Contents lists available at ScienceDirect

Vehicular Communications

journal homepage: www.elsevier.com/locate/vehcom

S-LDM: Server local dynamic map for 5G-based centralized enhanced collective perception

C.M. Risma Carletti [a,*], F. Raviglione [b], C. Casetti [a], F. Stoffella [c], G.M. Yilma [d], F. Visintainer [c]

[a] *Politecnico di Torino, Department of Control and Computer Engineering (DAUIN), C.so Duca degli Abruzzi 24, Turin, 10129, Italy*
[b] *Politecnico di Torino, Department of Electronics and Telecommunications (DET), C.so Duca degli Abruzzi 24, Turin, 10129, Italy*
[c] *SWX/SWE/SAI & ADX, Stellantis - CRF Trento Branch, Via Sommarive, 18, Povo (Trento), 38123, Italy*
[d] *NEC Labs Europe, Kurfürsten-Anlage 36, Heidelberg, 69115, Germany*

A R T I C L E I N F O

Keywords:
Local dynamic map
MEC service
V2X
Vehicular networks
Centralized maneuver management
S-LDM
5G
Field tests

A B S T R A C T

The automotive field is undergoing significant technological advances, which includes making the next generation of autonomous vehicles smarter, greener and safer through vehicular networks, which are often referred to as Vehicle-to-Everything (V2X) communications. Together with V2X, centralized maneuver management services for autonomous vehicles are increasingly gaining importance, as, thanks to their complete view over the road, they can optimally manage even the most complex maneuvers targeting L4 driving and beyond. These services face the challenge of strictly requiring a high reliability and low latency, which are tackled with the deployment at orchestrated Multi-Access Edge Computing (MEC) platforms. In order to properly manage safety-critical maneuvers, these services need to receive a large amount of data from vehicles, even though the useful subset of data is often related to a specific context on the road (e.g., to specific road users or geographical areas). Decoding and post-processing a large amount of raw messages, which are then for the most part filtered, increases the load on safety-critical services, which should instead focus on meeting the deadlines for the actual control and management operations. On this basis, we present an innovative open-source, 5G & MEC enabled service, called Server Local Dynamic Map (S-LDM). The S-LDM is a service that collects information about vehicles and other non-connected road objects using standard-compliant messages. Its primary purpose is to create a centralized dynamic map of the road that can be shared efficiently with other services managing L4 automation, when needed. By doing so, the S-LDM enables these services to widely and precisely understand the current situation of sections of the road, offloading them from the need of quickly processing a large number of messages. After a detailed description of the service architecture, we validate it through extensive laboratory and pilot trials, involving the MEC platforms and production 5G networks of three major European network operations and two Stellantis vehicles equipped with V2X On-Board Units (OBUs). We show how it can efficiently handle high update rates and process each messages in less than few tenths of microseconds. We also provide a complete scalability analysis with details on deployment options, providing insights on where new instances should be created in practical 5G-based V2X scenarios.

1. Introduction

In recent years, the automotive industry has been increasingly focusing on providing high levels of automation to customers, developing more and more enhanced Automated Driving Systems (ADS) towards the deployment of the so-called Society of Automotive Engineers (SAE) Level 4 (L4). This level of automation treats the driver almost as a passenger of the car, being able to manage autonomous driving and perform, in case of necessity, an automatic ADS fallback [1]. A crucial role in enabling fully automated driving is played by the exchange of data between vehicles and centralized services, as well as between vehicles themselves. Different telecommunications standards, including IEEE 802.11p (the automotive version of "Wi-Fi") and 3GPP C-V2X (representing the application of cellular networks to the vehicular field), facilitate said data exchange, commonly known as V2X (Vehicle-to-Everything).

* Corresponding author.
E-mail addresses: carlos.rismacarletti@polito.it (C.M. Risma Carletti), francesco.raviglione@polito.it (F. Raviglione), claudio.casetti@polito.it (C. Casetti), filippo.stoffella@crf.it (F. Stoffella), Girma.Yilma@neclab.eu (G.M. Yilma), filippo.visintainer@crf.it (F. Visintainer).

https://doi.org/10.1016/j.vehcom.2024.100819
Received 23 May 2023; Received in revised form 15 April 2024; Accepted 18 June 2024

Available online 25 June 2024

C-V2X enables, in particular, Ultra Reliable Low-Latency Communications (uRLLC), both between vehicles (through the *PC5* interface) and towards the infrastructure (through the *Uu* interface), thanks to the emergence of 5G.

With the availability of low-latency and high-throughput connectivity, it becomes feasible to create and implement centralized services that can obtain the latest information about the road situation, and provide timely instructions to safely perform highly automated maneuvers, up to SAE L4.

High levels of automation are difficult to attain by solely relying on Vehicle-to-Vehicle (V2V) distributed paradigms, as vehicles are equipped with limited computational capabilities and a restricted perspective of the road (indeed, every pure V2V technology has a limited communication range). It becomes thus crucial to develop and deploy 5G-enabled, real-time, centralized services for highly automated maneuver management, able to build an efficient dynamic map of the road, and provide this data either to the vehicles or to other centralized services, with the aim of centrally coordinating L4 maneuvers, such as centralized automated lane changes. These services, and the way they receive data from vehicles and interact with other infrastructure components thanks to 5G, are still an open research topic, which is currently being addressed by several H2020 European Projects, such as 5G-CARMEN [2].

Furthermore, centralized services for automated maneuver management, up to SAE L4, may require the reception of a large amount of data from vehicles. However, most of the times only a subset of preprocessed data is really required, related to a limited set of vehicles and/or to a limited portion of the road, i.e., a “map” of a section of the road. The provision of data to both centralized services and vehicles should also be performed in a timely manner, guaranteeing high reliability and low latency.

To tackle this challenge, this paper presents a centralized Local Dynamic Map (LDM) service, called Server Local Dynamic Map (S-LDM), developed as part of the 5G-CARMEN project [2], designed as a 5G-enabled Multi-Access Edge Computing (MEC) component. The S-LDM features a custom memory-efficient database containing the latest kinematic information plus path history data of all vehicles and non-connected objects, traveling in a given geographical area. In this way, it can effectively provide other MEC services with a filtered, processed version of the stored data they require for the execution of highly automated maneuvers.

The information about connected vehicles is gathered through the reception of standardized messages such as Cooperative Awareness Messages (CAMs) [3], from where valuable information can be extracted, such as their geographical position, speed, heading angle, steering and exterior lights state. For detected objects instead, information gathered by the vehicle onboard sensors can be sent to the S-LDM encoded as Virtual CAMs. The Virtual CAM, called in this way to distinguish it from the actual connected vehicles CAMs, has been chosen as a preliminary implementation anticipating the target Collective Perception Message (CPM). Instead of sending a list of objects as in a CPM, the absolute position of the detected object is calculated based on sensed data and encoded as a standard CAM. Besides Virtual CAMs, support to standardized CPMs (defined in [4]), is currently being developed and it is expected to be released in the next version of the S-LDM.

Our work focuses not only on the deployment of the S-LDM on the 5G-CARMEN infrastructure, but also on extensive evaluation campaigns. These campaigns involve performance tests in the laboratory as well as measurements carried out on MEC platforms of real network operators, both domestically and cross-border, using V2X-equipped vehicles provided by Stellantis.

Indeed, the S-LDM can be configured to support high scalability and cross-border operations, which are a focus of the 5G-CARMEN project. Being a centralized service, the goal of the S-LDM is to serve an elevated number of vehicles for a richer and more extensive vision of the road. To this end, an extensive scalability study of the service has been carried out for this work.

It should also be highlighted how the management of a broader vision of the road is critical as it allows for more accurate traffic flow prediction, which necessitates the collection and analysis of vast amounts of data from various sources. Hence, the data collected and processed by the S-LDM may enable centralized services to anticipate and mitigate traffic congestion and road incidents more effectively. Finally, the management of traffic flows enabled by centralized services, connected to the S-LDM, can lead to reduced vehicle idling and stop-and-gos, which are in turn related to lower fuel consumption and emissions.

A first overview of the S-LDM has been presented in our conference paper [5]. Nevertheless, the content presented here substantially extends the one in [5] as outlined below:

1. It extensively describes the 5G-CARMEN and C-Roads architecture with details on the Quadkey algorithm used for efficient message filtering;
2. It provides results of an extensive database evaluation, with a comparison showcasing the capabilities of the in-memory database integrated in the S-LDM;
3. It presents an analysis of the featured REST API, evaluating the latency introduced when transferring context data to other MEC services;
4. It provides results of new on-road tests performed cross-border.
5. It proposes new scalability and deployment options with a complete outline of the results obtained from an extensive scalability study;
6. It presents new critical features available within the S-LDM, including a dedicated interface for efficient on-demand data retrieval.

The rest of the paper is organized as follows. Section 2 summarizes the existent related works and projects. Section 3 provides a thorough description of the service architecture, as well as a complete description of each of the S-LDM modules, and the proposed deployment options. Section 4 outlines the results of the extensive pre-deployment evaluations. Section 5 describes the on-road test performed with the service deployed on a commercial MEC infrastructure. Section 6 provides a complete scalability analysis of the S-LDM for the different deployment options. Finally, Section 7 draws some concluding remarks and possible future work and research directions.

2. Related works and other projects

Since the introduction of the concept of the Local Dynamic Map (LDM) studied by the SAFESPOT Project [6], entities such as the European Telecommunications Standards Institute (ETSI) and the International Organization for Standardization (ISO) have pursued its standardization. ETSI defines the LDM in [7], as a facility storing information about vehicles, enabling Intelligent Transport Systems (ITS) applications to retrieve it on demand.

In the vision of ETSI, the LDM, running on an ITS Station (ITS-S) (e.g., a vehicle or a Road-Side Unit – RSU –), relies on information from onboard sensors to interpret the surroundings, combining static and dynamic data. Moreover, an interface is defined for the LDM to leverage other facilities services, such as the Cooperative Awareness Service [3] and the Decentralized Environmental Notification Service [8], enabled to act as LDM data providers, enhancing the view of the road with information from other ITS-S. Indeed, information found on received Cooperative Awareness Messages (CAMs) and Decentralized Environmental Notification Messages (DENMs) from other ITS-S can be stored in the LDM as LDM Data Objects, which can then be accessed by other applications that might be registered as data consumers of such LDM Data Objects, e.g., information of a given vehicle or road event.

Even though onboard perception systems found in modern vehicles provide rich context data, limited perception range and occlusion of objects in Non-Line-Of-Sight (NLOS) bring huge challenges for the correct execution of certain applications, such as Decentralized Intersection Management where a given vehicle may be occluded by another one in front, failing to detect a pedestrian crossing the road. Furthermore, as the market penetration rate of V2X-enabled vehicles increases, the presence of legacy vehicles constitutes an elevated number of highly dynamic objects to be tracked by a given ITS-S. With the aim of tackling the local perception limitations of vehicles, many works have studied the concept of Collective Perception [9–12], on which vehicles share their gathered sensor information through the exchange of Collective Perception Messages (CPMs), currently being standardized by ETSI. Indeed, the exchange of CPMs enables vehicles to expand the vision of their surroundings, including objects beyond sensor range and in NLOS. However, the reception of redundant sensor data from neighboring vehicles comes not only at an elevated channel utilization as addressed in [13] but at an elevated computational burden as well. In [14], an analysis on the effects of storing objects coming from different vehicles in the LDM, at high market penetration rates, shows the impact on the computational load and ultimately on the delay introduced on the LDM update pipeline.

The advent of the MEC paradigm has given rise to the possibility of deploying several services to cover multiple use cases by providing computing power at the edge. Thanks to the availability of computing power near the end-user, several works have explored the implementation of centralized LDM services running on a MEC architecture. In [15], a LDM module is proposed where the information from received CAMs and DENMs is stored in a SQL-based database and processed for the detection of possible collisions in order to notify involved vehicles by issuing DENMs. With a more modular LDM service approach, the solution presented in [16] stores received CAMs with the goal of providing information to an intersection control application. Other works, such as [17], store the received information with a prior change of format to JavaScript Object Notation (JSON) enabling the forwarding of serialized CAMs and DENMs towards other entities using the MQTT protocol. These works exclusively use SQL-based databases to store data, without exploring possible alternate approaches that could enhance the efficiency of data management. The most centralized LDM architectures, like the one presented in [16], are often designed to support specific applications and detect triggering events only relevant to these applications, without considering making the stored information available to other services. On the contrary, the S-LDM supports a more general triggering mechanism for latency-critical MEC services, and includes an on-demand data retrieval interface.

Several different works have studied the inclusion on the LDM of information not only about connected vehicles, but also about detected objects. In [18], the authors present a centralized service aiming to combine the perceptions received from different vehicles, offloading the computational load from the vehicles themselves. The proposed solution considers the exchange of raw sensor data instead of CPMs, as in our case, which compromises the scalability of such a service. Raw sensor data is indeed much larger than the post-processed data encoded in CPMs. On the other hand, the work presented in [19] considers the storage of detected objects information from received CPMs, working with a Neo4j database instead of a SQL-based one for a graph-based approach. Although this work proposes a scalability study, the considered rate of V2X messages is 10 Hz instead of 20 Hz, as in our case. Specifically, we consider 20 Hz in accordance with the update rate of modern, high-rate, vehicle on-board sensors for Adaptive Cruise Control (ACC) [20]. This requirement is aligned with pilot vehicles used in 5G-CARMEN which have been obtained enhancing a production highway assist system (ACC plus lane centering). Such frequency has been chosen for our project with the aim of studying safety-critical automated maneuvers executed in challenging conditions. Finally, of all works presented here, ours is the only one showcasing results of on-road tests with the service deployed in a commercial MEC infrastructure.

3. Service architecture and deployment options

The idea behind the proposed Server Local Dynamic Map can be summarized as follows.

5G-enabled MEC services are gaining growing interest and importance in the automotive and autonomous vehicle field. These services, in order to properly enable innovative use cases (such as collision avoidance, centralized platooning, Vehicular Edge Cloud [21], and many others), require the reception of a large amount of data from vehicles and other road users. Despite receiving a significant amount of information through messages such as CAMs, really useful data often include only a subset of post-processed information related to a specific context on the road. This context can correspond, for instance, only to data of vehicles and other non-connected road users involved in a highly automated maneuver, or only to one or more given geographical areas.

MEC services may be thus overloaded during the reception and information filtering phases, especially under a high number of subscribers and/or connected vehicles, reducing the amount of resources available to perform the actual safety-critical tasks. These tasks may include Vehicular Edge Cloud [21] or centralized platooning coordination [22].

There is thus a strong necessity for the deployment of a “middleware” service able to receive data from a large number of vehicles, process it in an optimal way to store an efficient map of the road, and provide a post-processed and filtered version of such data to other MEC services (e.g., centralized automated maneuver management services) when needed.

To fill this gap, we propose the S-LDM. Our service offers a broad set of capabilities, much wider than what ETSI foresees for its LDM [7]. Furthermore, it focuses on a centralized approach, with the following features:

- Storage of decoded and pre-processed data of messages from a large number of vehicles in a custom efficient database for fast data storage.
- Enhanced view of the road, extending the coverage that would be achievable with decentralized solutions.
- Maintenance of historical data, in addition to real-time data, about both vehicles and detected objects.
- Option to store road events (for instance through the reception of event-based DENMs).
- On-demand availability of a filtered and pre-processed version of the stored information.
- Detection of traffic conditions for triggering the transfer of relevant context data to other safety-critical MEC services.
- Compatibility with MEC and container orchestration systems, such as Kubernetes.
- Graphical User Interface for human monitoring of road traffic.
- Support to the C-Roads architecture for advance message dissemination and delivery, as described in Section 3.1.
- Multi-threaded Advanced Message Queuing Protocol (AMQP) client for message reception with support for multiple broker subscriptions, enabling cross-border use cases.

These outlined features enable offloading both vehicles and other services from the necessity of quickly processing a large number of messages, which could reduce the performance of actual latency-critical algorithms (e.g., control algorithms for centralized automated lane merge).

Finally, a full C++ implementation of the S-LDM has been released under an open-source GPLv2 license, and it is available on GitHub.[1]

The next sub-Sections detail first the architecture for ITS foreseen by the 5G-CARMEN and C-Roads projects, as part of which the S-LDM

[1] https://github.com/francescoraves483/S-LDM.

has been developed (Section 3.1), then the architecture of our service (Section 3.2) and finally how it is designed to be deployed in real-world scenarios (Section 3.3).

3.1. 5G-CARMEN and C-Roads architecture

The S-LDM has been developed as part of the 5G-CARMEN project architecture, which is based in turn on the architecture standardized in the C-Roads project [23]. The architecture foreseen by C-Roads investigates the usage of messaging protocols for the efficient transmission of messages from/to centralized MEC services, thanks to V2I and V2N communications for Cooperative ITS (C-ITS). The output of the project is a standard architecture for the transmission of ETSI C-ITS messages over the Advanced Message Queuing Protocol (AMQP) version 1.0, a platform-independent messaging protocol, which can be used both for peer-to-peer communication, or for communication with an intermediate centralized entity. This entity, called *message broker*, is in charge of receiving messages from vehicles and it acts as message collector and dispatcher. Messages are collected in broker *queues* (or *topics*, referring to the definition given by Apache [24]), which are addressed through their name (which can be hierarchical).

Vehicles and other road users can send their messages to the broker (such as CAMs and CPMs), and they subscribe to specific queues or topics for the reception of warning, road signage and maneuver management messages.

Conversely, MEC services can subscribe to the broker to receive messages from a set of vehicles of interest, based on the queue or topic they subscribe to, or on specific message properties, which can be embedded as additional information in the header of each AMQP message. These properties, called *application properties* correspond to an arbitrary number of key-value couples, in which the key is a string defining the property name, and the value can be any simple type (mainly string or number). Thanks to these application properties, additional information, normally not available in standard ETSI C-ITS messages, can be transmitted to the broker and to centralized MEC services (e.g., country and string ID of the originating vehicle).

According to C-Roads, the properties should also include the *Quadkey* of the location the vehicle is traveling in. In short, Quadkeys are a way to refer to squared geographical areas through hierarchical base-4 strings and were originally defined by Microsoft[2] [25]. More details are provided below.

Properties can be used by subscribing services to efficiently filter the messages at an AMQP broker level. Indeed, other than selecting a proper topic, a client subscribed to the broker can choose to receive only the messages with certain property values (e.g., only messages with `originatingCountry="DE"`). The broker will then forward to the MEC service only the messages matching an SQL-like filter on the properties provided by the client itself. This filter can be arbitrarily complex and include both logical operators and wildcards. It is worth mentioning that all the operations performed by the AMQP broker, such as message filtering, are executed transparently at an AMQP level and they are independent from the type of payload carried (i.e., independent from the Facilities Layer message type).

Furthermore, MEC services can also send messages such as DENMs and IVIMs to the broker for dissemination to relevant road users (or to all users traveling in a given geographical area) subscribed to the same broker. Communication between different MEC services through the broker is also possible.

As foreseen by C-Roads, the S-LDM receives ETSI-compliant messages through one or more AMQP 1.0 brokers [26]. The ETSI C-ITS messages, including the Facilities [3], GeoNetworking [27] and Basic Transport Protocol [28] layers, are further encapsulated into AMQP 1.0 for the transmission to the S-LDM according to the C-Roads specifications. As soon as CPMs are fully standardized, their reception and management support is also planned.

Being platform-independent, one of the advantages of AMQP is its independence on the lower layers of the stack. Indeed, the S-LDM can receive messages encapsulated into AMQP and transmitted to the broker either via IEEE 802.11p/IEEE 802.11bd, or via C-V2X and 5G, to guarantee high reliability. In addition, the approach foreseen by C-Roads and adopted by the S-LDM easily provides cross-OEM compatibility, as the vehicle OBU manufactures just need to include a client compliant with AMQP 1.0 [29] and the C-Roads specifications [30], which are both publicly available.

Furthermore, 5G-CARMEN extends the C-Roads architecture by foreseeing an underlying 5G network, as a fundamental enabler for latency-critical automotive use cases, such as decentralized maneuvers. For instance, the 5G-CARMEN project addressed the use case of decentralized cooperative automated maneuvers, which was enabled by an extended perception service (collective perception through sensor sharing) [31]. Indeed, 5G is able to guarantee the reliability and low latency required by components such as the S-LDM or other maneuver management services, which is not always guaranteed by 4G [32].

Fig. 2 shows how the S-LDM is integrated with the overall 5G-CARMEN architecture for C-ITS, considering a single broker scenario. As can be seen, the AMQP broker of choice is Apache ActiveMQ, being a reliable open source software.

With the aim of enabling the reception of messages via AMQP, the S-LDM includes an open-source library for AMQP message reception (i.e., Apache Qpid Proton) and a custom ETSI ITS decoding stack, developed to be as efficient as possible with the aim of enabling very high update rates of the internal dynamic map.

The S-LDM has been designed to be flexibly deployed in a wide range of MEC platforms, both Virtual Machine-based and container-based. With the aim of testing it in the field in real V2X scenarios, it has been integrated as a container inside the MEC platforms of three network operators participating in the 5G-CARMEN project (i.e., TIM of Italy, Magenta Telekom of Austria and Deutsche Telekom AG of Germany), as shown in Fig. 2. Each MEC platform runs a local AMQP broker to reduce the latency experienced by the vehicle-to-AMQP-broker communication, and in turn executes a Kubernetes-based orchestrated edge platform, developed by 5G-CARMEN [33], and designed to run and orchestrate 5G-enabled applications and services for Cooperative and Automated Driving (CAD).

The orchestrated mobile edge platform is designed and developed by following the cloud-native principles and is inline to the standardization frameworks provided by ETSI MEC, ETSI Network Function Virtualization (NFV), and 3GPP [33]. This design enables collaboration between 5G edges, thereby extending the range of the services/applications running on top of these edges, and allowing them to collaborate with peering service/application instances in different domains, to enable service continuity. The orchestrated edge platform includes several key design features, including coupling of 5G and MEC/NFV, end-to-end mobile data plane control, and application-specific support for orchestration operations.

It should be mentioned that the S-LDM has been tested in the field as part of this container-based scenario to investigate its usage within a challenging and real-world V2X 5G-enabled architecture. However, it is flexible enough to be integrated into virtually any Linux-based MEC platform, requiring only the subscription to one or more AMQP 1.0 brokers for the reception of messages from vehicles.

3.1.1. Quadkeys for vehicle localization

As mentioned earlier, each ETSI C-ITS message sent to a broker following C-Roads specifications should include the Quadkey property. The latter is also of particular relevance for the S-LDM, which can efficiently filter messages coming only from vehicles traveling in a given area, defined by a set of Quadkeys. Even though the S-LDM can prop-

[2] https://docs.microsoft.com/en-us/bingmaps/articles/bing-maps-tile-system.

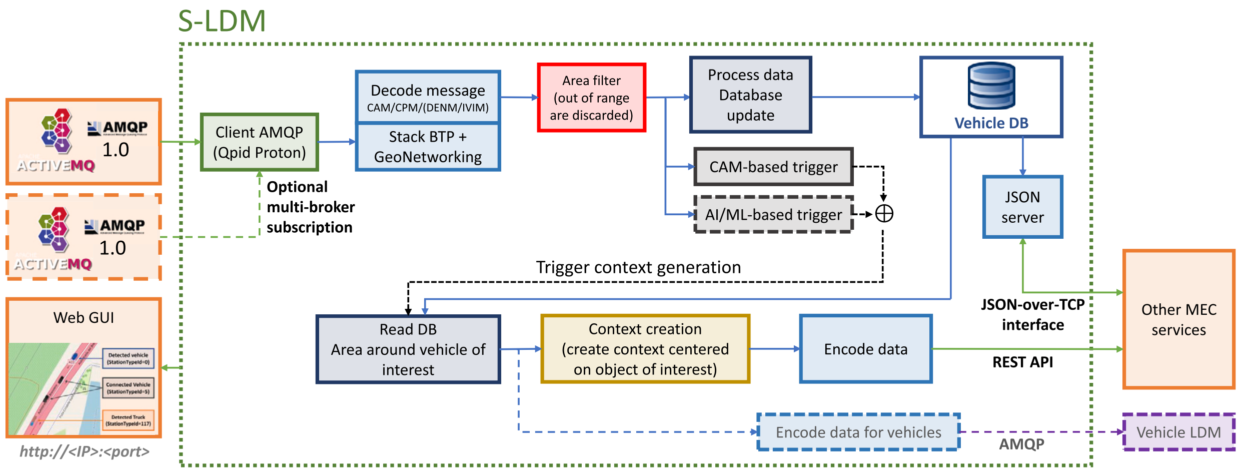

S-LDM
AMQP 1.0
ACTIVEMQ
Client AMQP (Qpid Proton)
Decode message CAM/CPM/(DENM/IVIM)
Stack BTP + GeoNetworking
Area filter (out of range are discarded)
Process data Database update
Vehicle DB
CAM-based trigger
AI/ML-based trigger
JSON server
Optional multi-broker subscription
Trigger context generation
Web GUI
http://<IP>:<port>
Read DB Area around vehicle of interest
Context creation (create context centered on object of interest)
Encode data
JSON-over-TCP interface
REST API
Other MEC services
Encode data for vehicles
AMQP
Vehicle LDM

Fig. 1. S-LDM service internal architecture.

erly process messages without this property, its use can improve the overall performance of the service when filtering messages of vehicles traveling outside its coverage area, as detailed in Section 3.3. Therefore, the main aim of this Section is to briefly present the Quadkeys to the reader, as defined by Microsoft in [25].

Quadkeys are a way to identify rectangular areas on a map, and can be used to represent the geographical position of vehicles as compact strings with base 4 digits. These strings are then transmitted as message properties and can be used for geographical message filtering.

The concept of Quadkey is based on a bi-dimensional Mercator projection of the Earth. Specifically, the projected map can be divided into rectangular tiles, corresponding to rectangular areas. Each of these tiles can then be uniquely identified by a Quadkey, with the following logic: the map is divided into four parts, which are assigned, clockwise from top-left, four digits (0, 1, 2 and 3). These digits represent the level 1 Quadkeys, corresponding to very large areas. Each rectangular tile can be further divided in four parts, defining the level 2 Quadkeys. Each of these parts will keep as first digit the one of the parent tile (i.e., the tile from which it originates) and add a second digit again from top-left to bottom-right. The process can be iteratively repeated defining progressively higher level Quadkeys. For instance, the area defined by Quadkey `2` contains the one defined by `21`, which contains the one defined by `212`, and so on.

Quadkeys are thus hierarchical strings which define progressively smaller areas as the level increases, as depicted in Fig. 3.

Microsoft also defines a convenient algorithm for translating a latitude and longitude value into a Quadkey [25]. Given as input the level of detail (L) and the coordinates of a point, the algorithm returns the level L Quadkey of the area which contains that point.

This algorithm is composed by few simple steps, which can be efficiently executed by the OBU of a vehicle to compute, given L, the Quadkey of the area comprising the current location. The size of this area (directly linked to how precise the localization of the vehicle is) is defined by L. This value can then be transmitted via AMQP 1.0 as a message property.

Quadkeys are crucial for efficient message filtering inside the S-LDM [32].

Indeed, each S-LDM instance can subscribe to receive all messages from a certain large area by generating a filter on the `quadkeys` property with one or more low level Quadkeys (with wildcards) corresponding to the desired area. For instance, if the filter contains the string "321%" (% is just a wildcard meaning "zero or any number of other digits"), the service will receive all and only the messages from vehicles sending (higher level) Quadkeys such as "**321**1210012" and "**321**0012111" (i.e., all messages from vehicles traveling inside the area corresponding to the level 3 Quadkey "321"). Vehicles can thus send a high level, more precise, Quadkey, and the S-LDM can easily receive and filter all messages from a large area by specifying a low-level Quadkey. This approach is particularly useful as all the filtering operations occur in the AMQP broker, after the S-LDM generates and sends the proper filter to the broker. In presence of a large number of vehicles, the S-LDM can thus focus on decoding only the messages coming from the area of interest, without the burden of decoding each single message and then filtering afterwards based on the information received in the Facilities or GeoNetworking layers [27].

3.2. S-LDM description

The S-LDM has been designed with a modular architecture, and it is composed of several sub-modules interacting with each other, from the reception of a message through the AMQP client, to the generation of the context data for other MEC services.

The internal architecture of the S-LDM and its sub-modules is schematized in Fig. 1, and described in the following sub-sections.

Before delving into the details of our service, we present a brief walk-through of the S-LDM operations from the reception of a message via AMQP 1.0 to the provision of data to other MEC services, with reference to Fig. 1. First, messages can be transmitted to the S-LDM via one or more AMQP brokers, compliant with the latest 1.0 version. If the C-Roads standard is implemented, Quadkeys can be used to pre-filter the messages, such that only messages of vehicles traveling in the S-LDM coverage area will be received by the S-LDM AMQP client (Section 3.2.1).

Messages are then decoded based on their type and thanks to a custom, efficient ETSI-compliant message decoder. Supported and planned message types include CAMs, virtual CAMs, CPMs, DENMs and IVIMs. Furthermore, the S-LDM supports messages both with and without the BTP and GeoNetworking layers (Section 3.2.2).

After being decoded, messages are passed through an efficient area filter module, that precisely filters only the messages of road users included in the current S-LDM instance coverage area (Section 3.2.3).

The filtered messages are then processed and used to store the kinematic and dynamic information of vehicles and other road users into a highly-efficient custom database (Section 3.2.4), that has been proved to be more efficient than other commonly-used databases in state-of-the-art solutions. This database represents the core component of the S-LDM, and it can provide information to other MEC services or directly to vehicles in three different ways:

- Section 3.2.5: a special condition on the road is detected by the S-LDM that will read the content of the database, extracting the data

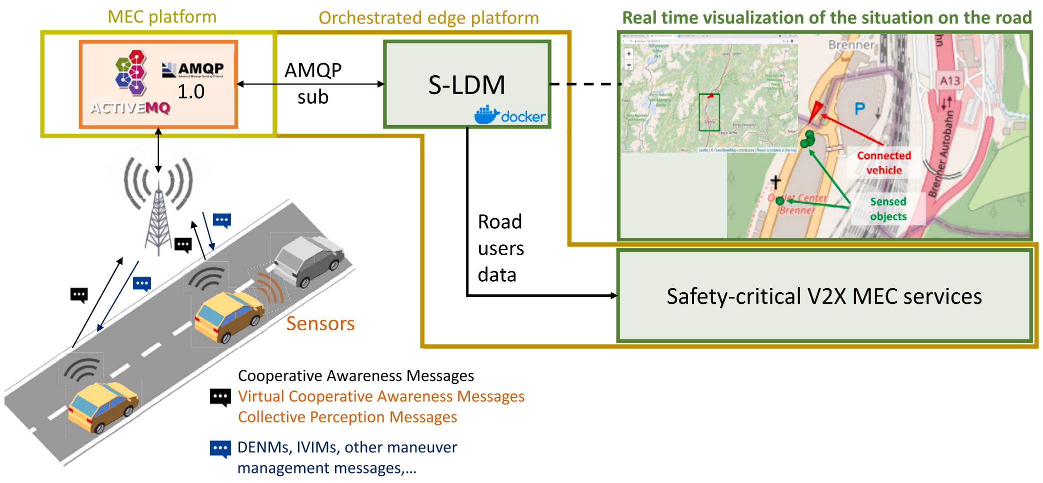

MEC platform
Orchestrated edge platform
Real time visualization of the situation on the road
AMQP 1.0
ACTIVEMQ
AMQP sub
S-LDM
docker
Connected vehicle
Sensed objects
Road users data
Safety-critical V2X MEC services
Sensors
Cooperative Awareness Messages
Virtual Cooperative Awareness Messages
Collective Perception Messages
DENMs, IVIMs, other maneuver management messages,...

Fig. 2. The S-LDM as part of the 5G-CARMEN architecture, in turn based on C-Roads standards.

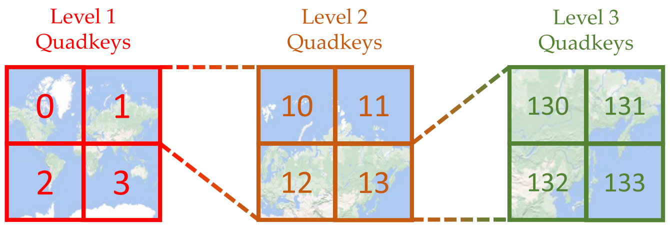


Fig. 3. Logic for the definition of Quadkeys on a Mercator projection of the world, as defined by Microsoft [25].

for the involved vehicles and transmitting it to other maneuver management services, thanks to a dedicated REST API;

- Section 3.2.6: MEC services query the S-LDM, requesting the most recent information related to specific vehicles in a given geographical area thanks to a simple, yet efficient, JSON-over-TCP interface;
- Section 3.2.8: depending on the application, the S-LDM may also provide a filtered and summarized version of the database content to the vehicle themselves, to update their local LDMs with precise centralized information related to objects on the road; this is enabled thanks to dedicated messages transmitted to subscribed vehicles via AMQP.

Finally, the content of the S-LDM can be leveraged to perform real-time monitoring by one or more road operators, thanks to a web-based GUI (Section 3.2.7).

3.2.1. *AMQP client and quadkey-based filtering*

The S-LDM can receive messages from vehicles thanks to a multi-threaded AMQP client. In case of a single connection to a single AMQP broker, a single thread is used for the reception of messages. However, each S-LDM also supports, optionally, subscribing to more than one broker. In this case, a new thread is created for each broker, to independently manage each AMQP connection in parallel.

The AMQP client within the S-LDM has been developed by integrating the Apache Qpid Proton library, which can quickly, efficiently handle the reception of messages from a broker compliant with AMQP 1.0.

When an S-LDM instance is created, the most important parameter is represented by its coverage area. Each S-LDM instance is indeed designed to cover a configurable geographical area, whose size can be configured depending on the final deployment and target use case managed by the MEC services connected to the S-LDM.

The S-LDM thus filters out all messages coming from vehicles outside its coverage area. As mentioned earlier, the S-LDM employs, if possible, pre-filtering with Quadkeys, with the aim of improving the efficiency of the filtering mechanism, and avoid wasting CPU resources in decoding each single ETSI message to detect whether the original vehicle is inside the coverage area.

Currently, rectangular coverage areas are supported, as they match well the shape of areas that can be defined with a set of Quadkeys. It is planned, however, to add support for circular and elliptic areas in future versions of the S-LDM.

With the aim of setting up a proper filter, to be later sent to the AMQP broker for filtering, the S-LDM takes as input the minimum and maximum latitude and longitude corresponding to the vertices of a rectangular coverage area. These values are then converted to the minimal list of Quadkeys covering the smallest area greater or equal than the input one, given a maximum level L_{filter}.

In order to generate a filter of the smallest possible size, the minimal list of Quadkeys is computed (e.g., instead of sending "`quadkeys LIKE '320% OR 321% OR 322% OR 323% OR 311%'`", the S-LDM will generate the smaller-size equivalent filter "`quadkeys LIKE '32% OR 311%'`", thanks to the hierarchical properties of Quadkeys). The usefulness of this approach is twofold. First, it makes the initial connection with the AMQP broker faster. Second, it makes it feasible to send filters corresponding to large areas, as there is a size limit (depending on the actual AMQP settings) in the SQL-like filter string that can be transmitted to the broker for filtering. Indeed, as the S-LDM default value for L_{filter} is 16, large areas may correspond to a very large number of level-16 Quadkeys, resulting in a very long filter. This filter can always be, however, rewritten to an equivalent form by "merging" higher-level Quadkeys making up a lower-level Quadkey. As an example, the set of level-3 Quadkeys "320", "321", "322" and "323" can always be referred to just by the level-2 Quadkey "32", thus significantly reducing the length of the corresponding SQL-like filter.

Fig. 4 shows an example of input and output from the algorithm generating the set of Quadkeys for message filtering, with $L_{filter} = 11$.

The algorithm pseudocode is reported in *Algorithm 1*, where the input values are (i) the minimum and maximum latitude of the input area Lat_{min} and Lat_{max}, (ii) the minimum and maximum longitude of the input area Lon_{min} and Lon_{max} and (iii) L_{filter}. Δ_{LatLon} (line 7) is an additive factor to be progressively summed up to the Lat_{min} and Lon_{min} values, to compute the corresponding Quadkeys covering the whole area up to Lat_{max} and Lon_{max}. Since, to avoid generating duplicated Quadkeys, it depends on the level of detail L_{filter}, it is computed through a lookup table, defined here as $\tau(L_{filter})$. $\mu(Lat, Lon, L)$ is in-

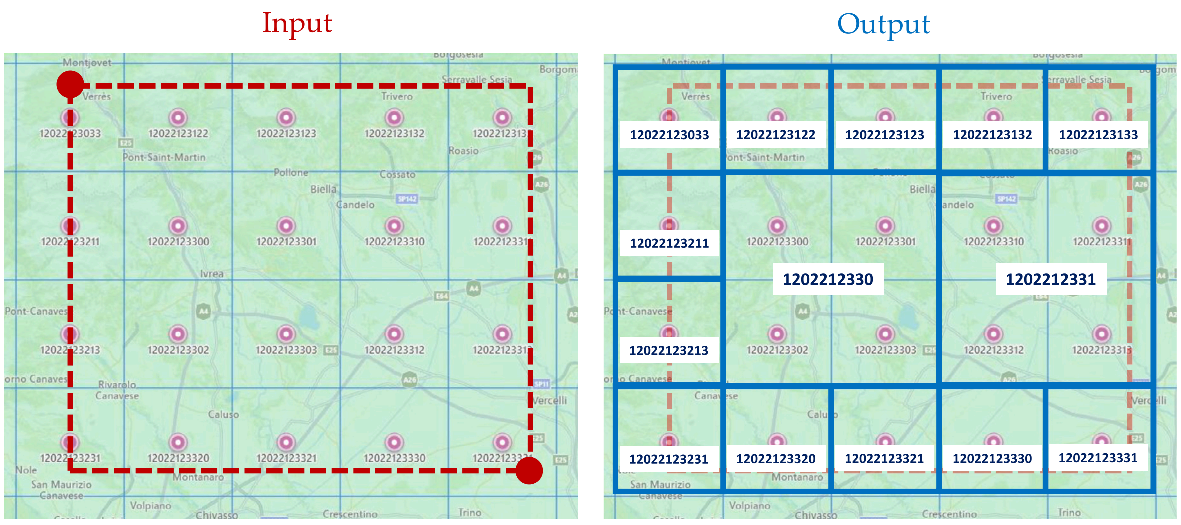


Fig. 4. Example of input and output to the Quadkeys filter generation algorithm, as part of the S-LDM, with $L_{filter} = 11$. As input, the latitude and longitude of the two points corresponding to red circles are given. As output, the minimum set of Quadkeys fully covering the specified area is provided. The set, in this case, corresponds to 12 level-11 Quadkeys and 2 level-10 Quadkeys.

stead the Microsoft algorithm to obtain the L_{filter} level Quadkey a point is located in, given its latitude and longitude [25].

The algorithm defines as LV (line 17) a counter which starts from L_{filtr} and is decreased each time, to consider for each iteration progressively lower-level Quadkeys to be merged together, if possible. Notice how the for loop inside the while loop (line 20) merges all LV level Quadkeys into $LV-1$ "low level" level Quadkeys if possible, replacing each four LV level Quadkeys with the corresponding $LV-1$ Quadkey inside $Set_{Quadkeys}$ (lines 26-27). This is done iteratively, considering one level of detail at a time, starting from the highest level (i.e., $LV = L_{filter}$, on line 17) until either no more merges are possible (if clause on line 31), or when the lowest possible level (i.e., level 1) is reached (while loop condition on line 18).

The final sorting step (line 36) places first the shorter Quadkeys, corresponding to larger areas, in which it should be more probable to have a greater number of vehicles (i.e., more messages are probably coming from these areas). Since the AMQP 1.0 filter is an OR between all the Quadkeys, having first the Quadkeys with a possibly greater number of vehicles may slightly improve the broker-side filtering performance (as the match may be found earlier when the broker is "scanning" all the Quadkeys in the filter).

After the proper set of Quadkeys has been generated, it is converted to an AMQP filter and transmitted to the AMQP broker during the S-LDM instance start-up phase. The latter also includes setting up the connection to all the brokers, with all the needed AMQP parameters (e.g., username, password, authentication mechanism, idle timeout).

As computing the proper minimal set of Quadkeys can be computationally expensive for large areas, after the first time the result is cached into a file, to significantly speed up the process if the same S-LDM instance is restarted with the same coverage area parameters. If the area parameters change, the cache file is updated accordingly.

Thanks to Quadkey pre-filtering, each AMQP client of each S-LDM instance can be set to receive only the messages of vehicles traveling in an area slightly larger than the coverage area (e.g., the blue area defined by the Quadkeys in Fig. 4, which fully covers the input red area), limiting the number of messages discarded at the Area Filter module and improving the performance of the S-LDM itself.

3.2.2. Message decoder

After being successfully received by the AMQP Client each message is decoded thanks to a custom ETSI C-ITS stack, developed to be as much efficient as possible with the goal of enabling very high update rates of the internal database. This stack currently supports reception of CAMs and DENMs (even though the S-LDM focuses on CAMs, the latter can be received from road operators to further enrich the internal map with road traffic and hazardous event information). Support for CPMs and IVIMs is currently planned for future versions of the service, to enable further enhancing the database of the S-LDM. This additional information may include, for instance, position of road signs received thanks to IVIMs [34].

The message decoder can efficiently receive and decode ETSI C-ITS messages comprising the GeoNetworking [27], Basic Transport Protocol (BTP) [28] and Facilities layers [3,8], further encapsulated inside AMQP 1.0, as foreseen by C-Roads. However, as messages are transmitted via AMQP 1.0, which does not strictly require the presence of additional Transport and Network layers, the S-LDM message decoder also supports reception of standardized messages without the GeoNetworking and BTP layers (e.g., "pure" CAMs). The detection whether a message includes the ETSI Network and Transport layers is performed transparently and automatically on a per-message basis. This enables different implementations (i.e., sending both "pure" ITS messages and "full" ITS messages including BTP and GeoNetworking, as recommended by C-Roads) to be supported out-of-the-box by the S-LDM. As mandated by the ETSI standards, the Facilities Layer messages are encoded with Abstract Syntax Notation One (ASN.1), used by many communications protocols, thanks to its cross-platform nature. An open-source tool called asn1c [35] has been leveraged to generate the files needed to decode the messages in ASN.1 format into an equivalent C/C++ structure. Once the Facilities Layer message is decoded into its corresponding C/C++ structure, all the fields of interest (from the ones specified in [3, Annex A] for CAMs and [8, Annex A] for DENMs) can be extracted to be saved in the main database, as further detailed in 3.2.4.

As mentioned previously, the S-LDM has the capability to gather information from both connected and non-connected vehicles, as well as other road users like pedestrians. These non-connected entities, despite lacking direct communication capabilities, are still relevant to autonomous vehicles as they may occupy the roadway or its immediate surroundings. Specifically, the S-LDM has been designed to receive CAMs and perceived objects information from a multitude of connected vehicles. It is worth noting that the S-LDM supports the reception of CAMs from each vehicle at a frequency of up to 20 Hz, surpassing the maximum frequency limit of 10 Hz set by ETSI, to facilitate precise and detailed updates for highly automated maneuver management. It should also be noted that the 5G-CARMEN project is experimenting with a transmission rate of 20 Hz over a mobile network. This high

Algorithm 1 Quadkey filter generation algorithm for the S-LDM.

```
1: Parse the input parameters
2: Read cache file is it exists
3: if Parameters ($Lat_{min}$, $Lat_{max}$, $Lon_{min}$, $Lon_{max}$, $L_{filter}$) are the same as stored in the cache file then
4:     Read Quadkey filter from cache file
5:     Terminate execution
6: end if
7: $\Delta_{LatLon} \leftarrow \tau(L_{filter})$
8: Define empty set of $L_{filter}$ level Quadkeys $Set_{Quadkeys}$
9: for $Lat_{current} = Lat_{min}$; $Lat_{current} \leq Lat_{max}$; $Lat_{current} += \Delta_{LatLon}$ do
10:     for $Lon_{current} = Lon_{min}$; $Lon_{current} \leq Lon_{max}$; $Lon_{current} += \Delta_{LatLon}$ do
11:         $Curr_{quadk} \leftarrow \mu(Lat_{current}, Lon_{current}, L_{filter})$
12:         Add $Curr_{quadk}$ to $Set_{Quadkeys}$
13:     end for
14: end for
15: Remove possible duplicated Quadkeys from $Set_{Quadkeys}$
16: Sort $Set_{Quadkeys}$ in increasing order (e.g., "321" goes after "320" and before "322") ▷ This steps helps looking for mergeable Quadkeys as they will be located in the set one after the other
17: $LV \leftarrow L_{filter}$
18: while $LV > 0$ do
19:     $merge_{occurred} \leftarrow False$ ▷ This flag remains set to $False$ is no Quadkeys could be merged at the current level and we should terminate the loop
20:     for $Index_{quadk} = 0$; $Index_{quadk} < Length(Set_{Quadkeys})$; $Index_{quadk} \leftarrow Index_{quadk} + 1$ do
21:         $LowLevel_{quadk}[0] \leftarrow$ first $LV - 1$ digits of $Curr_{quadk}[Index_{quadk}]$
22:         $LowLevel_{quadk}[1] \leftarrow$ first $LV - 1$ digits of $Curr_{quadk}[Index_{quadk} + 1]$
23:         $LowLevel_{quadk}[2] \leftarrow$ first $LV - 1$ digits of $Curr_{quadk}[Index_{quadk} + 2]$
24:         $LowLevel_{quadk}[3] \leftarrow$ first $LV - 1$ digits of $Curr_{quadk}[Index_{quadk} + 3]$
25:         if $LowLevelquadk[i]$ are all the same for $i = 0, .., 3$ and $Last\_Digit(Curr_{quadk}[Index_{quadk}]) = 0$ and $Last\_Digit(Curr_{quadk}[Index_{quadk} + 1]) = 1$ and $Last\_Digit(Curr_{quadk}[Index_{quadk} + 2]) = 2$ and $Last\_Digit(Curr_{quadk}[Index_{quadk} + 3]) = 3$ then
26:             Remove $Curr_{quadk}[Index_{quadk} + i]$ from $Set_{Quadkeys}$ for $i = 0, .., 3$
27:             Add $LowLevelquadk[0]$ to $Set_{Quadkeys}$
28:             $merge_{occurred} \leftarrow True$ ▷ Set to true if any Quadkeys have been merged at the current $LV$
29:         end if
30:     end for
31:     if $merge_{occurred} = False$ then
32:         Break
33:     end if
34:     $LV \leftarrow LV - 1$
35: end while
36: Sort $Set_{Quadkeys}$ by Quadkey length
37: Generate SQL-like filter with all Quadkeys inside $Set_{Quadkeys}$
38: Cache parameters ($Lat_{min}$, $Lat_{max}$, $Lon_{min}$, $Lon_{max}$, $L_{filter}$) and $Set_{Quadkeys}$ into cache file
```

transmission frequency value allows for fine-grained information to be conveyed. The decision to adopt this value is based on the typical update rate of high-resolution on-board sensors for automotive [36] such as the ones used in the vehicle prototypes. Sharing information between vehicles at this rate allows the vehicle to extend its field of view utilizing the perception of the vehicle in front as an extension of its own sensors (in this case up to 20 detection updates per second), preventing bottlenecks possibly given by the communication link.

In order to guarantee a timely, low-latency delivery of high-rate messages, it becomes evident how the S-LDM should be part of a proper 5G network, such as the one deployed and tested as part of 5G-CARMEN [32].

Concerning instead the non-connected objects, the S-LDM is expected to make use of CPMs, as soon as they are fully standardized. However, since the standardization process is still in progress at the time of writing, we developed the so-called *Virtual CAMs*, carrying information about detected objects. These messages are sent by connected vehicles that can detect these objects through their on-board sensors. These special messages have been chosen as a preliminary implementation before CPMs are standardized. Indeed, remote objects can be detected by vehicle sensors, and this information can be provided to the V2X OBU through the vehicle CAN bus. The OBU can then encode this data in Virtual CAMs used for tracking remote objects, as if the non-connected object were connected and sending CAMs. Indeed, instead of sending a list of objects as would be required by CPMs, the absolute position of the detected objects is calculated based on sensed data and encoded as a standard CAM. These fabricated Virtual CAMs are sent to the AMQP broker(s), to which the S-LDM is subscribed, by connected vehicles. Regarding the non-connected objects, the S-LDM is expected to make use of CPMs, once their standardization is finalized. However, since the standardization process was still ongoing at the time of writing, we developed a solution called Virtual CAMs to convey the information about detected objects. These messages are sent by connected vehicles that can detect remote objects through their on-board sensors. The latter estimate the relative position of detected objects and classify their types, such as vehicles, trucks and pedestrians. The object data are then communicated to the V2X OBU via CAN bus. Thanks to a precise positioning reference of the ego vehicle, the OBU can calculate the absolute position of the detected objects and encode this data along with object attributes into frames conforming to the standard CAM format, creating a Virtual CAM. These fabricated Virtual CAMs are sent as normal CAMs to the AMQP broker(s), to which the S-LDM is subscribed, by connected vehicles. The result is that the S-LDM perceives also the non-connected objects as if they were transmitting their own CAMs to the cloud service. As said before, these dedicated messages were implemented as an interim solution prior to the standardization of CPMs. The approach here is different from the one of the CPMs, where objects are listed in a single message. Indeed, using Virtual CAMs, each object has its own message. At the receiving unit (S-LDM and other connected vehicles), once decoded, the V2X objects are managed in the same way, independently of the type of C-ITS message.

3.2.3. Area filter

Each message, after being decoded, is provided as input to the Area Filter module, which checks the originating vehicle position (gathered either from the GeoNetworking layer, if available, or from the CAM message itself) and discards messages from road users located outside the coverage area. Despite the Quadkey pre-filtering mechanism, the presence of this module is required for two main reasons:

- The S-LDM performance can greatly benefit from the pre-filtering mechanism, but the service is designed to be as much flexible as possible, and it can thus support the reception of messages without the Quadkey property. In this case, all filtering should occur in the Area Filter module.
- Given any input area and L_{filter} value (i.e., 16, considering the default value in the S-LDM), the corresponding Quadkeys are likely to cover an area slightly larger than the input one, as shown in Fig. 4 by the difference between the red and blue areas. A very small portion of messages will thus be received anyway even if they do not belong to the desired input coverage area.

3.2.4. Main database

The received information is processed (extracting the relevant data from each standardized message) and used to update a custom, highly-efficient, in-memory, thread-safe database, storing the content of the centralized local dynamic map. It thus represents the core component of the S-LDM service, and should be able to support fast insertion and lookup operations.

The choice of an in-memory database comes from its higher performance when compared to a database writing to the disk. The latter would provide the advantage of being able to recover previously stored data in case of a service failure or forced restart, at the price of an increased INSERT execution time. However, recovering previously saved data would provide little to no advantage, as objects in V2X scenarios are usually characterized by a high mobility, and, by the time the service is restarted, they would have likely changed their kinematic and dynamic status quite significantly.

The choice for a custom database, instead of proposing the usage of already existing solutions, comes from an extensive evaluation covering four main database solutions:

- SQLite saved in main memory
- SQLite saved on disk with a Write-Ahead Log (WAL) mechanism
- WhiteDB (NoSQL database on shared memory) [37]
- Custom thread-safe database in main memory, described more in details below

Our custom solution leverages multiple hash tables saved in main memory, using sections of the vehicle station IDs as keys, and it has been implemented in C++. The code of the custom database solution is available as part of the S-LDM GitHub repository.[3]

More specifically, the database is composed by a main *upper* hash table and several *lower* hash tables, implemented as efficient *unordered maps*. The main hash table uses as key the first half of the station ID, and each entry is represented by a pointer to a *lower* data structure. Each lower data structure, in turn, uses as key the second half of the station ID, and each entry represents the actual data stored in the S-LDM for that specific road user.

The main advantage of using a hierarchical structure is that this enables the protection against concurrent reads and writes only of the *lower* hash tables. This allows the S-LDM to write read and write to the database at the same time, provided that the data that is being retrieved is not located in the same *lower* hash table that is currently being updated. Especially when managing a large number of nodes, this can provide significant performance gains, since parallel read and write operations are expected to occur relatively often. Indeed, protecting the whole database would result in the S-LDM being prevented from reading the database every time a new message is received and the map should be updated.

After a message has been decoded, the following main information is stored in the database, taking as a reference a connected or detected vehicle:

- the full road user station ID;
- the up-to-date position in geodetic coordinates;
- the up-to-date position as a Quadkey, as included by the sender, if available;
- the altitude;
- the speed;
- a set of timestamps, including the timestamp related to when the message was received, when the entry of the database was last updated, and the ITS PDU Header and GeoNetworking timestamps in the formats foreseen by ETSI [3,27];
- the vehicle width and length, if available; if not available, default values are used and a flag is set to store that they were not communicated;
- the type of road user and whether it is connected or detected;
- if available, additional information such as the exterior lights and turn indicators status.

It should be mentioned how the data listed above is the *main* set of information stored in the database, but additional data can be included depending on the type of message and on the sending entity.

Additionally, the received data is used to update, for each vehicle, its Path History (PH), i.e., the history of all the vehicle data, considering the last D_{limit} meters traveled, with D_{limit} being a configurable value, taking a default value of 300 m. The PH is stored in the database alongside the other information mentioned earlier, as *PH points*. Each “point” contains not only the position and timestamp, but also all the related information, as detailed above. Furthermore, only significant PH points are stored, to optimize memory usage. Indeed, before storing a new PH point, the S-LDM checks whether the vehicle moved for more than D^{PH}_{max} meters. If yes, the new point is stored. If not, the S-LDM further checks whether the vehicle moved for more than D^{PH}_{min} meters or changed its heading of more than H^{PH}_{min} degrees with respect to the last point stored. D^{PH}_{max}, D^{PH}_{min} and H^{PH}_{min} are configurable thresholds, tunable depending on the specific S-LDM deployment. If not explicitly set, they take the default values respectively of 20 meters, 1 meter and 10 degrees, that we proved being a good comprise between optimizing the memory usage by storing only a limited number of points, and providing sufficiently detailed historical data for each road user in the database. This allows us to store, for instance, a single PH point when the vehicle is standing still, avoiding to store a new point, without much useful content, every time a new CAM from the same vehicle is received.

The custom database, even though not based on an SQL model, supports all the main operations that are supported by SQL-based solutions, such as INSERT, DELETE, lookup of one or multiple entries based on their station IDs, in addition to geographical lookup of entries related to vehicles located within a given area.

The evaluation of the aforementioned database solutions has been performed on two different devices, with the aim of analyzing the impact of the available resources on the performance of the database solutions. The characteristics of the two devices are reported below:

- **Device 1**, high performance desktop PC: Intel Core i5-10600 K (6 cores, 12 threads), NVMe SSD
- **Device 2**, medium performance laptop: Intel Core i7-8550U (4 cores, 8 threads), SATA HDD

The main metric for the database evaluation is the average computation time, in microseconds, required by four fundamental operations:

- *INSERT* of a single vehicle (performed by the S-LDM each time new data is available for connected or non-connected objects)
- *Lookup* of a single vehicle (performed by the S-LDM when retrieving the information on a single object with its station ID identifier)
- *Geographical lookup* and extraction of the context (i.e., extraction of all vehicles within a certain radius around a reference point, to simulate the condition when a triggering condition is detected and context data should be sent to other MEC services)
- *DELETE* of a single vehicle (performed by the S-LDM to periodically clean up old entries of vehicles which are not detected for longer than 7 seconds)

Each operation has been performed 5000 times with sample data, always under the same conditions, and the obtained computation times have been averaged.

Figs. 5 and 6 show the obtained results as a function of the number of elements already inserted in each database. 95% confidence intervals are shown, even if most of the time they are very small.

Looking at the obtained results, the following take-way messages could be drawn:

- Latency-critical MEC services for high levels of automation, which require a maximum end-to-end latency as low as 10 ms, should not rely on databases writing their data to the disk or to secondary memory in general, unless strictly needed, during the operations of the service itself. Indeed, SQLite on disk always resulted to be the worst-performing, according to ours tests, with geographical area lookup times up to 6 ms, even when few vehicles are stored in the S-LDM. Indeed, if compared to our in-memory solution, writing to

[3] https://github.com/francescoraves483/S-LDM/blob/main/src/LDMmap.cpp.

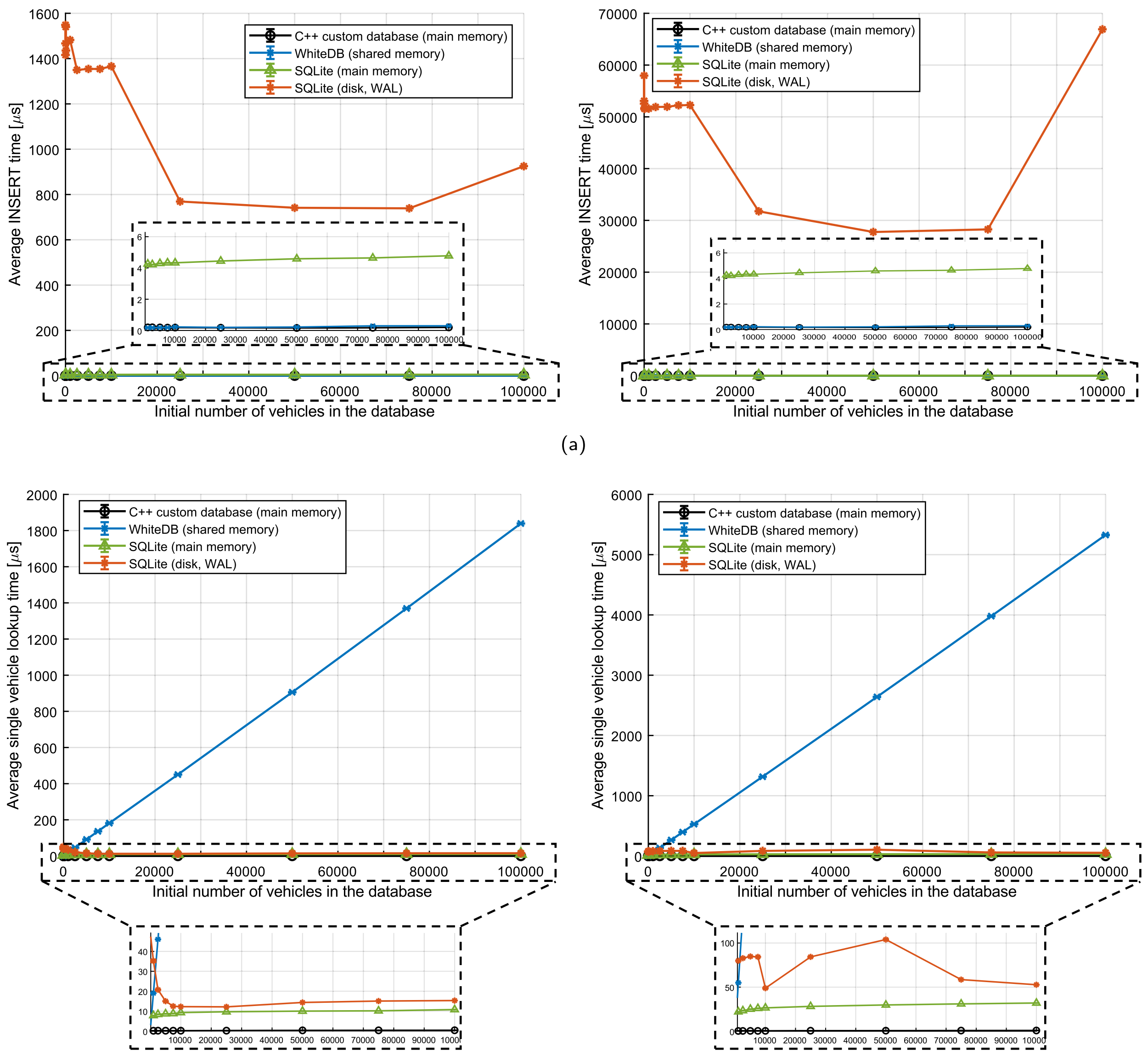
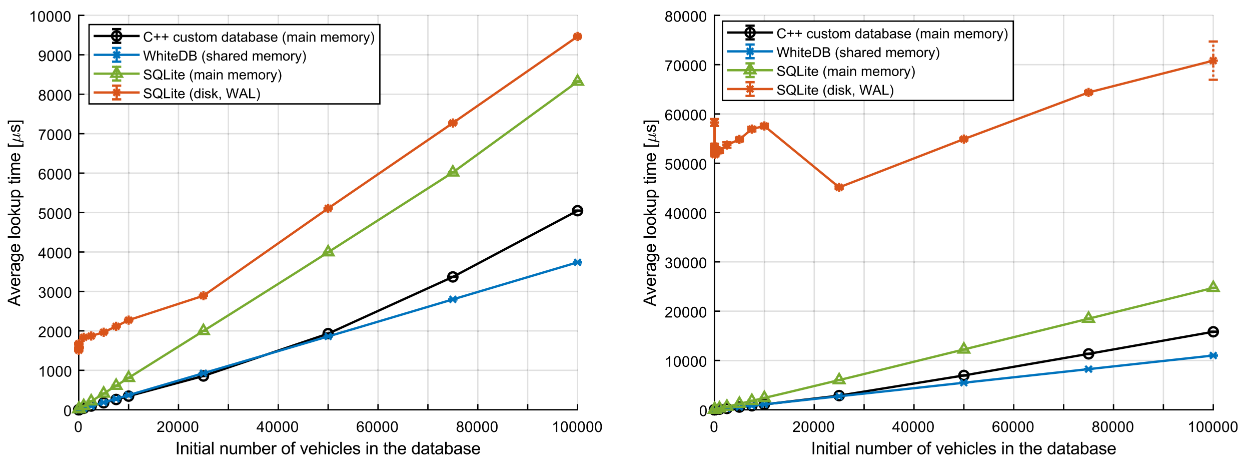


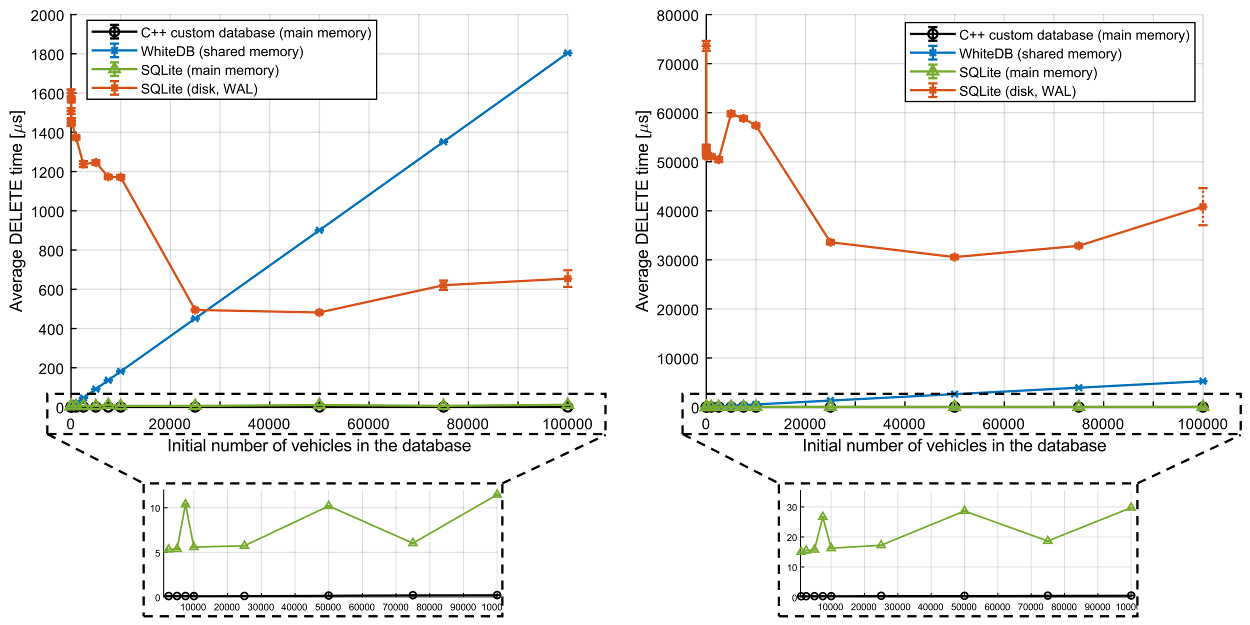
Fig. 5. Database operation average execution time, as a function of the total number of elements already inserted in the database. Plots on the left refer to device 1 (high-end desktop PC), while plots on the right to device 2 (mid-end laptop). (a) INSERT time, (b) Single vehicle lookup time.

disk with a WAL mechanism can cause up to a *57825x* INSERT delay increase for a mechanical hard drive (increasing the INSERT time from the 0.48 microseconds of our solution to around 27 milliseconds of SQLite with WAL), and up to a *3705x* increase for an SSD.

- NoSQL databases relying on unordered lists of entries, such as WhiteDB, have an operation time which is almost linearly dependent, in the worst case, on the number of stored items (i.e., vehicles and road objects) stored in the database. This is true for all operations requiring a search inside the database (i.e., all tested operations but for INSERT). This may not be ideal for the scalability of the MEC component, even though a solution like WhiteDB appears to be the best when performing geographical lookup operations, being slightly faster than the custom solution.
- When performing an INSERT operation, all tested databases (except SQLite when writing to the disk) proved to be quite efficient and well-suited to a service like the S-LDM. SQLite on main memory has an INSERT time of around 5 microseconds (almost independent of the size of the database itself), while WhiteDB and our custom solution can perform the insertion of new entries in less than 1 microsecond.
- Between SQLite in main memory and the custom database, developed specifically for the S-LDM, the best-performing appears to be the latter. Even though both can perform the needed operations in much less than 10 ms (in the order of few microseconds when the number of vehicles managed by each S-LDM instance is limited, as it is for real deployments), choosing the best performing one is preferable especially when developing 5G-enabled automotive services. Moreover, even though not shown here, the custom C++ database has been developed to support efficient locking mechanism when a thread attempts to read from the database and another tries to write to the database to perform, for instance, an INSERT operation. This situation is expected to occur almost every time a triggering condition is detected. SQLite, instead, according to the official documentation, performs a less-efficient full database-level



(a)



(b)

Fig. 6. Database operation average execution time, as a function of the total number of elements already inserted in the database. Plots on the left refer to device 1 (high-end desktop PC), while plots on the right to device 2 (mid-end laptop). (a) Geographical lookup time, (b) DELETE lookup time.

locking when more threads attempt to access the same database at the same time.

- The oscillations in the INSERT and DELETE average times with respect to the number of vehicles in the database, when employing SQLite with WAL, are likely due to the internal operation of WAL and to the disk usage at the time of testing.
- By comparing the plots on the left (for device 1) with the ones on the right (for device 2), one can also observe how the usage of a high performance CPU and disk can greatly improve the overall computation times. When centralized MEC services like the S-LDM are in place, it is thus crucial to invest not only in the underlying infrastructure (which is in any case fundamental to guarantee a sufficiently low latency and high reliability) but also on the 5G core and on the support to efficient MEC platforms, able to provide enough resources to time critical services.

In conclusion, the custom database solution appears to be the best performing option for the S-LDM, especially when considering scenarios where a large number of vehicles need to be stored in the database. Indeed, our solution is able, on average, to perform a single *INSERT* operation in less than one microsecond when no other operations are pending on the same database. It also outperforms other existing databases commonly used in state-of-the-art LDM solutions, such as SQL-based ones [38] as well as non-SQL-based solutions [19], showcasing insertion times under a microsecond compared to values over a millisecond for the two counterparts.

3.2.5. *Triggering conditions and context creation*

One of the most important functionalities of the S-LDM is its ability to detect when a certain triggering condition occurs, i.e., either a situation on the road which requires the intervention of a MEC service or a vehicle performing some action to signal that a maneuver with a high level of automation has been requested. Once one of these conditions is detected, the S-LDM computes a context around a reference vehicle or object, containing all the information on vehicles and other non-connected objects within a certain configurable radius around the reference node. This context is then made available to other MEC services, managing automated maneuvers in a centralized way, thanks to the capabilities of the underlying 5G network architecture.

One of the most important features of the S-LDM is its capability to detect the occurrence of "triggering conditions", i.e., either a vehicle performing some action to signal that a maneuver with a high level of automation has been requested, or a situation on the road which potentially requires the intervention of a centralized latency-critical MEC service. When the S-LDM detects one of these conditions, it efficiently reads the database and computes a context around a reference vehicle or other object. This context includes all the information on vehicles and other non-connected objects within a certain configurable radius around the reference node. The triggering condition detection is currently performed in a deterministic way, by analyzing a set of fields in the received messages (mainly, CAMs). For instance, the intention of performing a lane merge is detected when a vehicle switches on one of its turn indicators, encoding this information in the Exterior Lights field of the CAMs it sends to the S-LDM.

Alongside the deterministic CAM-based trigger, an AI/ML-based trigger based on traffic and vehicle kinematics prediction is currently under development. When it is integrated, both the output of the CAM-based trigger module and the one of the AI/ML-based module are going to be used to determine whether a triggering condition is occurring.

3.2.6. *REST API to other MEC services and JSON server*

After the creation of a context, the latter is made available to other MEC services through a dedicated main REST API, which encodes the pre-processed and filtered data into an easily manageable JSON format. This REST API expects the other services to expose a simple REST server, to which the S-LDM can connect as a client. Data is then periodically transmitted to the other services (with a configurable JSON message periodicity, usually lower than the update rate of the database) until a specific JSON reply is received.

Specifically, the S-LDM expects from the other services a JSON reply containing at least a field called `rsp_type`, which should be set to (i) `"ERROR"` in case errors occurred while parsing the POST request from the S-LDM, (ii) `"OK"` to indicate that the service could successfully use the information received from the S-LDM, and more information should be periodically sent, or (iii) `"STOP"` for correctly parsed information, after which no more information is needed from the S-LDM.

The S-LDM also foresees the possibility of retrieving the information on demand from the database through a separate *on demand*, lightweight JSON-over-TCP interface. Specifically, the S-LDM exposes a TCP server, called *JSON server*, and expects other services to perform requests specifying a location (i.e., latitude and longitude) and a radius, or a specific station ID. Requests are performed by sending, as TCP payload, proper JSON files containing the request parameters (i.e., latitude, longitude, radius or station ID(s)). Upon reception of a correct request, the S-LDM will then return a list of vehicles and other non-connected objects as a reply, either matching the station ID(s), or currently located within the specified area. This list is returned as a JSON file stored in the payload of a TCP packet.

3.2.7. *Web GUI*

The S-LDM also includes a web-based Graphical User Interface (GUI), enabling users and road operators to monitor the situation on the road in real time. This GUI shows the content of the local dynamic map stored inside the database and allows users to interact with the map to display additional information about each vehicle and other non-connected object (e.g., showing the ID and current heading).

Furthermore, thanks to the Station Type field of CAM messages, the S-LDM can properly classify and display with different icons several types of vehicles (such as passenger cars or heavy trucks).

3.2.8. *Vehicle local dynamic map updates*

Other than relying on a centralized LDM such as the S-LDM, vehicles may be equipped with a ETSI-compliant Local Dynamic Map [7], keeping track of all the objects and other connected vehicles perceived by the local communication and sensing capabilities.

On this basis, the S-LDM also foresees the (optional) possibility of periodically updating the information available to each vehicle, with wide, precise centralized information related to objects on the road. The updates to the single vehicle LDMs occur periodically through one or more AMQP brokers, thanks to dedicated AMQP messages containing the most relevant information for each vehicle. A separate AMQP client, running on a dedicated parallel thread, is expected to manage this process.

We highlighted this additional feature in the bottom part of Fig. 1 with dotted boxes, as it is not yet fully available in our S-LDM implementation. Nevertheless, it is planned to be deployed soon.

3.3. *Deployment and scalability options*

As mentioned earlier, with the aim of tackling scalability and supporting cross-border use cases, each S-LDM instance has been specifically thought to cover a limited portion of the road, and filter out all the messages of vehicles coming from outside this *coverage area*. Cross-border is a particularly challenging scenario, due to the need of covering the road without interruptions when moving between the MEC platforms (in turn running different instances of the S-LDM) of different network operators, belonging to different countries.

As investigated by 5G-CARMEN, 5G can help to reduce the interruption time and experienced latency when crossing the border, thanks to solutions such as local breakout and fast network reselection based on Equivalent Public Land Mobile Network (E-PLMN) [32]. However, MEC services should also be designed to support cross-border use cases in a scalable way, foreseeing multiple instances interacting with each other, or gathering the same set of messages to extensively cover the road on both sides of a border, and scale in presence of a large amount of connected vehicles.

On this basis, with the aim of extensively covering the road, several S-LDM instances should be created and deployed, each covering an area starting from where the previous S-LDMs areas end. The overall areas covered by neighboring S-LDMs should also be overlapping, in order to let each instance store a complete view of the road, even when the context generation involves a vehicle traveling near the border of the coverage area. This can be done during the instantiation phase within a MEC platform, thanks to the possibility of easily configuring the coverage area. A simple 2D representation of this mechanism is shown in Fig. 7.

As can be seen, thanks to the overlapping areas, some stretches of road are covered by more than one S-LDM instance. This enables a proper context generation even when a vehicle, like the highlighted car, is traveling near the coverage area border, and needs to be informed about all nearby road users, including the preceding red car already traveling inside the S-LDM *instance 4* area.

The full coverage area, including the portion overlapping neighboring S-LDM instances, is called *extended area* (red area for *instance 3* in Fig. 7), while the area covered by a single instance without the overlapping portions is called *base area* (area between the two dotted black lines for *instance 3* in Fig. 7). When a triggering condition is detected and a triggering vehicle is located in a common area, two scenarios may be considered:

- **In-country**: only the S-LDM with the triggering vehicle inside its base area will send the data to the other MEC services.
- **Cross-border** (e.g., if *instance 3* is in Italy, and *instance 4* is in Austria): both S-LDMs will send the data to the respective MEC services (e.g., to the MEC services located both in Italy and Austria). This ensures that vehicles connected to either one or the other operator and located in the common area can be reached by the output of the MEC services.

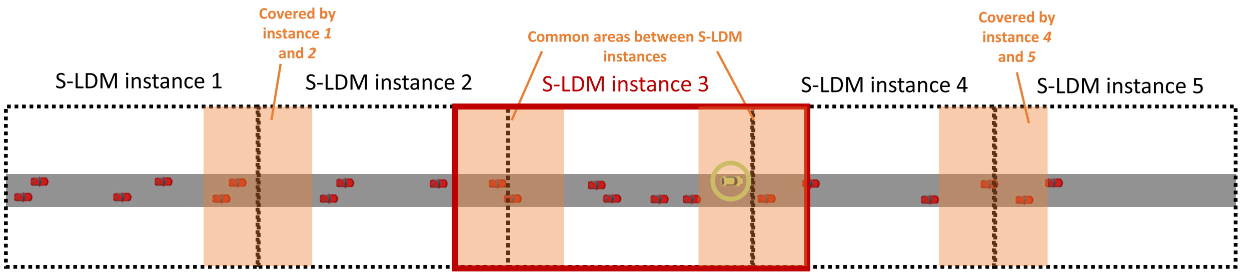

Covered by instance 1 and 2
S-LDM instance 1
S-LDM instance 2
Common areas between S-LDM instances
S-LDM instance 3
S-LDM instance 4
Covered by instance 4 and 5
S-LDM instance 5

Fig. 7. Multiple S-LDM architecture for cross-border and in-country scalability.

This mechanism further highlights the importance of an efficient filtering mechanism as part of each S-LDM instance, i.e., the Quadkey filtering described in Section 3.2.1, which is set to cover the extended area for each instance. Indeed, receiving all messages from all vehicles on the road, and then filtering them inside the Area Filter module (which requires decoding each single message) could be a computationally expensive operation, especially under high traffic conditions.

With the aim of properly tackling cross-border scenarios, the S-LDM multi-broker subscription is used, with the only requirement that the AMQP broker on one side of the border should be reachable from the other side, and vice-versa (as deployed and tested in 5G-CARMEN, thanks to 5G inter-operator connectivity). Each cross-border S-LDM instance belonging to each operator can thus receive messages coming from both vehicles attached to the operator on one side of the border, and from the ones already connected to the second operator belonging to the country on the other side of the border.

Moreover, to increase robustness on cross-border use-cases, the S-LDM includes the so-called *ageCheck* feature. This feature, as part of the message decoding sub-module (described in Section 3.2.2), reads the GeoNetworking timestamp [27] to check the age of the data stored inside the database before updating it with the new received data. This allows our service to discard potentially outdated messages received after crossing the border, due to out-of-order packets. Specifically, out-of-order messages may be received due to the jitter of the inter-operator connectivity when leveraging the multi-broker subscription.

When a set of S-LDM instances has been created to cover either an in-country or cross-border section of the road, their coverage area can also be regulated at runtime, creating new instances in case of a sudden increase in either vehicular or network traffic, or in resource usage from a single instance. Thanks to the native support for configurable coverage areas in the S-LDM, this can be managed by the underlying orchestrated edge platform scaling and life cycle management features. As an example, concerning cross-border S-LDM orchestration, the orchestrated edge platform developed by 5G-CARMEN [33] provides seamless service continuity and on demand scaling according to service resource and traffic requirements.

Thanks to its modular structure and to the flexible deployment options, the S-LDM can enable several centralized use cases, efficiently providing filtered and processed information when needed. Examples of these use cases include:

- **Centralized maneuver management**, in which the S-LDM can provide the maneuver management services with real-time information related only to the most relevant vehicles, letting such safety-critical services focus on the control task. In this use case, the S-LDM can be deployed in the MEC platforms of MNOs providing connectivity services to the vehicles, or on MEC servers available at the road operator premises. This enables the efficient provision of information to the actual maneuver coordination services, that can be interfaced with the S-LDM and be developed by the car manufacturers;
- **Smart traffic light control**, in which the S-LDM can be deployed in the MEC or cloud of a city mobility center, and interfaced with their own services to provide useful information, used then to control in a smart way the traffic light phases along one or more intersections, to reduce traffic and, consequently, emissions. The S-LDM could also prove to be useful for distributed virtual traffic light approaches such as the one presented in [39], thanks to the possibility of providing a broader view of the road to vehicles, by updating their local LDMs.
- **Centralized Platooning**, in which the S-LDM provides all the information relevant to a specific platoon of vehicles necessary for the Cooperative Adaptive Cruise Control (CACC) algorithm executed by a controller running on a MEC platform [40]. For this specific maneuvering management use case, the S-LDM could further improve the platoon maneuvers' safety with additional information about interfering vehicles in the platoon's vicinity. Indeed, including detected object information within the available context enables the development of safer platooning algorithms resilient to legacy non-connected vehicle intruders [41].

3.4. Security and privacy management

One of the main challenges of C-ITS revolves around ensuring the security and privacy of communications between C-ITS actors. In Europe, Cooperative Intelligent Transport Systems (C-ITS) rely on a centralized security model based on a Public Key Infrastructure (PKI) as outlined in [42]. This model covers both vehicle-based and road infrastructure-based stations, establishing a unified C-ITS trust domain between all ITS-S. Each ITS-S must adhere to the ETSI standards for digital certificates [43] and employ GeoNetworking for networking layer security [27]. While signed messages and digital certificates are enough for broadcast scenarios, IP-based communications (over protocols such as AMQP 1.0) demand additional security considerations, such as the use of TLS protocols [44].

Specifically, services such as the CA and DEN Basic Service, require authentication, authorization and integrity but not confidentiality [45]. ETSI defines the security profiles of both services, where the mechanisms for signing with an authorization ticket are specified. To comply with this mechanism, each ITS-S should be capable of constructing a certificate chain from the ITS-S that generated the message to the trust anchor such as a Root Certificate Authority (Root CA) that is known by the receiving ITS-S. Additionally, in order to ensure that an ITS-S is able to send CAMs and DENMs without disclosing its identity but being still accountable for their transmission, ETSI mechanisms support pseudonymity by using temporary identifiers in all messages. Moreover, to avoid the possibility of CAM or DENMs messages of a given ITS-S to be tracked, ETSI outlines strategies for limiting the amount of static information that is included in such messages, achieving unlinkability.

As described in Section 3.1, the S-LDM has been designed to follow the C-Roads specifications, including the standards related to security and privacy for the exchange of messages over AMQP 1.0. In particular, all communication between the S-LDM AMQP client and a given

AMQP broker are expected to use Transport Layer Security (TLS) with mutual authentication using standard X.509 certificates when establishing the connection as mandated in the requirements for the so-called Basic Interface (BI) [46]. With this mechanism, in addition to C-ITS message signing and authentication with the use of certificates according to [43], IP-based communication is expected to be secured by X.509 certificates according to IETF RFC 5280. Furthermore, all ETSI C-ITS messages supported by the S-LDM are expected to follow the header and certificate formats outlined in [43] as well as the structure defined in [27] for GeoNetworking secured packets, in the case of messages containing GeoNetworking and BTP headers. It should be mentioned, however, that it is up to the vehicles to include proper certificates and use TLS, and the S-LDM itself, since it acts as a middleware between the vehicles and other MEC services, can flexibly support non-secured messages too.

Finally, to protect the privacy of users, and depending on the use case, the S-LDM GUI, described in Section 3.2.7, can be configured to show different levels of information. Even though the information stored inside the S-LDM is designed to be viewed with full details only by road operators for real-time road monitoring, some applications may require some data to be hidden due to privacy laws or concerns. It is thus possible to selectively hide, for instance, the vehicle IDs or road user types, so that more anonymous information is displayed, while still storing internally (and locally to the S-LDM) the more detailed information required by other centralized V2X MEC services.

4. Laboratory pre-deployment evaluation

The S-LDM has been extensively evaluated first in a laboratory setting, and then in the field, thanks to the deployment on the 5G infrastructure managed by the 5G-CARMEN project. For the latter, two connected vehicles provided by Stellantis have been used, sending their messages to the S-LDM instances located respectively on the MEC platforms of TIM, Magenta Telekom and Deutsche Telekom AG, though the 5G New Radio (NR) *Uu* interface.

Before the deployment as a MEC service on the 5G-CARMEN architecture, a series of in-lab tests have been performed on a local instance of the S-LDM. This testing campaign was mainly aimed at evaluating the performance of the S-LDM, connected to a local instance of an ActiveMQ broker, on well-known hardware running Ubuntu 20.04 LTS.

During the development and in-lab pre-deployment tests, we resorted to a realistic vehicle emulation and simulation framework, called *ms-van3t*[47]. ms-van3t is a full-fledged V2X framework, available on GitHub with an open-source license [48]. It couples SUMO[4] and ns-3,[5] to provide at the same time simulation and emulation features. The first lets users simulate large scale scenarios with different access technologies (IEEE 802.11p, C-V2X, LTE, NR-V2X), while the second enables the emulation of an arbitrary number of vehicles, after setting up a realistic scenario on a given road topology.

The emulation mode of ms-van3t provides two main options. One option is to directly broadcast messages generated by vehicles over a physical Network Interface Card (NIC) of the device running the framework, while the second provides for the transmission of messages via UDP from the simulated vehicles to an external server. We relied on this last possibility, combined with a custom AMQP 1.0 client relaying UDP messages to ActiveMQ, to emulate the presence of vehicles sending their messages to an AMQP broker. Concerning the emulated scenario, we configured ms-van3t to emulate vehicles traveling on a stretch of the A22 motorway, between Trento Nord and San Michele all'Adige, on which the 5G-CARMEN project focused for most of the in-country pilots. It is worth highlighting that, since the S-LDM receives messages directly from the AMQP broker, its behavior will be exactly the same, no matter whether these messages originate from a simulated scenario or from real vehicles.

The testbed setup for the in-lab tests is depicted in Fig. 8.

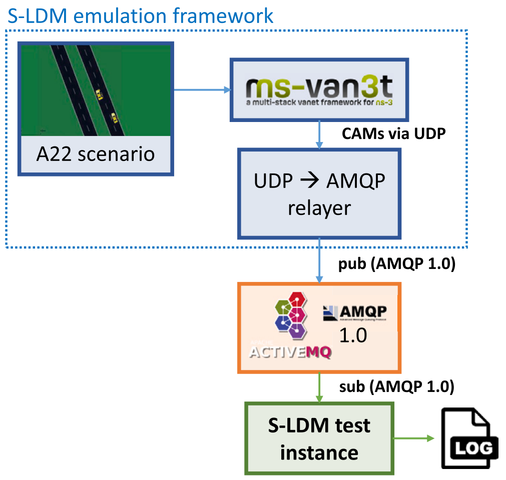


Fig. 8. Testbed setup with ms-van3t [47] for the laboratory pre-deployment performance tests.

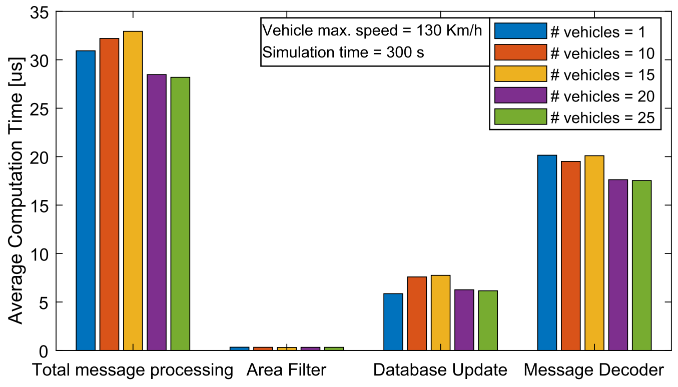


Fig. 9. S-LDM pre-deployment sub-modules performance tests.

4.1. S-LDM sub-modules

The main aim of in-lab tests was to assess the S-LDM performance, in terms of needed average computation time, focusing on three main sub-modules (with reference to Fig. 1): (i) Message Decoder (Section 3.2.2), (ii) Area Filter (Section 3.2.3) and (iii) Main Database (Section 3.2.4).

All in-lab measurements have been performed by setting a local instance of the S-LDM to cover an area around the chosen stretch of road. It should be noted that the number of vehicles in simulation is limited by the hardware used for the tests. Indeed, a non-negligible single-core computational effort is required by ms-van3t, when the number of vehicles increases beyond a certain amount. The device used for the tests, running an Intel Core i7-10750H CPU, was able to properly emulate up to 25 vehicles without slowing down the S-LDM or any other process running on it. This justifies why we chose to test up to 25 vehicles during the initial in-lab evaluation. A more extensive scalability study, performed after the S-LDM deployment on the 5G-CARMEN architecture, is described in Section 6.

The most relevant results are depicted in Fig. 9, together with the main ms-van3t emulation parameters. Notice vehicles can travel at a maximum speed of 130 km/h, in accordance with the speed limits of the selected stretch of road. It is evident that the S-LDM maintains a consistent level of performance up to at least 25 vehicles, with only minor fluctuations between different numbers of vehicles. These slight variations, which do not exceed 3 microseconds, may be attributed to

[4] https://sumo.dlr.de.

[5] https://www.nsnam.org/.

Table 1
Results of the S-LDM REST API one-way and RTT latency measurements. Average and standard deviation (σ) values are reported over 6 independent tests.

	Average one-way latency	Maximum one-way latency	Maximum one-way latency (excluding the first request)	Average RTT
Average over 6 tests [ms]	0.9950	2.2047	1.4942	2.2900
σ over 6 tests [ms]	0.0196	0.0744	0.4457	0.0715

small kernel scheduling time oscillations. Overall, this suggests that the S-LDM is capable of scaling effectively, at least when dealing with a limited number of subscribers.

Furthermore, in comparison to the solution presented in [19], the S-LDM, when serving 20 vehicles, demonstrated superior performance with a total message processing time of just 24 microseconds, significantly lower than the 950 microseconds recorded for its counterpart.

Furthermore, the results show how the most demanding sub-module is the Message Decoder, further justifying the Quadkey-based filtering approach described earlier, which enables filtering without the need of decoding each single received message.

Finally, all the measured times are much lower than 10 ms: this is considered as an upper bound to the end-to-end latency in the 5G-CARMEN project [49]. Indeed, the measured values reveal how few tens of microseconds are needed to fully process each message. This proved to be also noticeably lower than the latency contribution introduced by other components in the 5G-CARMEN architecture, demonstrating how the S-LDM can be an efficient low-latency 5G enabler for high levels of automation.

4.2. REST interface

After the evaluation of the performance of the main modules up to the database update, we performed a second set of laboratory test, focused on the REST interface used to transfer the context to other MEC services. Specifically, we analyzed the time needed to transfer the context data through the dedicated REST API, as depicted in Fig. 1.

Six independent tests have been performed, each under the same configuration, by transmitting the context data to a Python Tornado sample server,[6] acting as MEC service receiving the information from the S-LDM. 100 POST requests have been performed for each test, with a periodicity of 1 second. The results are reported in Table 1.

The Table reports the average and standard deviation values calculated over the six tests, focusing on the average and maximum one-way latency of each test, the average RTT (i.e., from the POST request to the reception of the reply from the other MEC service), and the maximum one-way latency after the exclusion of the first POST request. Better results are obtained in this case, since the first request usually requires more time due to the need of establishing the connection.

As can be seen, the results show that the average one-way delay is slightly less than 1 ms, with a fairly low standard deviation and with few maximum peaks up to around 2 ms. When sending context data to other MEC services, this added delay can be deemed completely acceptable, while ensuring, at the same time: (i) a standardized and interoperable REST interface for communication between containerized components in the 5G-CARMEN MEC platforms and (ii) the objective of keeping a full-chain latency under 10 ms [49]. Furthermore, an average RTT of around 2.3 ms ensures that periodic context updates can be sent to other MEC services with a periodicity as low as 3 ms.

[6] Available here. https://github.com/francescoraves483/S-LDM/tree/main/tester/Python-REST-server

5. On-road evaluation

After the first in-lab measurements, the S-LDM has been integrated as a container inside the 5G-CARMEN orchestrated edge platform [33], in turn executed on the MEC platforms of the three network operators participating in the project. It is worth mentioning how the S-LDM can be easily containerized and configured for deployment thanks to a dedicated Dockerfile available in the GitHub repository.

The deployment of our service on actual MEC platforms allowed us to begin a number of road test campaigns, letting us validate the S-LDM capabilities in real-world V2X scenarios. During each test, a 5G Non-Standalone (NSA) *Uu* link enabled the continuous reception of data on both connected and non-connected objects via AMQP 1.0.

As mentioned in Section 4, all measurements have been performed with two Stellantis vehicles equipped with V2X OBUs. The OBUs installed in the connected and automated vehicles are based on L3 Pilot [50] Maserati prototypes that were further enhanced with a perception system for High Automated Driving (HAD) in the 5G-CARMEN project. The V2X OBUs can communicate via *Uu* for 5G communication with the edge, and *PC5* for short-range communication. The *Uu* network connectivity is enabled by a 5G New Radio (NR) modem, ensuring the required data rate performance and taking advantage of the 5G-CARMEN cross-border features. The connectivity towards the S-LDM has been configured to ensure that the data transfer matches the on-board detection frequency, at a higher rate (i.e., 20 Hz) than what is normally foreseen by the standards for V2V communication [3], while also guaranteeing continuous service, and utilizing the shortest possible path to reach the MEC services and the S-LDM itself. The OBUs use a dedicated AMQP client for exchanging V2N packets, which include CAMs and virtual CAMs, with the AMQP broker.

The two main aims of the road tests can be summarized as follows:

- Evaluate the S-LDM capability of receiving CAMs (and virtual CAMs) at a very high rate, i.e., 20 Hz, through AMQP 1.0 and with the support of a 5G network.
- Evaluate the performance of the S-LDM when dealing with real-world connected vehicles and other non-connected objects.

5.1. In-country evaluation

The primary focus of this Section is the results of the most significant tests involving the MEC platform managed by TIM in Italy. Section 5.2, related to cross-border tests, involves instead both the edge platform in Italy and the one managed by Magenta Telekom (MTA) in Austria. It is also to be noted that the S-LDM deployed in Germany has been used for several additional test campaigns, which yielded similar results as the ones presented here.

Fig. 10(a) reports the Cumulative Distribution Function (CDF) of the overall messaging processing times, for each message received by the S-LDM during the whole duration of a test performed on the Italian side of the E45 motorway (i.e., the A22 motorway). This test has been performed between Sterzing-Vipiteno and Brennerpass, and the reported values are related to the two Maserati Prototypes (herein labeled as “Vehicle 1” and “Vehicle 2”). As can be seen, most message decoding and database update operations occurred in less than 50 microseconds, with very few spikes up to around 0.5-0.7 ms. It should be noted that

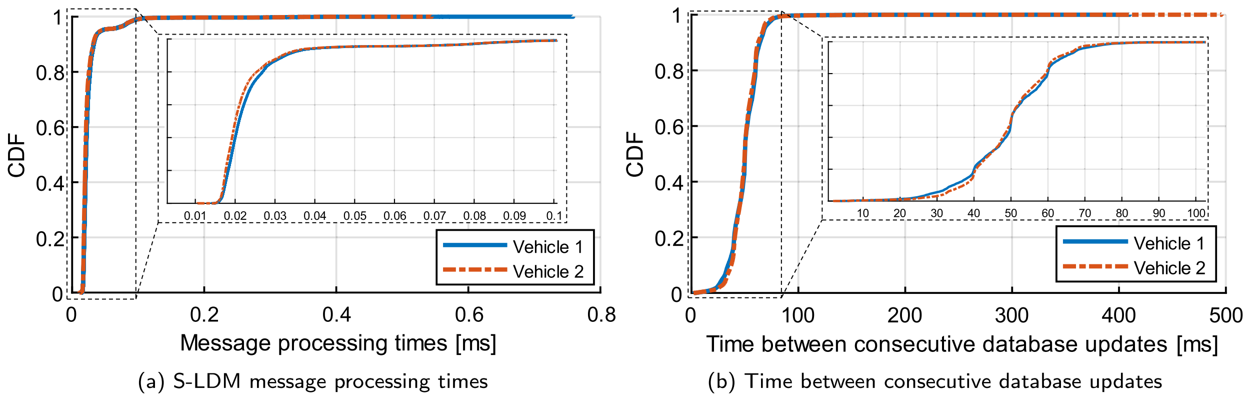


Fig. 10. Empirical Cumulative Distribution Functions results of the in-country on-road tests. The plots show the results related to each vehicle, during the whole road test duration.

these peaks are always caused by a single message among thousands of processed messages.

The outcome, in terms of performance, is thus good. Furthermore, despite the numbers are slightly different (due to the different resources assigned to the S-LDM process), the orders of magnitude are consistent with the pre-deployment results, as the great majority of messages are decoded and processed in a few tens of microseconds.

It was also possible to measure how much time occurs between two consecutive database updates for the same connected vehicle or detected object. This metric (which we named *Instantaneous Update Rate*) was gathered to understand how often the database was actually being updated, considering the processing times and network conditions on the A22 motorway. It is indeed crucial to be able to update the local dynamic map as quickly and promptly as possible, since it provides data to other 5G-enabled latency-critical services. The *Instantaneous Update Rate* is computed as the time difference between when the information about a given road user is received and when the data about the same user was last updated in the database.

The results are plotted in Fig. 10(b).

It can be observed that the database is updated on average every 50 ms for each road user, which is in line with the high-frequency generation of CAMs at 20 Hz. Although with a very low probability, few spikes were observed. These peaks are likely due to the jitter and delay caused by the underlying network architecture, as no significant performance issues were detected throughout the whole test duration, as reported in Fig. 10(a). They are nevertheless occasional and affect single packets, and they should not cause any noticeable issue in the services relying on the S-LDM to gather an up-to-date map of the road.

By examining the CDF, it is also possible to determine how, in more than 95% of cases, the database can be updated in under 70 ms after the previous update. As this periodicity is highly affected by the jitter of the underlying network architecture, which is based on 5G NSA, this result can be considered quite promising. This is especially true when considering that, according to ETSI, the standard highest CAM transmission frequency is set to 10 Hz [3]. According to the tests presented here, a network-intensive transmission of messages at 20 Hz is viable and can lead to tangible advantages, enabling the creation of an overall map of the road with update rates higher than what is currently foreseen nowadays.

5.2. Cross-border evaluation

The S-LDM can handle cross-border scenarios thanks to the subscription to multiple AMQP brokers as described in Section 3.3, which, in the 5G-CARMEN architecture, is in turn enabled by a dedicated inter-MEC communication infrastructure. With the aim of testing how the S-LDM performs in cross-border scenarios, several test runs were performed with two Maserati vehicles (as described in Section 5.1) along the Brennerpass border, on the E45/A22 motorway between Austria and Italy. For these tests, two S-LDM instances were deployed respectively in the MEC platforms managed by TIM in Italy and MTA in Austria. The same instances were then configured as subscribers to two AMQP 1.0 brokers, one deployed on the Italian MEC platform, and one on the Austrian MEC platform. Both S-LDM instances were set to cover overlapping areas around the border, in such a way that the whole border area was well-covered by both S-LDMs instances. In addition to evaluating the S-LDM performance when deployed in different MEC platforms, the disconnection time due to the MNO network reselection time at the border, from the S-LDM point of view, has been evaluated as well.

The tests reported here have been performed on the E45 stretch of road between Nößlach (Austria) and Brennerpass (Italy), focusing on the Austria to Italy direction as a significant setting. The path followed by one of the Maserati vehicles is depicted in Fig. 11. The map takes as reference the S-LDM instance in Italy, and shows, in blue, the positions recorded through messages received from the Italian broker, and, in red, the ones gathered thanks to the CAM messages received from the Austrian broker (thanks to the multi-broker subscription for cross-border scenarios). Additionally, this plot shows how the S-LDM can be used, as an additional feature, to understand under which network a vehicle is connected to, given its geographical position. Indeed, thanks to the multi-broker subscription, it is possible to distinguish which messages are coming from which broker (and, consequently, which messages are coming from which network).

The overall outcome of the cross-border tests was positive, since our service was able to receive real-time information to track connected vehicles and detected objects on both sides of the border, by processing the messages from the respective AMQP brokers. Even though several runs have been performed, only one of them is reported (being the most significant) in order to make a more descriptive outline of the results.

Fig. 12 shows the normalized message delay as a function of the time since the beginning of the test, taking as reference the S-LDM instance running on the TIM MEC platform. The normalized message delay represents a measurement of the delay between message transmission from vehicles and reception by the S-LDM, minus the minimum observed during the whole test. If d_i is the delay measured for packet i, then the normalized message delay for the same packet is computed as:

$$\hat{d}_i = d_i - \min_i d_i \quad (1)$$

The minimum normalized delay thus corresponds to a value of “0”. Using a normalized delay value was necessary as the time synchronization between the MEC platforms of the different MNOs was not guaranteed, thus not ensuring consistent measurements in case of absolute delay values.

As can be seen, the normalized delay remains mostly stable during the whole test duration, with few sparse peaks, whose duration in time is always relatively short and under a second. These spikes

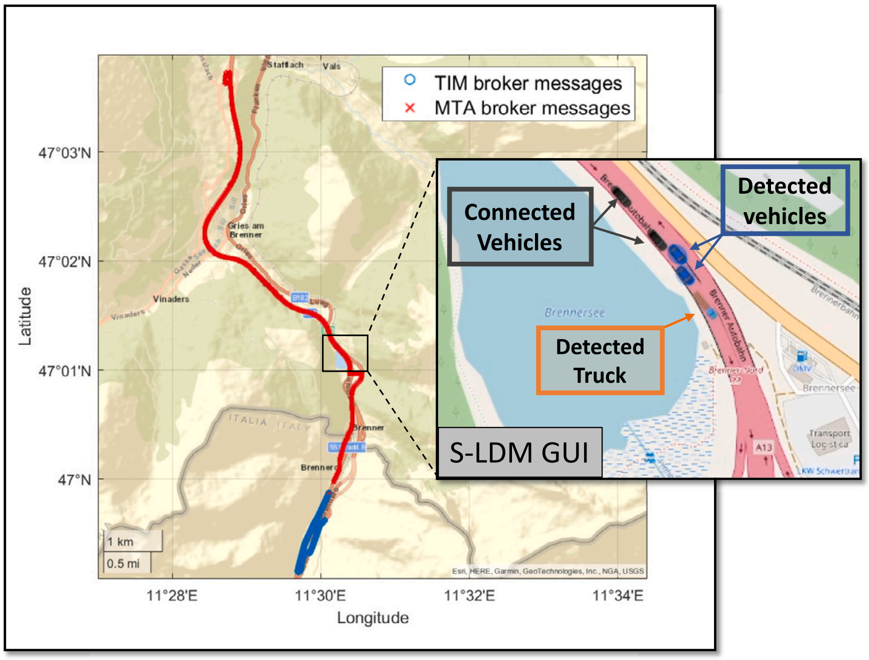


Fig. 11. Path followed by one of the two Stellantis vehicles during the cross-border road tests from Austria to Italy. The messages received by the S-LDM from the TIM network and AMQP broker are depicted in blue, while the messages from the MTA network and broker are depicted in red.

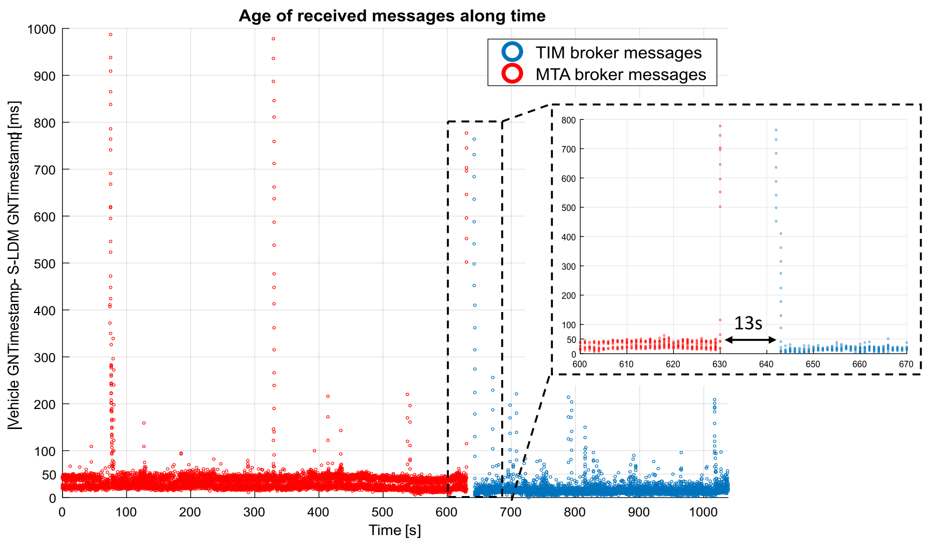
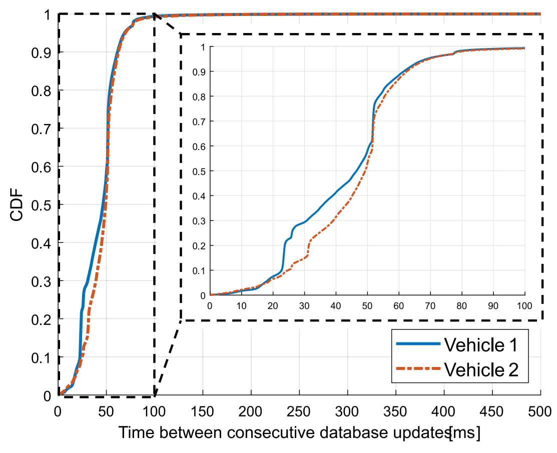


Fig. 12. Normalized message delay (i.e., a measurement of the delay between message transmission from vehicles and reception by the S-LDM, normalized to the minimum observed during the whole test) along the duration of a single test run, for a single vehicle. The S-LDM instance running on the TIM MEC platform has been taken as reference.

are probably due to the jitter and delay caused by the underlying 5G NSA network, and are expected to be noticeably reduced when moving to a full-fledged 5G SA architecture. Furthermore, it is possible to observe how the average normalized delay for the MTA broker is, overall, slightly higher. This is expected, as the data shown in Fig. 12 is taken from the S-LDM instance in Italy, with respect to which all messages from the MTA broker in Austria experience an additional inter-MEC communication delay.

Finally, it was possible to observe a disconnection interval of only 13 seconds, when moving from Italy to Austria, which remained quite stable for all the test runs along the border. Indeed, focusing on the direction from Austria to Italy, we measured an S-LDM average disconnection time, due to fast network reselection, of 13.3729 s, with a standard deviation of 2.0203 s, and a 95% confidence interval of ±0.7062 s.

The CDFs of the *Instantaneous Update Rate* of the S-LDM, concerning both the TIM and MTA instances and during the most significant Austria-to-Italy cross-border test, are reported in Fig. 13. As can be seen, and in line with the in-country tests, it is possible to observe how around 80% of the time, for a given vehicle, the database is updated with a periodicity of 50 ms or less (lower values are possible due to the underlying network jitter). As in the previous case, this is in accordance with the 20 Hz CAM frequency. Very similar results could be observed concerning both the S-LDM instance deployed to the TIM MEC platform, and the one deployed to the MTA MEC platform, proving how the


(a) TIM MEC platform S-LDM instance

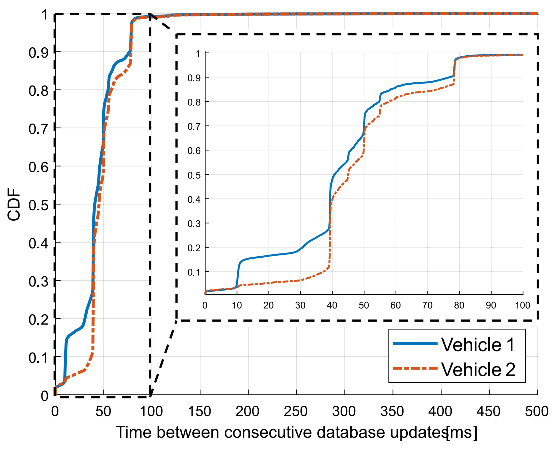
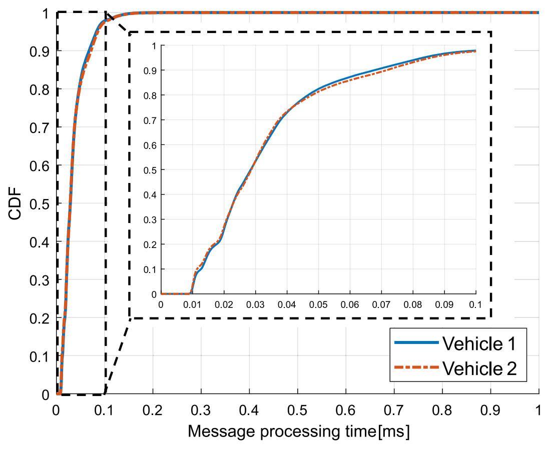

(b) MTA MEC platform S-LDM instance.

Fig. 13. Cumulative Distribution Function of the time between consecutive database updates for each of the Maserati vehicles during the most significant Austria-to-Italy cross-border test.


(a) TIM MEC platform S-LDM instance

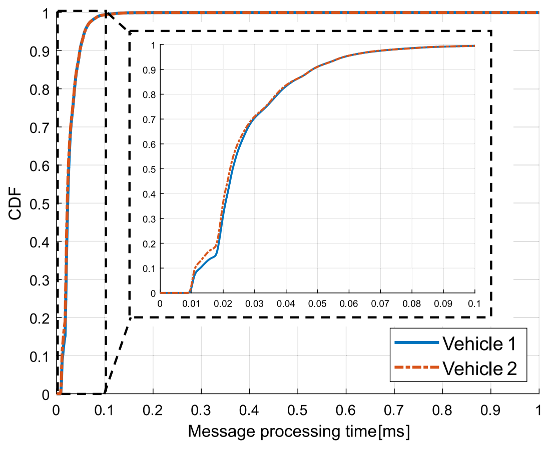

(b) MTA MEC platform S-LDM instance.

Fig. 14. Cumulative Distribution Function of the message processing times for each of the Maserati vehicles during the most significant Austria-to-Italy cross-border test.

S-LDM can efficiently store and provide a real-time map of the road in cross-border scenarios.

Finally, it was possible to evaluate the overall message processing times during the whole duration of the same cross-border test. The CDFs related to both S-LDM instances are depicted in Fig. 14.

Similarly to the in-country case, 90% of messages are processed in under 72 microseconds on the instance at the TIM MEC, and 50 microseconds on the instance at the MTA MEC. The difference between the two, given the microsecond-scale of all values, can be explained by the different hardware configurations supporting the two MEC platforms. Furthermore, the average processing time proved to be around 35 microseconds for the S-LDM deployed in Italy, and 28 microseconds for the one instantiated in Austria. These values are consistent with the in-lab and in-country road tests, and show how the S-LDM can perform very well under different circumstances, including cross-border scenarios.

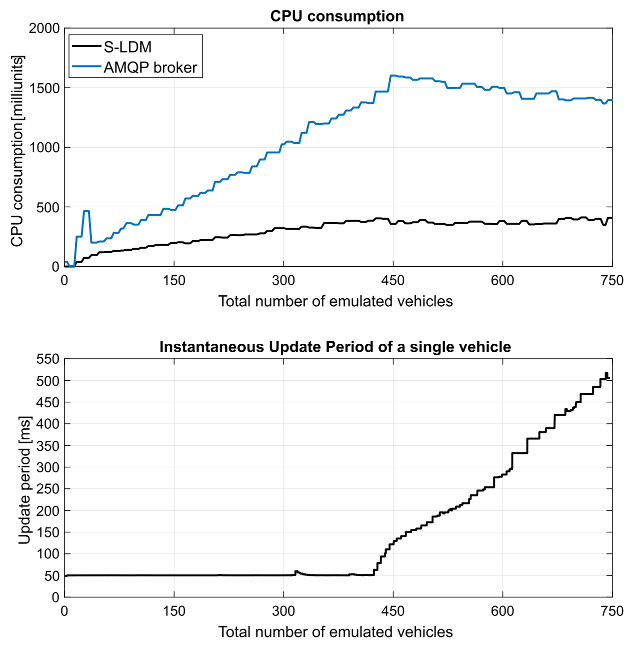
Furthermore, all the results presented in this Section, and in the previous one, allowed us to validate our component in the field, showing its effectiveness in processing messages with very low latency, storing a highly efficient map of the road and providing a filtered and processed context to other MEC services when needed, thanks to a dedicated REST interface.

6. Scalability features and analysis

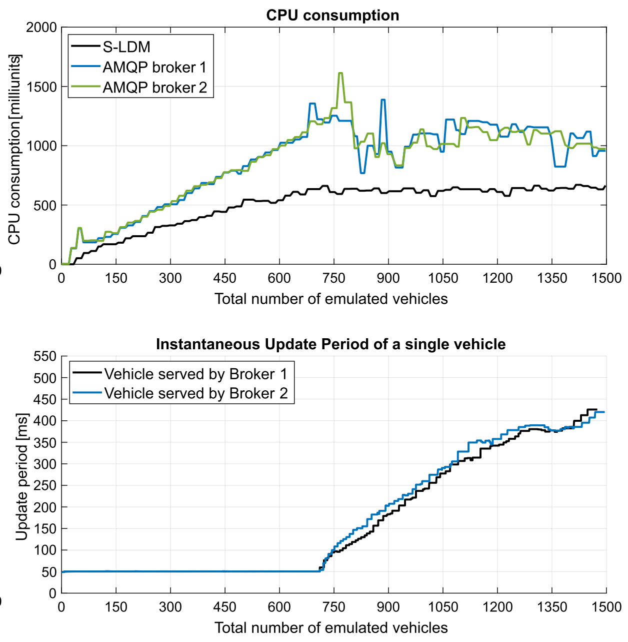
Scalability represents one of the major challenges for most V2X MEC services, especially considering the increasing market penetration rate resulting in a higher connected vehicle density. As each S-LDM instance is designed to cover a limited portion of the road to tackle scalability, multiple instances are expected to be deployed in future practical scenarios covering a whole motorway or urban road, possibly adapting the coverage areas in real-time, depending on the number of active message producers (i.e., vehicles) and/or on resource usage.

A series of scalability tests have been performed measuring the performance of the service in terms of computational time and resource utilization in order to validate its capability of (*i*) storing real-time data and (*ii*) providing it to other services, when an elevated number of vehicles are being served.

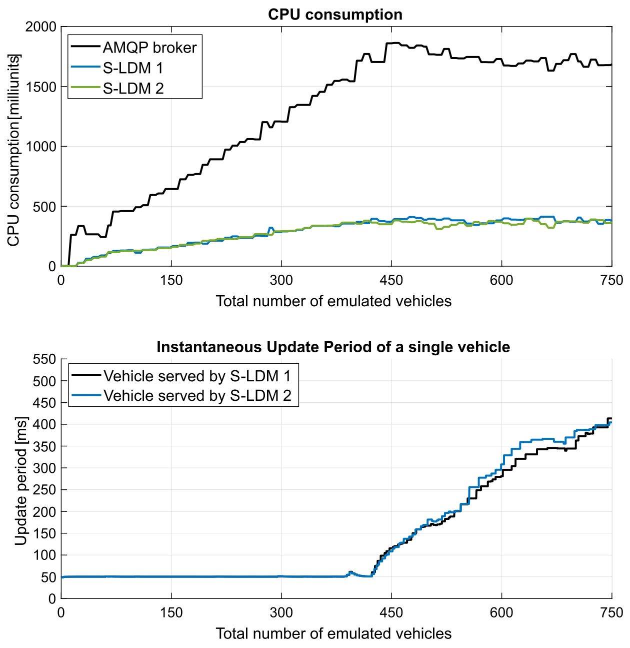
It should be mentioned how, to the best of our knowledge, this is the first study to extensively study the scalability of a system like the S-LDM.



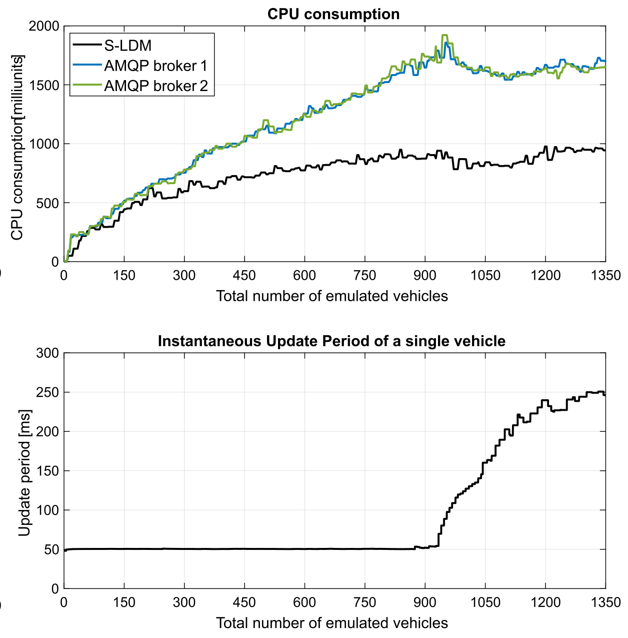
(a) One S-LDM instance subscribed to a single AMQP broker.



(b) One S-LDM instance subscribed to two AMQP brokers.



(c) Two S-LDM instances subscribed to a single AMQP broker.



(d) One S-LDM instance subscribed to two AMQP brokers, with two AMQP clients per broker.

Fig. 15. (Top): CPU consumption of S-LDM and AMQP broker pods in milliunits as a function of the number of vehicles being emulated. (Bottom): *Instantaneous Update Rate* of a reference vehicle (i.e., reference stationID) as the total number of vehicles managed by a single S-LDM instance increases.

6.1. Real-time data storage scalability

As described in Section 3.3, in order to extensively cover the road, S-LDM instances are envisioned to be created such that each one covers an area starting from where the previous instance area ends. When considering cross-border scenarios, there is a need for every coverage area to overlap nearby ones. Therefore, in the case of overlapping areas at the border between two countries, the respective S-LDM instances are expected to receive messages from multiple AMQP brokers.

Furthermore, with the aim of evaluating the architecture performance between the S-LDM and the AMQP broker services, three scenarios have been considered for the tests:

1. One S-LDM instance subscribed to a single AMQP broker.
2. One S-LDM instance subscribed to two AMQP brokers.
3. Two S-LDM instances subscribed to a single AMQP broker.

All the tests have been performed leveraging the testbed depicted in Fig. 16, deployed on an OpenStack platform, running Ubuntu Server

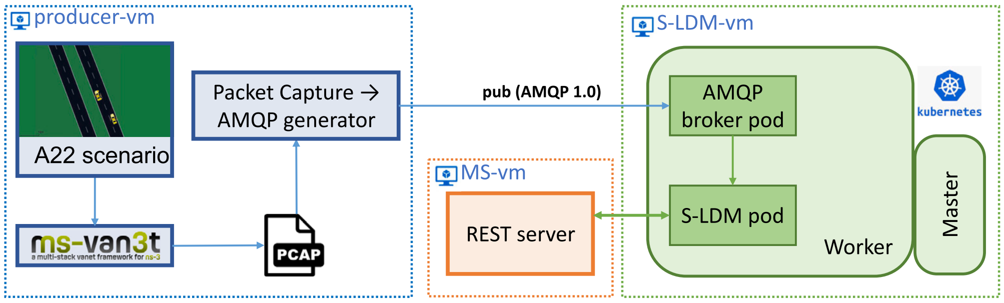


Fig. 16. Testbed setup for S-LDM laboratory scalability performance tests.

20.04 LTS. A Kubernetes-orchestrated cluster was deployed, using the `kubeadm` toolbox in the *S-LDM VM*, in order to emulate the behavior of the service running at the MEC in the 5G-CARMEN on-road architecture. A variable number of connected vehicles sending CAMs at 20 Hz (as foreseen by 5G-CARMEN) were emulated with an ad-hoc emulator running on the *Producer VM*. This emulator can realistically emulate a very large number of vehicles, starting from pre-recorded packet traces generated offline by ms-van3t [47], which is in turn configured with a scenario on the A22/E45 motorway. `kubectl` has been used to monitor and log the resource usage of each pod and container running on the cluster. For each test, the number of emulated vehicles was increased by 10 every 15 seconds, up to 750 vehicles, for a total duration of 1125 seconds. Finally, a sample automated driving centralized Maneuvering Service (MS), has been deployed on a third *MS VM*, to create a setup similar to the on-road configuration used for the in-country and cross-border tests on the 5G-CARMEN architecture.

Concerning test 1, representing a simple in-country scenario, the CPU consumption shows a linear increase for both S-LDM and AMQP components up to 430 emulated vehicles, as depicted in the top part of Fig. 15(a). When the number of vehicles increases over this value, the CPU consumption becomes almost constant for the S-LDM and slowly decreases for the AMQP broker. This behavior translates in an increased time between database updates performed by the S-LDM for a given vehicle, as shown in the bottom part of Fig. 15(a), where the periodicity remains stable at 50 ms (due to the 20 Hz CAM periodicity of the emulated vehicles) and then it starts significantly increasing from 430 emulated vehicles. These results are due to an AMQP-level buffering behavior that results in messages getting sent in “bulks” when a very large number of producers are sending data to the S-LDM through AMQP 1.0. For a given vehicle being served, this behavior not only introduces message delays while they are getting buffered, but also causes multiple messages to be placed in the same “bulk”, which results in groups of consecutive database updates performed in very short intervals (lower than a millisecond). For descriptive purposes, only database update intervals larger than a millisecond are considered for Fig. 15. This result also allows us to conclude that a single S-LDM instance, connected to a single AMQP broker in an in-country scenario, is able to handle up to 430 vehicles without any performance loss. This value can thus be used to properly define a deployment strategy, with proper coverage areas, for multiple instances. Considering vehicles traveling at 130 km/h and spaced 36 m, in a two-lane per direction of travel motorway, a single S-LDM instance would be thus able to cover at least, roughly, 3.9 km of road.

When considering multiple AMQP broker subscriptions, multiple AMQP client threads need to be running on the S-LDM (one client for each broker). With the goal of assessing the performance of the S-LDM while subscribed to multiple brokers, such as in a cross-border scenario, test 2 was performed with a single instance of the S-LDM subscribed to two AMQP brokers. It is worth mentioning that the number of emulated vehicles sending messages was increasing by 10 every 15 seconds, up to 750 vehicles, for each of the AMQP brokers. Therefore, if compared to test 1, the total number of served vehicles in the database, at every time step, is the double, while keeping the same number of vehicles being handled by each of the AMQP client threads. The CPU consumption of all pods with respect to the number of vehicles being served by each component is depicted in Fig. 15(b). Intuitively, due to the fact that each broker is serving the same number of vehicles at every step on both tests, both AMQP brokers present a linear increase of CPU usage presenting very similar rate as in test 1. On the other hand, the S-LDM presents a linear increase of CPU usage at twice the rate, reaching a maximum use of 660 CPU milliunits. As depicted in Fig. 15(b), the CPU consumption of all components stops increasing after a total number of 710 emulated vehicles. These results are in line with the database update periodicity for a given vehicle being served by a given AMQP broker, depicted at the bottom of Fig. 15(b), which shows that up to 710 vehicles can be served without any performance loss. This number corresponds to slightly less than the double of the single broker subscription case.

Test 3 was performed with the aim of testing the AMQP performance when multiple S-LDM instances are subscribed to it. All emulated vehicles were “traveling” in an area covered by two S-LDM instances, representing an overlapping area between two adjacent S-LDM instances. At the top of Fig. 15(c), the CPU consumption of all pods with respect to the number of vehicles being served by each service is depicted. The results show a linear increase of CPU consumption by all services up to a value of 430 emulated vehicles, as with test 3, after which the resource usage remains stable and the performance starts to decrease. The main outcome of this test shows that the CPU usage of the AMQP broker does not seem to be affected significantly when serving multiple S-LDM instances, with a maximum increase of only 16% with respect to the outcome obtained in test 1. The outcome of this test, combined with the results of the previous experiments, shows the presence of a performance bottleneck at an AMQP level, for high numbers of connected vehicles. This bottleneck, as can be inferred from the results of test 3 (i.e., from the CPU usage of the broker which is not significantly increased when serving two S-LDM instances), is however not due to the AMQP broker but rather to the AMQP client module within the S-LDM, which is based on the Qpid Proton library. Thus, even though an already significant number of vehicles can be managed by each single S-LDM instance, future performance improvements may consider including and testing other AMQP 1.0 client implementations.

Finally, an additional performance test has been performed with the goal of benchmarking the S-LDM performance when running multiple AMQP client threads for each AMQP broker. A similar scenario to the one of test 2 has been considered, where two AMQP client threads were connected to each of the two AMQP brokers. For a given AMQP broker, each of the AMQP clients has been configured to subscribe to messages coming from an area half the size of the one considered for test 2, hence selecting half of the previously considered Quadkeys. The clients together are set to cover the full area of test 2. As depicted at the top of Fig. 15(d), a linear increase of CPU consumption by all services is observed up to a value of 930 emulated vehicles. As in all previous

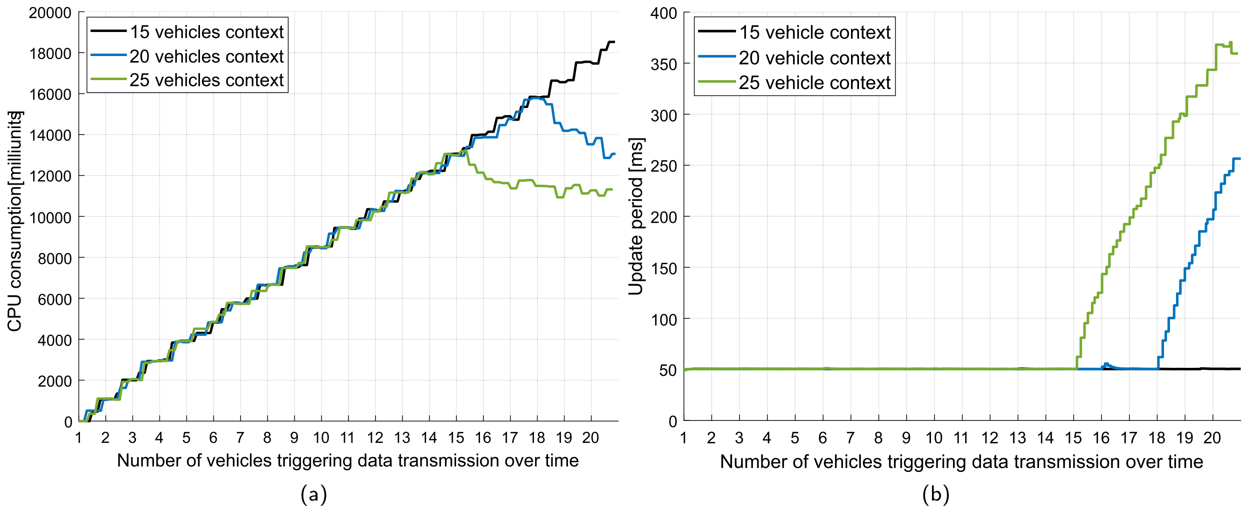


Fig. 17. (a) CPU consumption of S-LDM pods, measured in milliunits, with respect to the number of vehicles requesting context information and the number of vehicles located within the context range of each requesting vehicle. (b) *Instantaneous update periodicity* of a reference vehicle (i.e., reference stationID), with respect to the number of requesting vehicles managed by a single S-LDM instance and different numbers of vehicles in the triggered context.

tests, the database update periodicity for a given vehicle being served by a given AMQP broker, depicted at the bottom of Fig. 15(d), shows a stable, fast behavior up to the same point where the linear increase of CPU consumption tapers off. The outcomes of this experiment show how running more than one AMQP client thread for a given AMQP broker further increases the number of vehicles manageable by the S-LDM without any performance loss. Specifically, we observe an increase up to almost 30% with respect to test 2 (with one S-LDM subscribed to two brokers, with one client thread for each broker).

6.2. Data provision to other services scalability

The tests described in this Section were conducted to assess how well the service performs in terms of resource consumption while providing road context to other latency-critical MEC services. For these experiments, the same testbed setup as in the previous tests has been used, as depicted in Fig. 16. Specifically, on the *MS VM*, a custom REST server application was designed to accept the POST request coming from the S-LDM instance when a triggering condition is detected. The purpose of the custom REST server is to imitate the behavior of an actual centralized maneuver management service. With this aim, it has been configured to receive 10 POST requests, one every 50 ms, before sending a reply with the `rsp_type` field set to `"STOP"`.

For what concerns the tests outlined in this Section, we set up a gradual increment in the number of vehicles triggering context data transmission to the REST server: every 50 seconds, from one emulated vehicle up to 20 vehicles, resulting in a total test duration of 1000 seconds for each test. In particular, for this test, to simplify the data collection and generate controlled triggering conditions, we decided to set the CAM-based trigger to detect when a vehicle switches on its right-turn indicator. The test was conducted three times, with each iteration featuring 5 additional non-triggering vehicles placed within the 150-m context range of the vehicles requesting context data. It is worth mentioning that, in every test, the batch of non-triggering vehicles was generated simultaneously with the triggering ones and was configured to travel at the same speed to remain within the context range throughout the entire test period.

The results regarding the S-LDM CPU consumption, concerning both the number of triggering vehicles and the number of vehicles located within a given range around each triggering vehicle (i.e., within a context of 150 m), are presented in Fig. 17. It can be observed that the CPU consumption rises by 1000 CPU milliunits for every new vehicle triggering data transmission towards the REST server. This can be attributed to the S-LDM generating a new REST client thread for each service request, responsible for retrieving data from the database and sending POST requests with associated context information.

In addition, our findings reveal that the CPU consumption remains unaffected by the number of vehicles located inside the context transmitted to the other MEC services. However, as depicted in Fig. 17(a), once the total number of served vehicles approaches the previously mentioned maximum value of 430 vehicles (when relying on a single AMQP client and broker), as is the case for 15 triggering vehicles each with 25 additional non-triggering ones within context range, the CPU consumption decreases. This decrease is due to the delay in AMQP message reception, which leads to a decline in performance. It is worth mentioning that for the results showcased in Fig. 17, the total number of emulated vehicles depends on the number of non-triggering vehicles within context range. Indeed, for the case of 18 triggering vehicles each with 20 additional non-triggering ones, the total number of emulated vehicles is roughly the same as the case of 15 and 25 respectively, thus presenting the same behavior. This proves that the performance decrease in context data transmission to the REST server is not affected by the number of triggering vehicles nor by the number of vehicles in their context range, but only by the total number of served vehicles.

This behavior is a result of the triggering conditions being, as described for this test, dictated by the information contained in CAMs, specifically, by the status of exterior lights, found in the *lowFrequencyContainer*. Indeed, the *lowFrequencyContainer* is available on CAMs every 500 ms, coinciding with the duration needed to transmit all POST requests while supplying context information of a requesting vehicle. Consequently, as shown in Fig. 17(b), when message delay due to buffering occurs at the AMQP client, triggering conditions detected in messages are delayed as well, resulting in fewer REST client threads operating simultaneously at a given point in time.

Other than providing useful thresholds for a possible deployment strategy of the S-LDM, with multiple instances, the tests presented in this Section provide interesting insights on a possible smart scaling policy (as already hinted at in Section 3.3). The latter, in particular, could be based on the combination of road traffic (which should be possibly predicted to perform scaling actions proactively) and CPU usage. Although the number of vehicles triggering the transmission of road context increases the CPU consumption of the S-LDM, it has been proven that it does not affect the performance of the database, which is only limited by the total number of vehicles served by an AMQP client instance. Thus, future versions of the S-LDM may also consider improving the single instance performance by spawning multiple AMQP clients for each single broker subscription, in different threads, when an oncoming performance reduction is detected or predicted (e.g., when getting near the maximum number of vehicles manageable by a single client instance).

7. Conclusions and future work

We presented in this paper an innovative V2X 5G-enabled MEC service, capable of receiving messages from a large number of connected vehicles, both related to connected and non-connected road objects.

This service, called Server Local Dynamic Map (S-LDM), can pre-process and store the information about large portions of the roads (i.e., coverage areas of each S-LDM instance) in a highly efficient, centralized Local Dynamic Map [7], and provide a proper filtered data context to other services managing highly automated maneuvers. The latter can thus focus on the latency-critical control algorithm, avoiding being overloaded by message decoding and processing operations in presence of high vehicle densities and penetration rates.

The S-LDM can receive ETSI-compliant messages from one or more AMQP brokers, following the C-Roads standards. The messages of vehicles located in the coverage area are decoded, filtered and processed to be stored in a vehicle database, that represents one of the core components of our service. Then, in addition to real-time monitoring for road operators, pre-processed context data can be provided to other maneuver management services in an efficient way, either a single time or periodically. This happens either when the S-LDM itself detects special conditions on the road that require the intervention of centralized V2X services, or when such services query the S-LDM for information that may be needed for several different applications, from controlling traffic light phases to optimally manage an automated lane merge involving multiple vehicles.

After detailing the architecture and deployment options of the S-LDM, we evaluated our service both with in-lab tests, and through on-road experiments. The latter were performed as part of the 5G-CARMEN project, funded by the European Commission [2], involving two prototype Stellantis vehicles and the MEC platforms of three major European Mobile Network Operators (TIM of Italy, MTA of Austria and DTAG of Germany). This allowed us to test the S-LDM in both in-country and cross-border configurations, showing how it can always process each message and update the internal database in few tens of microseconds. Furthermore, the S-LDM has been designed to work as a cross-OEM service, and several compatibility tests have been performed with both Stellantis and BMW vehicles before the deployment for the road tests.

Our service was also tested with high message rates (i.e., 20 Hz), showing how it can steadily maintain a per-vehicle database update rate around 50 ms, up to 430 vehicles for each single instance, with a single AMQP client for AMQP broker configuration.

Finally, with respect to [5], we presented a novel extensive scalability analysis, providing interesting insights on how the S-LDM could be deployed and scaled in real-world scenarios. Specifically, we proved how the total number of manageable vehicles without any performance loss depends on the number of AMQP clients subscribed to each broker. This also provides useful information for an orchestration policy based on resource consumption and number of managed road objects.

As future work and research directions, we plan to:

- implement the S-LDM to Vehicle LDM interface;
- finalize the design and development of a smart AI/ML trigger and algorithm, as described in Section 3.2.5;
- design and implement an internal scaling mechanism which automatically detects performance bottlenecks and spawns new AMQP client when needed, dynamically adapting the coverage area of each client.

Concerning the second point, the S-LDM could play a crucial role in future AI-enabled automated vehicle MEC services. Use cases like DL task offloading from camera frames are emerging and they may require a centralized view of the road to find optimal solutions to the offloading problem, to be then communicated to the vehicles. Indeed, several works already analyze the problem of task offloading to the edge [51,52], that is emerging as a problem in vehicular networks. The S-LDM could thus provide two main advantages: (i) it could provide centralized offloading services with real-time information about road users, so that optimal offloading strategies can be employed and communicated to vehicles, and (ii) it can provide services that need to perform the offloaded tasks with any additional information they may need, offloading them from the need of decoding and filtering information, and leaving free resources for computationally heavy inference jobs.

In addition, the inclusion of an AI/ML trigger could further enhance the capability of the S-LDM to detect special conditions on the road, to provide in a timely manner the information needed to other maneuver management MEC services. For instance, an AI-based algorithm could analyze the current and historical position of vehicles to detect whether a lane merge is likely to occur. If this kind of maneuver is estimated to occur with a probability higher than a given threshold, the S-LDM could start processing and sending information about the involved vehicles to a maneuver management services, that will be able to timely react and provide proper maneuver recommendations to autonomous vehicles as soon as the actual maneuver is initiated.e

CRediT authorship contribution statement

C.M. Risma Carletti: Writing – review & editing, Writing – original draft, Validation, Software, Investigation. **F. Raviglione:** Writing – review & editing, Writing – original draft, Validation, Software, Investigation, Conceptualization. **C. Casetti:** Writing – review & editing, Writing – original draft, Validation, Software, Investigation. **F. Stoffella:** Writing – review & editing, Writing – original draft, Validation. **G.M. Yilma:** Writing – review & editing, Writing – original draft, Validation, Investigation. **F. Visintainer:** Validation, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The S-LDM service proposed in this article has been made available on GitHub under an open-source license.

Acknowledgements

This work has been partially supported by the European Union – Next Generation EU, through the National Center for Sustainable Mobility (Grant No. E13C22000980001). In addition, this work was Funded by the European Union through the project CONNECT under grant agreement no. 101069688. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] SAE J3016 2021-04 - Surface Vehicle Recommended Practice - Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles, SAE J3016 2021-04 (Revision of SAE J3016 2018-06), 2021, pp. 1–41.

[2] Home page - 5G-CARMEN, https://5gcarmen.eu/, 2022.

[3] ETSI, ETSI EN 302 637-2 V1.4.1 (2019-04) - Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service, Standard ETSI EN 302 637-2 V1.4.1, European Telecommunications Standards Institute, 2019.

[4] ETSI, ETSI TS 103 324 V2.1.1 (2023-06) - Intelligent Transport System (ITS); Vehicular Communications; Basic Set of Applications; Collective Perception Service; Release 2, Standard ETSI TS 103 324 V2.1.1, European Telecommunications Standards Institute, 2023.

[5] F. Raviglione, C.M.R. Carletti, C. Casetti, F. Stoffella, G.M. Yilma, F. Visintainer, S-ldm: server local dynamic map for vehicular enhanced collective perception, in: 2022 IEEE 95th Vehicular Technology Conference: (VTC2022-Spring), 2022, pp. 1–5.

[6] SAFESPOT D8.1.1 Final Report, in: SAFESPOT Final Rep—Public Version http://www.safespot-eu.org/documents/D8.1.1_Final_Report_-_Public_v1.0.pdf, 2010.

[7] ETSI, ETSI EN 302 895 V1.1.1 (2014-09) - Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Local Dynamic Map (LDM), Standard ETSI EN 302 895 V1.1.1, European Telecommunications Standards Institute, 2014.

[8] ETSI, ETSI EN 302 637-3 V1.3.1 (2019-04) - Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specifications of Decentralized Environmental Notification Basic Service, Standard ETSI EN 302 637-3 V1.3.1, European Telecommunications Standards Institute, 2019.

[9] A. Rauch, F. Klanner, K. Dietmayer, Analysis of v2x communication parameters for the development of a fusion architecture for cooperative perception systems, in: 2011 IEEE Intelligent Vehicles Symposium (IV), 2011, pp. 685–690.

[10] H.-j. Gunther, O. Trauer, L. Wolf, The potential of collective perception in vehicular ad-hoc networks, in: ITST 2015, 2015, pp. 1–5.

[11] P. Sondi, L. Rivoirard, M. Wahl, Performance evaluation of vehicle-to-vehicle communications for a collective perception application in vehicular ad hoc networks, in: IEEE PIMRC 2018, 2018, pp. 602–603.

[12] M. Shan, K. Narula, Y.F. Wong, S. Worrall, M. Khan, P. Alexander, E. Nebot, Demonstrations of cooperative perception: safety and robustness in connected and automated vehicle operations, Sensors 21 (2021).

[13] G. Thandavarayan, M. Sepulcre, J. Gozalvez, Redundancy mitigation in cooperative perception for connected and automated vehicles, in: 2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring), 2020, pp. 1–5.

[14] G. Volk, Q. Delooz, F.A. Schiegg, A. Von Bernuth, A. Festag, O. Bringmann, Towards realistic evaluation of collective perception for connected and automated driving, in: 2021 IEEE International Intelligent Transportation Systems Conference (ITSC), 2021, pp. 1049–1056.

[15] A. Olmos, F. Vázquez-Gallego, R. Sedar, V. Samoladas, F. Mira, J. Alonso-Zarate, in: An Automotive Cooperative Collision Avoidance Service Based on Mobile Edge Computing, 2019, pp. 601–607.

[16] G. Avino, P. Bande, P.A. Frangoudis, C. Vitale, C. Casetti, C.F. Chiasserini, K. Gebru, A. Ksentini, G. Zennaro, A mec-based extended virtual sensing for automotive services, IEEE Trans. Netw. Serv. Manag. 16 (2019) 1450–1463.

[17] R. Vilalta, R. Casellas, R. Sedar, F. Vázquez-Gallego, R. Martínez, S.K. Datta, M. Lefebvre, F. Gardes, J.-M. Odinot, J. Härri, J. Alonso-Zarate, R. Muñoz, Vehicular message exchange in cross-border scenarios using public cloud infrastructure, in: IEEE 35GWF 2020, 2020, pp. 251–256.

[18] Q. Liu, Jiang Han, T. Xie, B. Kim, Livemap: real-time dynamic map in automotive edge computing, https://arxiv.org/abs/2012.10252, 2020.

[19] M. García, I. Urbieta, M. Nieto, J. González de Mendibil, O. Otaegui, ildm: an interoperable graph-based local dynamic map, Vehicles 4 (2022) 42–59.

[20] F. de Ponte Müller, Survey on ranging sensors and cooperative techniques for relative positioning of vehicles, Sensors (2018).

[21] T. Mekki, I. Jabri, A. Rachedi, M. ben Jemaa, Vehicular cloud networks: challenges, architectures, and future directions, Veh. Commun. 9 (2017) 268–280.

[22] M.R. Hidayatullah, J.-C. Juang, Centralized and distributed control framework under homogeneous and heterogeneous platoon, IEEE Access 9 (2021) 49629–49648.

[23] The C-Roads Platform, https://www.c-roads.eu/platform.html, 2022.

[24] Apache, How does a queue compare to a topic, https://activemq.apache.org/how-does-a-queue-compare-to-a-topic, 2023.

[25] J. Schwartz, Bing maps tile system, https://docs.microsoft.com/en-us/bingmaps/articles/bing-maps-tile-system, 2021.

[26] 5G-CARMEN D2.2, 5G-CARMEN preliminary system architecture and interfaces specifications, 2021, https://5gcarmen.eu/wp-content/uploads/2021/10/D2.2-18May2021.pdf.

[27] ETSI, ETSI EN 302 636-4-1 V1.4.1 (2020-01) - Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 4: Geographical Addressing and Forwarding for Point-to-Point and Point-to-Multipoint Communications; Sub-Part 1: Media-Independent Functionality, Standard ETSI EN 302 636-4-1, European Telecommunications Standards Institute, 2020.

[28] ETSI, ETSI EN 302 636-5-1 V2.2.1 (2019-05) - Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 5: Transport Protocols; Sub-Part 1: Basic Transport Protocol, Standard ETSI EN 302 636-5-1, European Telecommunications Standards Institute, 2019.

[29] ETSI, AMQP v1.0, Standard ETSI TR 103 562 V2.1.1, European Telecommunications Standards Institute, 2019.

[30] C.-R.P.C. AustriaTech, The C-Roads Platform publishes harmonised C-ITS specifications for Europe, https://www.c-roads.eu/fileadmin/user_upload/media/Dokumente/Harmonised_specs_text.pdf, 2023.

[31] S. Hakak, T.R. Gadekallu, P.K.R. Maddikunta, S.P. Ramu, P. M, C. De Alwis, M. Liyanage, Autonomous vehicles in 5G and beyond: a survey, Veh. Commun. 39 (2023) 100551.

[32] 5G-CARMEN D3.3, Intermediate report on 5G technological enablers for CCAM, https://5gcarmen.eu/wp-content/uploads/2021/10/D3.3-July-2021.pdf, 2021.

[33] N. Slamnik-Krijestorac, G.M. Yilma, M. Liebsch, F.Z. Yousaf, J. Marquez-Barja, Collaborative orchestration of multi-domain edges from a connected, cooperative and automated mobility (ccam) perspective, IEEE Trans. Mob. Comput. (2021) 1.

[34] ETSI, ETSI TS 103 301 V2.1.1 (2021-03) - Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Facilities Layer Protocols and Communication Requirements for Infrastructure Services; Release 2, Standard ETSI TS 103 301 V2.1.1, European Telecommunications Standards Institute, 2021.

[35] GitHub - ms-van3t-devs/ms-van3t, https://github.com/vlm/asn1c.git, 2022.

[36] F. De Ponte Müller, Survey on ranging sensors and cooperative techniques for relative positioning of vehicles, Sensors 17 (2017).

[37] WhiteDB team, Home page - Whitedb, http://whitedb.org/, 2013.

[38] H. Shimada, A. Yamaguchi, H. Takada, K. Sato, Implementation and evaluation of local dynamic map in safety driving systems, J. Transp. Technol. 05 (2015) 102–112.

[39] M. Rapelli, C. Casetti, M. Sgarbi, A distributed v2v-based virtual traffic light system, in: 2020 International Conference on COMmunication Systems & NETworkS (COMSNETS), 2020, pp. 122–128.

[40] C. Quadri, V. Mancuso, M. Ajmone Marsan, G.P. Rossi, Platooning on the Edge, MSWiM ’20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 1–10.

[41] R. Xu, Y. Guo, X. Han, X. Xia, H. Xiang, J. Ma, Opencda: an Open Cooperative Driving Automation Framework Integrated with Co-Simulation, 2021.

[42] ETSI, ETSI TS 102 940 V2.1.1 (2021-07) - Intelligent Transport Systems (ITS); Security; ITS Communications Security Architecture and Security Management; Release 2, Technical Specification ETSI TS 102 940, European Telecommunications Standards Institute, 2021.

[43] ETSI, ETSI TS 103 097 V2.1.1 (2021-10) - Intelligent Transport Systems (ITS); Security; Security Header and Certificate Formats; Release 2, Technical Specification ETSI TS 103 097 V2.1.1, European Telecommunications Standards Institute, 2021.

[44] C-Roads, C-ITS Security & Governance, Technical Specifications Version 2.0.7, C-Roads, 2023.

[45] ETSI, ETSI TS 102 941 V2.2.1 (2022-11) - Intelligent Transport Systems (ITS); Security; Trust and Privacy Management; Release 2, Technical Specification ETSI TS 102 941, European Telecommunications Standards Institute, 2022.

[46] C-Roads, C-ITS IP Based Interface Profile, Technical Specifications Version 2.0.8, C-Roads, 2023.

[47] M. Malinverno, F. Raviglione, C. Casetti, C.-F. Chiasserini, J. Mangues-Bafalluy, M. Requena-Esteso, A multi-stack simulation framework for vehicular applications testing, in: ACM DIVANet 2020, ACM, New York, NY, USA, 2020, pp. 17–24.

[48] GitHub - ms-van3t-devs/ms-van3t, https://github.com/ms-van3t-devs/ms-van3t, 2022.

[49] 5G-CARMEN D5.3, 5G-CARMEN Final Pilot Report, 2021, https://5gcarmen.eu/wp-content/uploads/2022/10/5.3-October-2022.pdf.

[50] Home page - L3 Pilot, https://l3pilot.eu, 2022.

[51] F. Raviglione, C. Casetti, F. Restuccia, Edge-v: enabling vehicular edge intelligence in unlicensed spectrum bands, in: 2023 IEEE 97th Vehicular Technology Conference (VTC2023-Spring), 2023, pp. 1–5.

[52] B.-J. Qiu, C.-Y. Hsieh, J.-C. Chen, F. Dressler, DCOA: double-check offloading algorithm to road-side unit and vehicular micro-cloud in 5G networks, in: GLOBECOM 2020, 2020, pp. 1–6.

Carlos Mateo Risma Carletti is a Ph.D. student at the Department of Control and Computer Engineering in Politecnico di Torino. He is working on the evaluation of vehicular networking technologies for automated driving and the evaluation of different approaches for vehicular cooperative perception. He is also dealing with the development and testing of innovative vehicular micro-cloud applications.

Francesco Raviglione was born in Biella, Italy, in 1994. He received a B.Sc. degree in Computer Engineering from Politecnico di Torino, followed by an M.Sc. degree in Mechatronics Engineering, with a focus on automotive and embedded systems. He then got a Ph.D. cum laude in Electronics and Telecommunication Engineering in Politecnico di Torino, presenting a final thesis work titled “Open platforms for connected vehicles”. He is currently an assistant professor with time contract at the Department of Electronics and Telecommunications in Politecnico di Torino. He is currently working in the field of developing and evaluating platforms able to provide vehicular connectivity, on open source, customizable, solutions for wireless networking use cases, and on network measurements and performance assessment.

Claudio Casetti is a full professor at the Department of Control and Computer Engineering, Politecnico di Torino. He has published more than 200 papers in peer-refereed international journals and conferences on the following topics: vehicular networks, 5G networks, transport and network protocols in wired networks, and IEEE 802.11 WLAN.

Filippo Stoffella is working at Stellantis-CRF. He received a M.Sc. degree in Telecommunication Engineering in 2018 at the University of Trento and then a Second Level master in Autonomous Driving and Enabling Technology in 2020. Since 2018 he has been working at Stellantis-CRF Trento branch focusing on Vehicle to Vehicle and Vehicle-to-Infrastructure communication technology (V2X).

Girma Mamuye Yilma received his M.Sc. degree in Telecommunications Engineering in 2016 from the University of Trento, Italy. Currently, he is a Senior Research Engineer at NEC Laboratories Europe. His main research interest lie in the area of Management and Orchestration for NFV, MEC and O-RAN.

Filippo Visintainer, M.Sc. in Physics, is a Technical Fellow of Cooperative Systems at Stellantis-CRF Trento branch, currently working as Project Leader within the automated driving software development department. He has worked in R&D on Automotive Sensing, Intelligent Transportation Systems and Connected and Automated Driving functions, focusing on Vehicle to Vehicle and Vehicle-to-Infrastructure communication technology (V2X).