A Dynamic Map-based Framework for Real-Time Mapping of Vehicles and their Surroundings

Mariem Maiouak[1] and Tarik Taleb[1,2]
[1] Communications and Networking department, Aalto University, Espoo, Finland
[2] Centre for Wireless Communications, University of Oulu, Oulu, Finland
Emails: mariem.maiouak@aalto.fi, tarik.taleb@aalto.fi

***Abstract*— The great attraction that unmanned vehicles have gained in the past years has marked this century as the era of automation. In particular, automated vehicles have recently become the center of attention of many research activities. All of which have had one goal in common: achieving a complete or partial automation of driving functions, all while ensuring safety and security of other traffic agents. It is with this purpose in mind that the concept of dynamic maps has been introduced to allow vehicles to be aware of their surroundings and have precise knowledge of their environment and all of its components. Aiming for a mapping system that sets out a layered view of a vehicle vicinity ranging from highly-static to highly-dynamic data layers, researchers have taken different approaches. In this paper, we present a system, inspired from the concept of dynamic maps, that collects data from user vehicles and maps them out by location and time, keeping a history of all recorded information. We then evaluate the performance of the system module in charge of localization and tracking of users' devices, and present and discuss the obtained results.**

I. INTRODUCTION

The key idea beneath the concept of dynamic maps is to provide autonomous vehicles with access to traffic data ranging from permanently static, such as roads, intersections, and nature (e.g., trees), to highly dynamic data, such as locations of moving persons and objects (e.g., pedestrians, cyclists, and buses). In order to form a better understanding of the requirements on a dynamic maps system, we take a closer look at the process of automated driving. The levels of automated driving have been set out by the National Highway Traffic Safety Administration (NHTSA) of the US Department of Transportation as follows:

- **Level 0:** The vehicle is completely controlled by a human driver. There is no automated driving.
- **Level 1:** Only a few driving functions are automated, the rest are performed by a human driver.
- **Level 2:** At this level, the driving is done autonomously; at least the cruise control and lane steering are conducted autonomously. However, there is a driver present monitoring the situation and may intervene if needed.
- **Level 3:** At this level the driver is not required to monitor the system as closely as in Level 2.
- **Level 4:** All driving and parking functions are autonomous and the driver is not expected to intervene at any point.

The concept of automated driving hinges on the capability of vehicles to locate and detect objects in their environments, analyze the collected data, and alter their behaviors accordingly [1][2]. The tasks required of an autonomous vehicle can be then categorized in the following way:

- **Sensing –** an autonomous vehicle is equipped with a number of sensors that make it aware of its surroundings. The data collected from these sensors is combined for maximum accuracy. Some of these sensors are GPS, Inertial Measurement Unit (IMU), LiDAR, Cameras, and Radar.
- **Perception –** after acquiring the data, the system analyzes it to build an understanding of its surroundings. This process goes through three main steps, namely the localization of the vehicle itself and its nearby objects, the detection and classification of these objects, and the tracking of the identified objects.
- **Decision –** based on the information established in the previous stages, the system should then generate an adequate action plan for the situation. This stage of autonomous driving mainly utilizes probabilistic processes and Markov chains. It aims to complete the following tasks:
  - Prediction – the system set in place should be able to predict future movements of nearby vehicles.
  - Path planning – all possible paths that the vehicle can take are established. The best one is then identified based on a cost function.
  - Obstacle avoidance – this process works on two levels. The first is pro-active; in this level the planned path is constantly being updated to avoid collision. The second level is reactive, and comes into play in case of emergency cases whereby the first level fails.

For our work, we focus on the first two sub-processes of automated driving. One of the main challenges. raised in these two phases, lies in the accurate positioning of the detected objects, which, in turn, cannot be achieved without being able to localize the vehicles precisely in real-time and with a small latency. Our implementation, in comparison to other research works [3][4][5][6], also allows the users to keep a history log of all data collected by the device cameras. In fact, the idea of a highly dynamic precise mapping system

of road components has inspired us to create a platform that goes a step further than mapping the different road components by location, and provides the user with an effective tool to map out the objects by timestamp as well. To this end, after analyzing the visual data that the system has received from the streaming devices, and extracting the necessary information to identify and classify the detected objects, it keeps a copy of every stream and associates it with a geographical path. Along with the drawn stream path that the end user can access, he can also view the detected objects, their locations and timestamps, and a visual log of their time of detection.

The remainder of this paper is organized in the following fashion. Section II covers the state-of-the-art of dynamic maps. Section III describes the envisioned architecture along with the different components of our dynamic map system. Section IV explores our framework API and the technologies used for enabling it. This section also discusses in dept how the device tracking system works and the process that a video feed undergoes from the moment it is captured through a device camera to when it is streamed back through the user interface. Section V describes the experiment set to evaluate the device tracking module to measure its latency and provide an accurate representation of its usage of the system resources. To this end, the section provides results on the system performance under different settings and with different numbers of users, and propose solutions for its optimization. Finally, the paper concludes in Section VI.

II. Related work

The idea of dynamic maps was first introduced within the 7.3.1 work project [7] of the SAFESPOT project in 2010. This project was co-funded by the European Commission Information Society Technologies [8]. The idea behind it was to create a connected dynamic network between traffic agents in order to help vehicles become more aware of their environments and enable them to avoid obstacles without the help of a human driver. The concept of a Local Dynamic Map (LDM) was standardized a year later in 2011 through the ETSI TR 102 863 (V1.1.1) report [9]. This report was the first to introduce the layered view of LDM whereby data provided by it can range from highly dynamic data, like pedestrian and vehicle locations, to highly static data, like roads and intersections. The second ETSI report, ETSI EN 302 895 (V1.1.0) final draft [10], came in 2014 as an extension to the first report. The ISO standards ISO/TS 17931:2013 [11] and ISO/TS 18750:2015 [12] came respectively in 2013 and 2015 and presented a similar dynamic maps architecture to that of the ETSI standards.

Some research works have also proposed an implementation of dynamic maps, such as the work of Netten et al. that introduced DynaMap [13], and the work of Shimada et al. that proposed an implementation of LDM [14] based on the SAFESPOT project specifications. Ulbrich et al. presented another implementation that combined static map data with dynamic sensory collected data to enable automated driving [15]. El Zoghby et al. introduced their distributed approach to dynamic maps whereby vehicles cooperate within a VANet (Vehicular Ad-hoc Network) in order to form a complete view of LDM [16]. Xu et al. proposed an implementation of 3D point cloud map based on data collected from LiDAR units on vehicles [17]. This concept is the same as the high-definition 3D maps that were created within the project launched in 2016 under the support of the Japanese government program, Strategic Innovation Promotion Program Innovation of Automated Driving for Universal Services (SIP-adus).

III. System architecture

A. System overview

In order to achieve the concept of dynamic maps and successfully utilize it in an Intelligent Transport System (ITS) environment, many requirements, that prove to be quite demanding in terms of computational power, need to be satisfied. As a first step, we chose to focus our work on tracking the user devices' GPS locations in real time and saving a log of their streams.

Our system allows users to stream their environment through a camera lens. Once these video feeds are received by the server, they are processed to extract pertinent data around the different traffic agents that they have captured. The detected objects are then highlighted and the new video frames are streamed back. The system also tracks the users' moving vehicles in real-time. Every piece of information that is collected through the different service modules is mapped out and is publicly accessible through a web platform. This platform provides its users with a variety of maps that showcase the collected data filtered by zones. The end users can share and view their movements in real-time.

B. Components and architecture

The idea behind our system was inspired from the concept of LDMs that contain highly dynamic data and other semi-dynamic data (e.g., traffic signs). Our system keeps a history log of all traffic agents captured through the cameras of its clients, and in this aspect, may be utilized in a law enforcement and security setting. As depicted in Fig.1, it consists of the following modules:

- **Object detection and tracking module:** Responsible for processing the video streams and detecting the different traffic agents appearing in their frames and classifying them.
- **Streaming module:** In charge of receiving the streams from the user devices, redirecting them to the object detection module and streaming back the live feeds with the detected objects.
- **Local Dynamic map:** A web platform that gives the users access to a number of maps. These maps display user devices, the streams generated by them and the objects captured by their cameras.
- **Device tracking module:** In order to satisfy the LDM timing and positioning accuracy requirements, the traffic agents locations in the database needs to be up to date

at all time. To this end, this module keeps track of the users GPS coordinates through a real-time database.

- **Data management module:** It stores and retrieves user, object and stream data into and from the database.
- **Authentication module:** It allows the users to subscribe to the system and be authenticated before accessing its services.

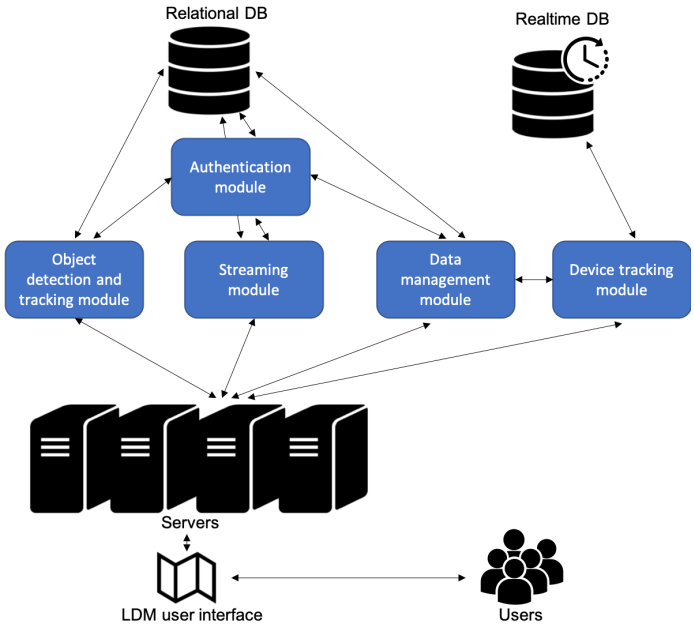


Fig. 1: Overview of the envisioned system architecture.

IV. Framework API and enabling technologies

A. Scenarios and system functionality

As portrayed in the sequence diagram in Fig.2, our system functionality can be summarized in three main services:

- **Real-time object detection**
  The user first signs into the system through his mobile device. Upon signing up, the user is issued a token that will be returned to him each time he signs into the system. This token is then used when the user sends a streaming request to the video streaming and processing server. After checking the authenticity of the request, the server starts receiving the video stream. This process automatically triggers the video processing service, by redirecting the video data to it. The object detection module then extracts the features from the video frames, and classifies the detected objects. It then draws boxes around these objects, and indicates the class and level of accuracy of each detection. Upon doing so, the object detection module starts sending the new frames, as they are generated, to the streaming module. The latter then serves these new frames to the end user using the same token that was provided to the end user in the beginning.
- **Device tracking**
  This service is distributed between the mapping platform side and the mobile system. From the mobile system side, the user can either allow location tracking or not. If he chooses to allow it, a communication channel is then established between the device and the real-time database. Through this channel, the device location is uploaded in the database every time it changes. This service runs in the background of the device.
  From the mapping platform side, in order to display the vehicle movements, the server tracking module establishes a web socket connection with the real-time database, and listens for changes to the vehicles GPS locations. Once a change is detected, it is automatically displayed on the map that is accessible to the end user.
- **Live video feed mapping and visualization**
  Aside from the location data of the user devices, the mapping platform allows users to access a range of other information around the recorded streams. These information include the detected objects and the vehicles path history. The mapping service filters these data by location and separates them by geographical zones. By doing so, the system creates a hierarchy of pages that branch out from a root worldview page, where the user can access traffic and population information about different countries. We discuss in more depth the mapping services offered by the platform in the following subsection.

B. Data mapping framework API

In order to map out the different data recorded by the system as efficiently as possible, we opted for a hierarchical approach. Through this approach, we can list the following services that are offered by the data mapping framework API:

- **getCountries**: This service retrieves a list of 250 countries. For each country it gets the following data: GPS location, population, currency, area, capital, language, flag, number of recorded streams, number of users, and number of detected objects by class. The system also assigns a rating to each country based on the number of users and streams hosted in it.
- **getCities**: This service takes as input a country ID and returns a list of its cities along with representative images from the Google Places API for each city.
- **getVehicles**: It returns the moving vehicles located in a particular city. For each vehicle, it provides the brand, the VIN (Vehicle Identification Number), the user associated with it, the last known location if the vehicle tracking is off or the vehicle authentication token if the tracking is on.
- **getStreams**: This service gives access to the streams that have been or are being recorded by a vehicle. If the stream is live the service returns the URL where the edited stream with the detected objects is being broadcasted. And if the stream is no longer live, it returns a link to the recorded video of this stream along with the recording time, date and location.
- **getObjects**: Given that the system associates each video frame with a GPS location, this service returns, for a specific stream, a list of detected objects, their classes, their time and location of detection and a snapshot

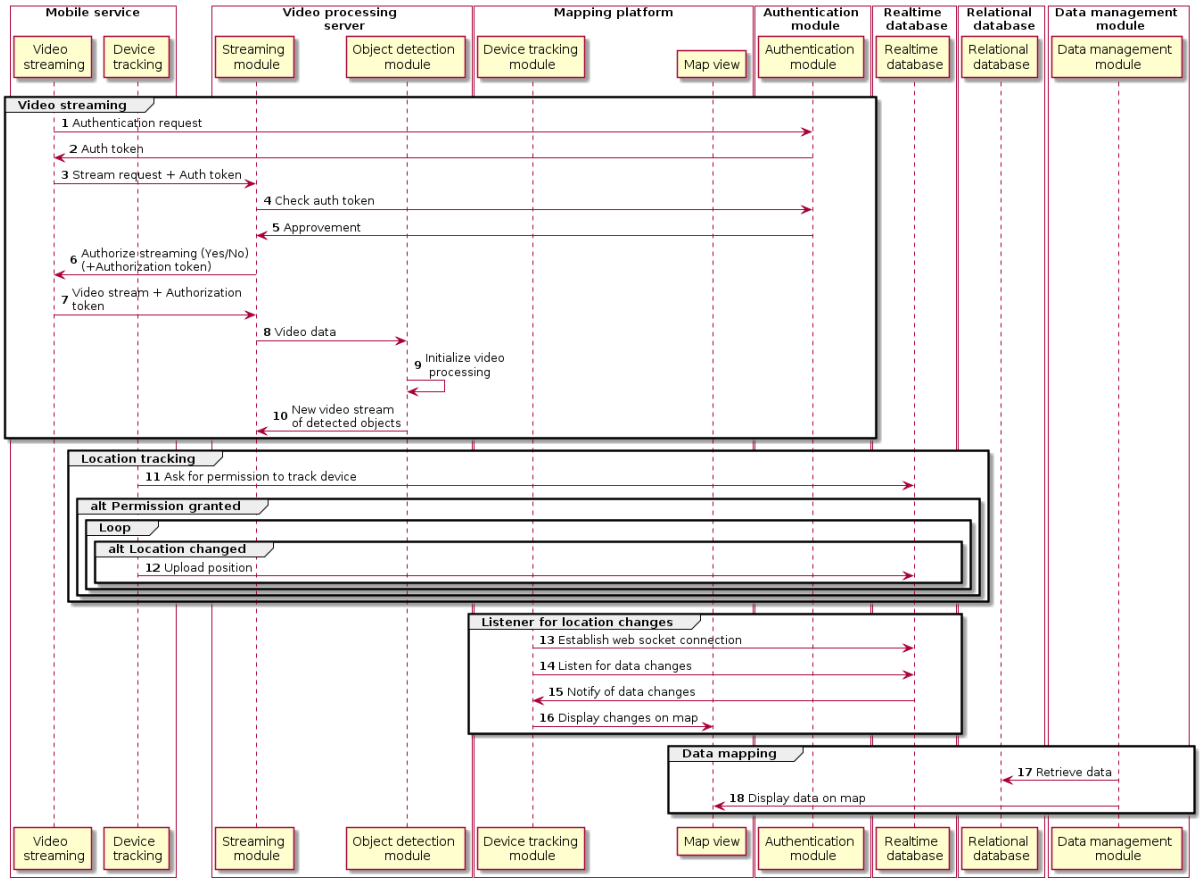


Fig. 2: Sequence diagram describing the signaling messages exchanged for the different operations of the system.

image that has been captured at the time of detection. This service also provides, for a given stream, the path that the vehicle followed when recording it.

C. Tools and technologies

For the object detection service, we used Tensorflow-GPU with CUDA 9.0 and OpenCV. We relied on the open source pre-trained model of SqueezeDet [18] to classify the objects. For the streaming service, we used two RTMP streaming servers: one for receiving the vehicle streams and redirecting them to the object detection service, and another for broadcasting the updated streams with the detected objects to the end users. For the device tracking module, we used Firebase real-time database with websockets to listen for data changes to vehicle locations. And finally for the data mapping module, we used Google maps API and Google Places API.

V. Experiment and results

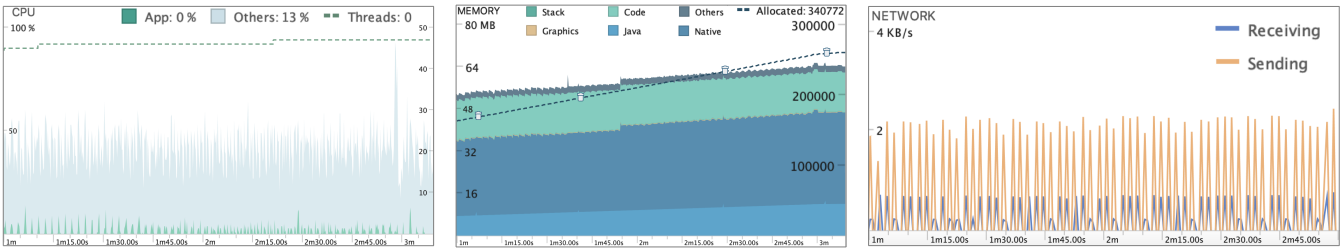
In order to test the performance of our tracking module, we ran the service for 60 minutes and recorded the results. We present a 3 minute sample of the evaluation process for both the client and the server side.

A. Client app evaluation

For the client app, we run the service on a moving device that is constantly updating its location in the real-time database, and record the CPU, memory and network usage of the app, to determine the device resource consumption by the service. When observing the results depicted in Fig.3a, we observe that the CPU usage of the app during the tracking process is negligible, as it does not surpass 1%, compared to the 13% that other running apps consume. The memory usage of the app is displayed in a stacked graph (Fig.3b) that shows the memory allocated to Java and native objects, the memory used by Java and native stacks in our app, the memory used for graphics and the memory allocated to code and resources. The number of Java objects allocated by our app is displayed on the right y-axis of the graph. We can see that the memory usage of the app does not exceed 80Mb. This number, while not small, is acceptable, considering that the app is constantly sending location updates to the real-time database. This mechanism is also portrayed in Fig.3c where we display the mobile device network usage variations throughout the experiment. We can observe that the spikes in the network usage mirror the recurrent data transfers corresponding to the location updates.

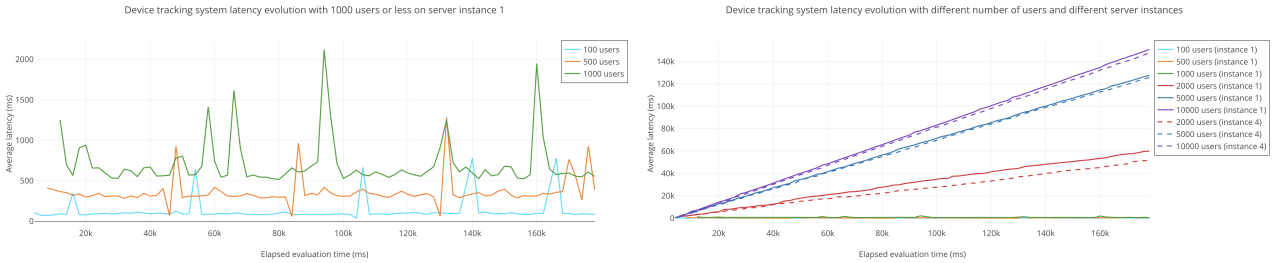
B. Server side evaluation

For the server side, we simulated the movement of varying numbers of vehicles running the client app, we then accessed the mapping platform that loads the moving vehicles on the map, and listens for updates to their location through



(a) CPU usage variations. (b) Memory usage variations. (c) Network usage variations.

Fig. 3: Resource usage of the device tracking client app.



(a) System latency with 1000 users or less measured on the server instance 1.

(b) System latency for different numbers of users and using different server flavors.

Fig. 4: Device tracking module latency results in different settings.

a web socket to the real-time database. We then measured the latency between the time of the location update in the real-time database, and its retrieval by the location update listeners of the mapping module. We measured the average latency every two seconds throughout the whole evaluation time. We start off by running this experiment on one server flavor (instance 1) with varying numbers of users (100, 500, 1000, 2000, 5000, and 10000), (Fig.4a). We notice that the system performs very well with up to 1000 users, even when tested on the least powerful server flavor (instance 1). In fact for these numbers of users, the system latency is generally under 1 second, and never goes over 2 seconds (Fig.4a).

Therefore, we record the system latency for four different server flavors for 2000, 5000, and 10000 moving vehicles. The specifications of these instances are shown in Table I. We noticed that the difference in latency between the four instances was very small, so we chose to show only the results of the best and worst performing instances in the same diagram, to showcase how small the difference is (Fig.4b).

TABLE I: Server flavors

Name	vCPUs	Memory (Gb)	Clockrate (GHz)	Bandwidth (Gb/s)
Instance 1	2	8	2.3	0.45
Instance 2	4	16	2.3	0.7
Instance 3	8	32	2.3	1
Instance 4	16	64	2.4	10

After observing that the system performs relatively well with up to 1000 users, regardless of the underlying instance, and after noticing that even a big upgrade to the server resources did not improve the results much, we came up with a possible solution that would ensure that the system would experience an average latency in case of 10000 vehicles comparable to that in the case of 1000 vehicles. Our proposed implementation (Fig.5) would set in place an orchestrator service that would receive, for example, a location update request for N vehicles. This service would calculate, based on a previously set up algorithm, the number of server instances it needs to create in order to respond to the request. The system should be able to scale up and down depending on the need. After defining the number of server instances needed, the orchestrator would then function as a load balance, determining which vehicles should be assigned to which servers based on the location of the vehicles and their distance from the servers that would listen for changes to their locations [19]. It then distributes the location update requests across the server instances, who then, in turn, create a listener service for each vehicle.

VI. CONCLUSIONS

In this paper, we introduced a different approach for a dynamic maps implementation. This approach aimed to keep a log of traffic data and allow users to efficiently filter and access the recorded vehicle video streams and the information that has been extracted from them. The implementation we proposed allows the mapping of traffic agents by space and time. The paper also presented a few experiments that we have conducted on the device tracking module to calculate its latency under different settings and with varying numbers of users. The obtained results indicated that the best solution to achieve a smaller latency regardless of the number of users or the server specifications is to implement an orchestrator that

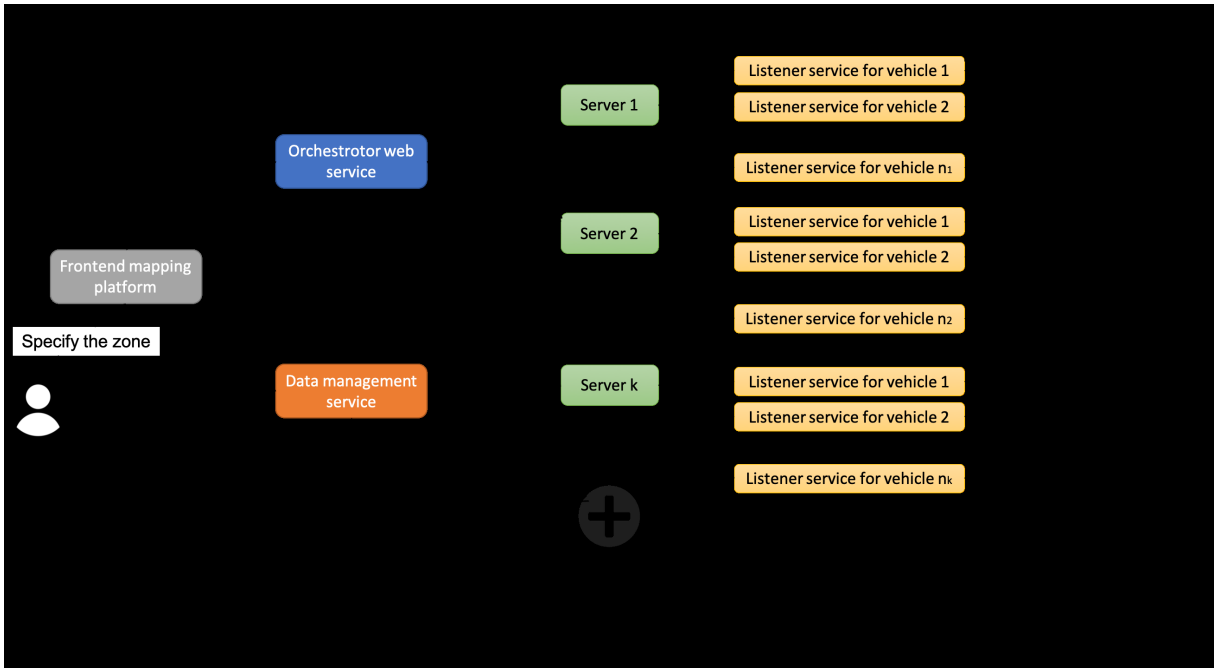


Fig. 5: A solution for ensuring short latency in vehicle location update and tracking regardless the number of connected vehicles.

would rely on several optimization algorithms, to distribute the location update requests on multiple server instances depending on the need. We will implement this solution in future research works in order to optimize the overall system performance.

ACKNOWLEDGMENT

This work was supported in part by the European Unions Horizon 2020 research and innovation programme under the MATILDA project (Grant No. 761898), and by the Academy of Finland 6Genesis Flagship (Grant No. 318927).

REFERENCES

[1] G. Lugano, "Virtual assistants and self-driving cars," 2017 15th International Conference on ITS Telecommunications (ITST), Warsaw, 2017, pp. 1-5.

[2] Y. Lin, P. Wang and M. Ma, "Intelligent Transportation System(ITS): Concept, Challenge and Opportunity," 2017 ieee 3rd international conference on big data security on cloud (bigdatasecurity), ieee international conference on high performance and smart computing (hpsc), and ieee international conference on intelligent data and security (ids), Beijing, 2017, pp. 167-172.

[3] S. Noh, K. An and W. Han, "Toward highly automated driving by vehicle-to-infrastructure communications," 2015 15th International Conference on Control, Automation and Systems (ICCAS), Busan, 2015, pp. 2016-2021.

[4] A. Rachedi, H. Badis, "BadZak: An hybrid architecture based on virtual backbone and software defined network for Internet of vehicles", IEEE International Conference on Communications (ICC), 1-7, 2018

[5] B. Kim, D. Kim, K. Kim and K. Yi, "High-level automated driving on complex urban roads with enhanced environment representation," 2015 15th International Conference on Control, Automation and Systems (ICCAS), Busan, 2015, pp. 516-521.

[6] D. Ferguson, C. Baker, M. Likhachev and J. Dolan, "A reasoning framework for autonomous urban driving," 2008 IEEE Intelligent Vehicles Symposium, Eindhoven, 2008, pp. 775-780.

[7] Vivo, G.: LDM API and Usage Reference. Technical Report D7.3.1 (2010).

[8] Andreone, L., Brignolo, R., Damiani, S., Sommariva, F., Vivo, G., Marco, S.: Safespot final report. Technical Report D8.1.1 (2010).

[9] ETSI EN 302 895 (V1.1.0): Intelligent transport systems - Extension of map database specifications for Local Dynamic Map for applications of Cooperative ITS. Technical report, ETSI (2014)

[10] ETSI TR 102 863 (V1.1.1): Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Local Dynamic Map (LDM); Rationale for and Guidance on Standardization. Technical report, ETSI (2011)

[11] ISO/TS 17931:2013: Intelligent transport systems - Extension of map database specifications for Local Dynamic Map for applications of Cooperative ITS. Technical report, ISO (2013)

[12] ISO/TS 18750:2015: Intelligent transport systems - Cooperative systems - Definition of a global concept for Local Dynamic Maps. Technical report, ISO (2015)

[13] Netten, B., Kester, L., Wedemeijer, H., Passchier, I., Driessen, B.: Dynamap: A dynamic map for road side its stations. In: Proceedings of ITS World Congress 2013 (2013)

[14] Shimada, H., Yamaguchi, A., Takada, H., Sato, K.: Implementation and evaluation of local dynamic map in safety driving systems. J. Transp. Technol. 5(2), 102112 (2015)

[15] Ulbrich, S., Nothdurft, T., Maurer, M., Hecker, P.: Graph-based context representation, environment modeling and information aggregation for automated driving. In: Proceedings of 2014 IEEE Intelligent Vehicles Symposium Proceedings, pp. 541547 (2014)

[16] Zoghby, N.E., Cherfaoui, V., Denoeux, T.: Evidential distributed dynamic map for cooperative perception in vanets. In: Proceedings of 2014 IEEE Intelligent Vehicles Symposium Proceedings, pp. 14211426 (2014)

[17] Y. Xu, V. John, S. Mita, H. Tehrani, K. Ishimaru, and S. Nishino, 3d point cloud map based vehicle localization using stereo camera, in 2017 IEEE Intelligent Vehicles Symposium (IV), 2017, pp. 487492.

[18] Wu, B., Iandola, F., H. Jin, P., Keutzer, K., "SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving". 446-454. 10.1109/CVPRW.2017.60, 2017.

[19] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta and D. Sabella, "On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration," in IEEE Communications Surveys Tutorials, vol. 19, no. 3, pp. 1657-1681, thirdquarter 2017.