ((new_socket
			= accept(server_fd, (struct sockaddr*)&address,
					(socklen_t*)&addrlen))
			< 0) {
			perror("accept");
			exit(EXIT_FAILURE);
		}
		int numberOfVehicles;
		valread = read(new_socket, buffer, 1024);
		numberOfVehicles = atoi(buffer);
		printf("number of vehicles %d\n", numberOfVehicles);
		int vehicleDistances[numberOfVehicles], safetyDistances[numberOfVehicles];
		bool status[numberOfVehicles];
		hello = "ok";
		for (size_t vehicleId = 0; vehicleId < numberOfVehicles; vehicleId++){
			send(new_socket, hello, strlen(hello), 0);
			valread = read(new_socket, buffer, 1024);
			vehicleDistances[vehicleId] = atoi(buffer);
			printf("vehicle value received\n");
		}

		for (size_t vehicleId = 0; vehicleId < numberOfVehicles; vehicleId++){
			safetyDistances[vehicleId] = 50;
		}
		clCodeObj->distanceCompare(numberOfVehicles, vehicleDistances, safetyDistances, status);
		printf("got the values");

		for (size_t vehicleId = 0; vehicleId < numberOfVehicles; vehicleId++){
			if(status[vehicleId]) {
				hello = "1";
			}
			else {
				hello = "0";
			}
			send(new_socket, hello, strlen(hello), 0);
			valread = read(new_socket, buffer, 1024);
		}
	}

private:
	int server_fd, new_socket, valread;
	struct sockaddr_in address;
	int opt;
	int addrlen;
	char buffer[1024] = { 0 };
	char* hello;
	ClCode *clCodeObj;
};
int main(int argc, char const* argv[])
{
	Server socketServerObj = Server();
	socketServerObj.run();
	return 0;
}
```

Code for Client Socket

client.cpp

```
#include <iostream>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#include <stdlib.h>
#define PORT 8080


class Client {
public:
  Client() {
    sock = 0;
    hello = "Hello from client";
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
      printf("\n Socket creation error \n");
      exit(-1);
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

  }
  ~Client() {
    // closing the connected socket
    close(client_fd);
    // delete pointer
    // delete hello;
  }
  void run() {
    // Convert IPv4 and IPv6 addresses from text to binary
    // form
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)
      <= 0) {
      printf(
        "\nInvalid address/ Address not supported \n");
      exit(-1);
    }

    if ((client_fd = connect(sock, (struct sockaddr*)&serv_addr,
                             sizeof(serv_addr)))
           < 0) {
           printf("\nConnection Failed \n");
           exit(1);
      }
    int numberOfVehicles;
    std::cout << "Number of vehicles: ";
    std::cin >> numberOfVehicles;


    // std::string vehicleDistancesStr = "";

    hello = (char *) std::to_string(numberOfVehicles).c_str();
    send(sock, hello, strlen(hello), 0);
    int distance;
    for(std::size_t vehicleId = 0; vehicleId < numberOfVehicles; vehicleId++) {
      valread = read(sock, buffer, 1024);
      printf("%s",buffer);
```

```
        std::cout << "\nEnter distance for vehicle " << vehicleId <<": ";
        std::cin >> distance;
        hello = (char *) std::to_string(distance).c_str();
        send(sock, hello, strlen(hello), 0);
      }
      bool status[numberOfVehicles];

      for(std::size_t vehicleId = 0; vehicleId < numberOfVehicles; vehicleId++) {
        valread = read(sock, buffer, 1024);
        status[vehicleId] = atoi(buffer);
        hello = "ok";
        send(sock, hello, strlen(hello), 0);
      }

      std::string speed;
      for(std::size_t vehicleId = 0; vehicleId < numberOfVehicles; vehicleId++) {

        speed = (status[vehicleId]) ? "Decelerate." : "Accelerate.";

        std::cout << "Vehicle " << vehicleId << " should " << speed << std::endl;
      }
    }
private:
        int sock, valread, client_fd;
        struct sockaddr_in serv_addr;
        char* hello;
        char buffer[1024] = { 0 };

};

int main(int argc, char const* argv[])
{
  Client clientSocket = Client();
  clientSocket.run();
        return 0;
}
```

Code OpenCL

cl_code.hpp

```
#define CL_HPP_TARGET_OPENCL_VERSION 300

#include <iostream>
#include <CL/opencl.hpp>

class ClCode {
  public:
  ClCode();
  ~ClCode();
  void distanceCompare(std::size_t numberOfVehicles, int vehicleDistances[], int safetyDistances[], bool status[]);
  private:
  std::vector<cl::Platform> all_platforms;
  cl::Context *context;
  cl::Device default_device;
  cl::Program *program;
};
```

cl_code.cpp

```
#include "../include/cl_code.hpp"

ClCode::ClCode() {
cl::Platform::get(&all_platforms);
```

```
    if(all_platforms.size()==0){
        std::cout<<" No platforms found. Check OpenCL installation!\n";
        exit(1);
    }
    std::cout << "size of platforms " << all_platforms.size() << std::endl;
    cl::Platform default_platform=all_platforms[0];
    std::cout << "Using platform: "<<default_platform.getInfo<CL_PLATFORM_NAME>()<<"\n";

    //get default device of the default platform
    std::vector<cl::Device> all_devices;
    default_platform.getDevices(CL_DEVICE_TYPE_ALL, &all_devices);
    if(all_devices.size()==0){
        std::cout<<" No devices found. Check OpenCL installation!\n";
        exit(1);
    }
    default_device=all_devices[0];
    std::cout<< "Using device: "<<default_device.getInfo<CL_DEVICE_NAME>()<<"\n";


    context = new cl::Context({default_device});

    cl::Program::Sources sources;

    std::string kernel_code=
            "    void kernel distance_safety(global const int* vehicleDistance, global const int* safetyDistance, global bool* status){      "
            "        if(vehicleDistance[get_global_id(0)] > safetyDistance[get_global_id(0)]) {                    "
            "            status[get_global_id(0)] = true;                      "
            "        }                                   "
            "        else {                               "
            "            status[get_global_id(0)] = false;                      "
            "        }                                   "
            "  }                                        ";
    sources.push_back({kernel_code.c_str(),kernel_code.length()});


    program = new cl::Program(*context,sources);
    if(program->build({default_device})!=CL_SUCCESS){
        std::cout<<" Error building: "<<program->getBuildInfo<CL_PROGRAM_BUILD_LOG>(default_device)<<"\n";
        exit(1);
    }
}
ClCode::~ClCode(){
    delete program;
    delete context;
}


void ClCode::distanceCompare(std::size_t numberOfVehicles, int vehicleDistances[], int safetyDistances[], bool status[]) {
    // create buffers on the device
    cl::Buffer buffer_vehicleDistances(*context,CL_MEM_READ_WRITE,sizeof(int)*numberOfVehicles);
    cl::Buffer buffer_safetyDistances(*context,CL_MEM_READ_WRITE,sizeof(int)*numberOfVehicles);
    cl::Buffer buffer_status(*context,CL_MEM_READ_WRITE,sizeof(bool)*numberOfVehicles);


    //create queue to which we will push commands for the device.
    cl::CommandQueue queue(*context,default_device);

    //write arrays A and B to the device
    queue.enqueueWriteBuffer(buffer_vehicleDistances,CL_TRUE,0,sizeof(int)*numberOfVehicles,vehicleDistances);
    queue.enqueueWriteBuffer(buffer_safetyDistances,CL_TRUE,0,sizeof(int)*numberOfVehicles,safetyDistances);


    //run the kernel
```

```
    cl::Kernel kernel_add=cl::Kernel(*program,"distance_safety");
    kernel_add.setArg(0,buffer_vehicleDistances);
    kernel_add.setArg(1,buffer_safetyDistances);
    kernel_add.setArg(2,buffer_status);
    queue.enqueueNDRangeKernel(kernel_add,cl::NullRange,cl::NDRange(numberOfVehicles),cl::NullRange);
    queue.finish();

    printf("queue finish\n");
    //read result C from the device to array C
    queue.enqueueReadBuffer(buffer_status,CL_TRUE,0,sizeof(bool)*numberOfVehicles,status);

}
```

code for Makefile

```
Makefile
CC = g++

CFLAGS = -g -std=c++17 -Wall -pedantic


all:server client

server: src/server.cpp
	$(CC) $(CFLAGS) -o bin/server.out src/server.cpp src/cl_code.cpp -I ./include -lOpenCL

client: src/client.cpp
	$(CC) $(CFLAGS) -o bin/client.out src/client.cpp -I ./include


clean:
	$(RM) bin/cl_code.out bin/server.out bin/client.out

run_server:
	./bin/server.out

run_client:
	./bin/client.out
```

JAVA Programming:

SERVER CODE:

```
import java.io.IOException;
import java.lang.String;
import java.net.ServerSocket;
import java.net.Socket;
import java.io.EOFException;
import java.io.ObjectInputStream;
import java.io.PrintWriter;
import java.lang.Thread;
import java.io.Serializable;
import java.text.DecimalFormat;
public class server {

    public static void main(String[] args) throws IOException {

        System.out.println("----------------------------------" + '\n' + "----------------------------------");
        System.out.println(Messages.START.getMessage());
        System.out.println("----------------------------------" + '\n' + "----------------------------------");
        ServerHelper _serverHelper = new ServerHelper();
        _serverHelper.run();
```

```java
    }
}
class ServerHelper {
    // private int disconnect_client;
    Socket socket;

    public ServerHelper() {
        _clientCount = 0;
        try {
            _server = new ServerSocket(PORT);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    void run() {
        try {
            while (true) {
                socket = _server.accept();

                ServerThread serverThread = new ServerThread(socket, _clientCount);
                _clientCount = _clientCount + 1;
                System.out.println("**********************************" + '\n' + "      New connection Alert" + '\n'
                        + "**********************************");
                System.out.println("Number of client connected: " + _clientCount);
                System.out.println("[SERVER] Active threads:" + ServerThread.activeCount());
                System.out.println("---------------------------------" + '\n' + "---------------------------------");

                serverThread.start();
                if (ServerThread.activeCount() == 0) {
                    socket.close();
                    _server.close();
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

    ServerSocket _server;
    int _clientCount;
    private static final int PORT = 9090;

}
class ServerThread extends Thread {

    private Socket socket;
    private int count;

    public ServerThread(Socket socket, int count) {
        this.socket = socket;
        this.count = count + 1;

    }

    @Override
    public void run() {

        while (true) {

            try {
                ObjectInputStream clientInput = new ObjectInputStream(socket.getInputStream());
```

```java
    PrintWriter output = new PrintWriter(socket.getOutputStream(), true);
    monitorPlatoonData check = new monitorPlatoonData();
    platoon clientPlatoon;
    String object_detect = "Road Safe no object";
    leader lead1 = new leader();

    try {

        String weather_con = check.Weather_monitoring(lead1.getWeather());
        // output.println(weather_con);
        System.out.println("Server Status:  " +
                "Server Speed: " + lead1.getSpeed() + "\n" +
                "Server Location: " + lead1.getLocation() + "\n");

        clientPlatoon = (platoon) clientInput.readObject();
        System.out.println("Client[" + count + "] has sent : ");
        System.out.println("Distance: " + clientPlatoon.getDistance() + '\n'
                + "Signal Strength: " + clientPlatoon.getSignal_strength() + '\n'
                + "Speed: " + clientPlatoon.getSpeed() + '\n'
                + "Location: " + clientPlatoon.getLocation().lat
                + " " + clientPlatoon.getLocation().lng + '\n'
                + "Object Status " + clientPlatoon.getobject_detection() + "\n"
                + "Quit: " + clientPlatoon.getQuit());

        // Object detection,change of command
        if (clientPlatoon.getobject_detection() == true) {
            object_detect = check.Object_detection_status(clientPlatoon.getObject_detected_inmtrs(),
                    clientPlatoon.getSpeed());
        }
        // call monitor methods
        String dist_result = check.monitor_distance(clientPlatoon.getDistance());
        String signal_status = check.monitor_signal_strength(clientPlatoon.getSignal_strength());
        String speed_status = check.monitor_speed(clientPlatoon.getDistance(),
                clientPlatoon.getSpeed());
        output.println("[SERVER] information to Client[" + count + "]   \t " + "Server Speed: "
                + lead1.getSpeed() + "\t" + weather_con);
        output.println("[SERVER] instructions to Client[" + count + "]   \t " + dist_result + "   \t"
                + signal_status + "   \t" + speed_status + "   \t" + object_detect);

        System.out.println("[SERVER] instructions to Client[" + count + "]\n " + dist_result + "\n"
                + signal_status + "\n" + speed_status + "\n" + weather_con + "\n" + object_detect);
        System.out.println("------------------------------- ");

    } catch (ClassNotFoundException e) {
        e.printStackTrace();
        // Catch also all other exceptions.
        break;
    } catch (Exception e) {
        // Print what exception has been thrown.
        System.out.println(e);
        break;
    }
}
// When the client disconnects then the server experiences EOF (End-Of-File).
catch (EOFException e) {
    System.out.println(e);
    break;
}
// any other exceptions
catch (IOException e) {
    // e.printStackTrace();
    System.out.println(e);
    break;
}
```

```
        }
    }

}
public class monitorPlatoonData {

    Velocity_equations vel_equ = new Velocity_equations();
    leader lead = new leader();
    private double std_distance = 30.0;
    private int std_speed = lead.getSpeed();
    private int std_signal_strength = 50;

    public String monitor_distance(double distance) {
        String s;
        double r;
        if (distance < std_distance) {
            r = std_distance - distance;
            s = "You are close to next Vehicle " + r + "  slow down  ";
        } else if (distance > 30) {
            r = distance - 30;
            s = "You are far from next vehicle " + r + "   Move faster   ";
        } else {
            s = "Distance okay";
        }
        return s;
    }

    public String monitor_signal_strength(int signalstrength) {
        String s;
        int r;
        if (signalstrength < std_signal_strength) {
            r = std_signal_strength - signalstrength;
            s = "You are " + r + " less than minimum required signal strength  ";
        } else if (signalstrength > 100) {
            r = signalstrength - 100;
            s = "CAUTION: Signal Strength to high";
        } else // need to add one more condition to check if distance is in required lenght
        {
            s = "Signal strength okay";
        }
        return s;

    }

    public String monitor_speed(double distance, int speed) {
        // speed -> Current Speed of the Platoon
        // distance -> Distance between the Platoon and the Vehicle in front of it
        String s;
        int r;
        String acceleration;
        String time;
        double differnce_distance = distance - std_distance;
        // difference_distance -> the distance which needs to be maintained,
        // i.e. if the platoon is at 40m distance from Vehicle before it and Standard
        // distance is
        // 30m then 10m needs to be covered
        differnce_distance = (differnce_distance < 0) ? -differnce_distance : differnce_distance;
        if (differnce_distance == 0) {
            s = "You are at safe distance, reduce speed to " + std_speed;
        } else {
            if (speed < std_speed) {
                acceleration = vel_equ.vel_calculation(speed, std_speed, differnce_distance);
                time = vel_equ.time_calculation(speed, std_speed);
```

```
            r = std_speed - speed;
            s = "You are " + r + " slower than leader, accelerate by " + acceleration + "  for the given time " + time;
        } else if (speed > std_speed) {
            acceleration = vel_equ.vel_calculation(speed, std_speed, differnce_distance);
            time = vel_equ.time_calculation(speed, std_speed);
            r = std_speed - speed;
            s = "You are " + r + " faster than leader.Slow down! decelerate by " + acceleration
                + "  for the given time "
                + time;
        } else {
            s = "Speed okay";
        }
    }
    return s;

  }

  public String Object_detection_status(double distance, int speed) {
    // code to handle the status when object is detected
    String ret_str = "";
    double differnce_distance = std_distance - distance;
    if (distance <= 20) {
        ret_str = "You are " + distance + "  units from the Object, Slow down to maintain distance   "
            + differnce_distance;
    } else {
        ret_str = "No object Detected ";
    }
    return ret_str;
  }

  public String Weather_monitoring(String Weather) {
    String ret_str = "";
    if (Weather.contains("Road_slippery")) {
        ret_str = "CAUTION: ROAD SLIPPERY ";
    } else if (Weather.contains("Snowing ")) {
        ret_str = "CAUTION: SNOWING ";
    } else if (Weather.contains("Rainy")) {
        ret_str = "CAUTION: RAINING ";
    } else {
        ret_str = "BEAUTIFUL DAY DRIVE SAFE";
    }
    return ret_str;
  }

}

class Location implements Serializable {
   public double lat, lng;

   Location(double lat, double lng) {
      this.lat = lat;
      this.lng = lng;
   }
};

public class leader implements Serializable {
   private double distance;
   private int speed, signal_strength;
   private String weather;
   private Location location;

   public String getRandomWeather() {
      //Returns random weather conditions
      String[] weatherArray = { "Rainy", "Normal", "Road_Slippery", "Snowing" };
```

```java
        String weather = weatherArray[(int) (Math.random() * weatherArray.length)];

        return weather;
    }

    public double getDistance() {
        return distance;
    }

    public void setDistance(double distance) {
        this.distance = distance;
    }

    public int getSpeed() {
        return speed;
    }

    public void setSpeed(int speed) {
        this.speed = speed;
    }

    public int getSignal_strength() {
        return signal_strength;
    }

    public void setSignal_strength(int signal_strength) {
        this.signal_strength = signal_strength;
    }

    public Location getLocation() {
        return location;
    }

    public void setLocation(Location location) {
        this.location = location;
    }

    public String getWeather() {
        return weather;
    }

    public void setWeather(String weather) {
        this.weather = weather;
    }

    public leader() {
        this.distance = 50.0;
        this.speed = 50;
        this.signal_strength = 100;
        this.location = new Location(50, 60);
        this.weather = getRandomWeather();
    }

    public leader(double _distance, int _speed, int _signal_strength, Location _location, String _weather) {
        this.distance = _distance;
        this.speed = _speed;
        this.signal_strength = _signal_strength;
        this.location = _location;
        this.weather = _weather;
    }

}
public enum Messages {
    START("Server started!"),
```

```
    CONNECT("Connecting client"),
    ACCELERATE(""),
    DECELERATE(""),
    DISCONNECT("Disconnecting client");

    private String message;
    Messages(String message) {
        this.message = message;
    }

    public String getMessage() {
        return message;
    }
}
public class Velocity_equations {

    // rounding off the final out come to 2 decimal places
    private static final DecimalFormat dec = new DecimalFormat("0.00");
    public double acceleration;
    public double acceleration_fin;
    public double time;

    public String vel_calculation(int ini_vel, int fin_vel, double distance) {
        // convert velocity to m/s, so we divide the given kmph by 3.6 to m/s
        // v^2 -u^2 = 2as, V-> Final Velocity, u -> initial Velocity a -> acceleration,
        // s -> distance
        // Standard distance is 40m, Platoon is at 50m, then it has to cover 10 m with
        // given acceratation
        double ini_vel_m = (ini_vel / 3.6);
        double fin_vel_m = (fin_vel / 3.6);
        acceleration = ((Math.pow(fin_vel_m, 2)) - (Math.pow(ini_vel_m, 2))) / (2 * distance);
        // convert accelration back to kmph^2
        acceleration_fin = acceleration * 3.6;

        return (dec.format(acceleration_fin));
    }

    public String time_calculation(int ini_vel, int fin_vel) {
        // calculate the time for which the acceleration needs to be maintained
        // v = u + at, a-> calculated from previous function, t -> time
        double ini_vel_m = (ini_vel / 3.6);
        double fin_vel_m = (fin_vel / 3.6);
        time = (fin_vel_m - ini_vel_m) / (acceleration);

        return (dec.format(time));
    }
}
```

CLIENT CODE:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.Random;
import java.io.Serializable;
import java.util.ArrayList;

public class client {

    private static final String SERVER_IP = "127.0.0.1";
    private static final int SERVER_PORT = 9090;
```

```
private static int singal_strength_std = 20;
private Random random;

client() {
   random = new Random();
}

public double getRandomDist() {
   //Returns random distance values
   double[] distancesArray = { 10.0, 35.0, 40.0, 20.0, 50.0 };
   double distance = distancesArray[(int) (Math.random() * distancesArray.length)];

   return distance;
}

public int getRandomSpeed() {
   //Return random speed values
   int[] speedArray = { 20, 30, 40, 50, 60, 70, 80, 90, 120 };
   int speed = speedArray[(int) (Math.random() * speedArray.length)];

   return speed;
}

public int getRandomSignal() {
   //return random signal values
   int[] signalArray = { 30, 40, 50, 60, 70, 80, 90, 100 };
   int signal = signalArray[(int) (Math.random() * signalArray.length)];

   return signal;
}

public int getRandomObjectDistance() {
   //Returns random object distance
   int[] ObjectDistanceArray = { 5, 10, 15, 20 };
   int signal = ObjectDistanceArray[(int) (Math.random() * ObjectDistanceArray.length)];
   return signal;
}

public Location getRandomLocation() {
   /**
    * range from -90 to 90 for latitude and -180 to 180 for longitude.
    * returns the random
    */
   int maxLat = 90;
   int minLat = -90;
   int maxLng = 180;
   int minLng = -180;

   double randomLat = random.nextInt(maxLat + 1 - minLat) + minLat;
   double randomLng = random.nextInt(maxLng + 1 - minLng) + minLng;
   Location randomLocation = new Location(randomLat, randomLng);
   return randomLocation;
}

public int monitor_object_distance(int Objdistance) {
   if (Objdistance <= 20 && Objdistance > 15) {
      return 0;
   }
   if (Objdistance <= 15 && Objdistance < 10) {
      return 1;
   }
   if (Objdistance <= 10 && Objdistance > 5) {
      return 2;
   }
```

```java
        if (Objdistance < 5 && Objdistance >= 1) {
            return 3;
        }

        return 0;
    }

    public static void main(String[] args) throws IOException {
        double dist = 0;
        int signal = 0;
        int speed = 0;
        Location location;
        Location location_1 = new Location(0, 0);
        int objectDistance = 0;
        int obj_Det = 0;
        System.out.println("----------------------------------" + '\n' + "----------------------------------");
        Socket socket = new Socket(SERVER_IP, SERVER_PORT);
        System.out.println("[CLIENT] Connected to server on port: " + SERVER_PORT + " ip: " + SERVER_IP);
        System.out.println("----------------------------------" + '\n' + "----------------------------------");
        int flag_1 = 0;
        client clientobj = new client();
        parking_space park_assit = new parking_space();
        Location park_spot = new Location(0, 0);
        BufferedReader toreadserverresponse = new BufferedReader(new InputStreamReader((socket.getInputStream())));
        BufferedReader keyboard = new BufferedReader(new InputStreamReader(System.in));
        while (true) {
            System.out.println(">Hit Enter to send data to Server-- ");
            String command = keyboard.readLine();
            platoon p = new platoon();
            ObjectOutputStream objectOutputStream = new ObjectOutputStream(socket.getOutputStream());
            if (command.equalsIgnoreCase("Q")) {
                System.out.println("Incorrect Input");
                p.setQuit(false);
            } else {
                if (flag_1 == 0) {
                    dist = clientobj.getRandomDist();
                    signal = clientobj.getRandomSignal();
                    speed = clientobj.getRandomSpeed();
                    objectDistance = clientobj.getRandomObjectDistance();
                    location = clientobj.getRandomLocation();
                    location_1 = location;
                    obj_Det = clientobj.monitor_object_distance(objectDistance);
                    flag_1 = 1;
                } else {
                    dist = dist + 10;
                    signal = signal + 10;
                    speed = speed + 5;
                    location = location_1;
                    objectDistance = clientobj.getRandomObjectDistance();
                    obj_Det = clientobj.monitor_object_distance(objectDistance);
                }
                if (obj_Det < 1) {
                    p.setobject_detection(false);
                }
                if (obj_Det >= 1) {
                    p.setobject_detection(true);
                    p.setObject_detected_inmtrs(objectDistance);
                }
                p.setDistance(dist);
                p.setSpeed(speed);
                p.setSignal_strength(signal);
                p.setLocation(location);
                p.setQuit(false);
            }
```

```
            if (p.getSignal_strength() < singal_strength_std) {
                park_spot = park_assit.Parking_assitance(p.getLocation());
                System.out.println("Client parked due to communication loss parking spot is latitude " + park_spot.lat
                        + " Longitude is " + park_spot.lng + "\n");
                p.setQuit(true);
            } else {
                System.out.println("Sending details to Server....");
                System.out.println("Distance: " + p.getDistance() + '\n'
                        + "Signal Strength: " + p.getSignal_strength() + '\n'
                        + "Speed: " + p.getSpeed() + '\n'
                        + "Location: " + p.getLocation().lat + " " + p.getLocation().lng + '\n'
                        + "Object Distance: " + p.getObject_detected_inmtrs() + '\n'
                        + "Quit: " + p.getQuit());
                System.out.println("------------------------------- ");

                objectOutputStream.writeObject(p); // sending data to server
                String serverresponse_1 = toreadserverresponse.readLine();
                System.out.println("[CLIENT] Server sent: " + '\n' + serverresponse_1);
                System.out.println("------------------------------- ");
                System.out.println("Waiting for Server response... ");
                String serverresponse = toreadserverresponse.readLine();
                System.out.println("[CLIENT] Server sent: " + '\n' + serverresponse);
            }

            System.out.println("----------------------------------" + '\n' + "----------------------------------");
            if (p.getQuit()) {
                break;
            }

        }
        socket.close();
        System.exit(0);
    }

}

public class platoon implements Serializable {

    private double distance;
    private int speed, signal_strength;
    private Location location;
    private boolean quit;
    private int Object_detected_inmtrs;
    private boolean object_detection;

    public boolean getobject_detection() {
        return object_detection;
    }

    public void setobject_detection(boolean object_detection) {
        this.object_detection = object_detection;
    }

    public int getObject_detected_inmtrs() {
        return Object_detected_inmtrs;
    }

    public void setObject_detected_inmtrs(int object_detected_inmtrs) {
        Object_detected_inmtrs = object_detected_inmtrs;
    }

    public boolean getQuit() {
        return quit;
    }
```

```
    public void setQuit(boolean quit) {
        this.quit = quit;
    }

    public double getDistance() {
        return distance;
    }

    public void setDistance(double distance) {
        this.distance = distance;
    }

    public int getSpeed() {
        return speed;
    }

    public void setSpeed(int speed) {
        this.speed = speed;
    }

    public int getSignal_strength() {
        return signal_strength;
    }

    public void setSignal_strength(int signal_strength) {
        this.signal_strength = signal_strength;
    }

    public Location getLocation() {
        return location;
    }

    public void setLocation(Location location) {
        this.location = location;
    }


    public platoon() {
        this.distance = 0.0;
        this.speed = 0;
        this.signal_strength = 0;
        this.location = new Location(0, 0);
        // this.weather = "";
        this.Object_detected_inmtrs = 20;
        this.object_detection = false;
        this.quit = false;

    }

    public platoon(double _distance, int _speed, int _signal_strength, Location _location, String _weather,
            int object_detected_inmtrs, boolean object_detection, boolean _quit) {
        this.distance = _distance;
        this.speed = _speed;
        this.signal_strength = _signal_strength;
        this.location = _location;
        // this.weather = _weather;
        this.Object_detected_inmtrs = object_detected_inmtrs;
        this.object_detection = object_detection;
        this.quit = _quit;

    }

}
```

```
public class parking_space {
    // used to suggest nearby parking locations
    // this is local data stored in each platoon and they can access them everytime
    // signal is lost from the server
    // number of parking spots here is minimum, it can be added as required
    public Location Parking_assitance(Location current) {
        Location l1 = new Location(50, 50);
        Location l2 = new Location(60, 80);
        Location l3 = new Location(70, 90);
        Location l4 = new Location(90, 100);
        ArrayList<Location> locations = new ArrayList<Location>();
        locations.add(l1);
        locations.add(l2);
        locations.add(l3);
        locations.add(l4);
        Location ret_fin = new Location(0, 0);
        for (int i = 0; i < locations.size(); i++) {

            if (current.lat <= locations.get(i).lat) {
                ret_fin = locations.get(i);
            } else {
                ret_fin.lat = 100;
                ret_fin.lng = 150;
            }
        }
        return ret_fin;
    }
}
```