IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 26 May 2023, accepted 9 June 2023, date of publication 14 June 2023, date of current version 20 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3286023

APPLIED RESEARCH

Platooning-as-a-Service in a Multi-Operator ETSI MEC Environment

GIOVANNI NARDINI[1,2], (Member, IEEE), ALESSANDRO NOFERI[1], AND GIOVANNI STEA[1]
[1]Dipartimento di Ingegneria dell'Informazione, University of Pisa, 56122 Pisa, Italy
[2]Polo Universitario "Sistemi Logistici," University of Pisa, 57128 Livorno, Italy

Corresponding author: Giovanni Nardini (giovanni.nardini@unipi.it)

This work was supported in part by the Italian Ministry of Education and Research (MIUR) in the Framework of the Future-Oriented REsearch LABoratory (FoReLab) Project (Departments of Excellence); in part by the MOST—Sustainable Mobility National Research Center through the European Union Next-GenerationEU through Piano Nazionale di Ripresa e Resilienza (PNRR)—MISSIONE 4 Componente 2, Investimento 1.4—D.D. 1033 17/06/2022 (Spoke 10: Logistics and Goods) under Grant CN00000023; and in part by the Center for Logistic Systems, University of Pisa.

ABSTRACT Multi-access Edge Computing (MEC) is expected to support platooning of vehicles, by running the control logic that computes the acceleration values of the vehicles based on their position and speed. Connectivity between vehicles and the MEC system is likely to be realized through the mobile network. However, platoons will be composed of vehicles that are customers of different co-located mobile operators, hence – in all likelihood – will also be customers of different MEC systems. In this paper, we devise and evaluate an architectural framework that realizes Platooning-as-a-Service (PlaaS) in a multi-operator MEC environment, compliant with the ETSI MEC standard: we describe the entities involved and their interactions, we release an open-source proof-of-concept implementation for the popular Simu5G simulator, and we evaluate the effect of our framework – and especially its latencies – on platoon stability.

INDEX TERMS Edge computing, MEC, multi-operator, platooning, Simu5G.

I. INTRODUCTION

The deployment of ubiquitous high-speed mobile access, started with 4G and continuing with 5G, has paved the way for distributed applications having real-time requirements, such as Cooperative Adaptive Cruise Control (CACC)-based platooning. In the latter, vehicles organize themselves so as to form a line traveling at the same speed on a lane, keeping a near-constant inter-vehicle distance. Near-instant communication capabilities enable centralized control strategies: a centralized controller, knowing the state of all vehicles in the platoon, can make optimal control decisions, hence improving the application performance (e.g., reducing inter-vehicle distance for the same cruise speed, or further reducing accelerations/decelerations of vehicles, thus saving fuel).

Parallel to the above, there has been a growing trend towards deployment of computing infrastructures as near as possible to the end users, to reduce round-trip latency and leverage context information. Multi-access Edge Computing (MEC), being standardized by the European Telecommunications Standards Institute (ETSI), is probably the best example of a computing infrastructure, independent of – but synergic with – the underlying communication infrastructure, that runs applications on demand on behalf of users. While ETSI MEC is designed to be independent of the access technology, hence its name, it is clear that its deployment will chiefly occur in conjunction with 4G, 5G and beyond-5G mobile networks. The ETSI MEC infrastructure, in fact, is likely to be owned by the very providers that own the mobile network infrastructure as well.

In such an environment, a MEC-enabled mobile network is the ideal candidate to host CACC-based platooning: both the maneuver and the longitudinal platooning controllers can run as applications in the MEC system, receiving updates (e.g., position and speed) from the participating vehicles, and issuing cruise-control commands to them. This makes sense on several accounts: it enables complex algorithms to be used (which might otherwise be impossible to run on a

The associate editor coordinating the review of this manuscript and approving it for publication was Vlad Diaconita.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

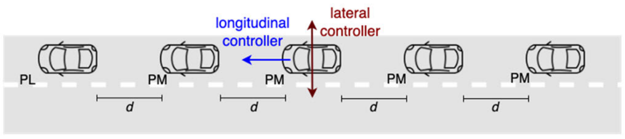


FIGURE 1. Platoon with one PL and four PMs, maintaining a constant inter-vehicle distance d.

single car, e.g. a platoon leader, due to resource constraints) thanks to the large amount of computing resources available; it enhances security, by allowing vehicles to leverage the authentication, security and privacy features of a trusted computing environment; it relieves individual vehicles of the burden of running cooperative algorithms themselves, with all the problems that this would entail (e.g., interoperability, software version clashes, updates, etc.).

MEC-enabled platooning has already been suggested in [9], [10], [11], [12], [13], and [14]. These works assess the impact of the network (e.g., delay, losses) on the performance (e.g., inter-vehicle distance, stability) of a given *running platoon control logic*, with a static number of platoon members. Moreover, they all assume that the control application runs on a *single MEC system*. In this paper, we take a radically different perspective: our aim is to design a Platooning-as-a-Service (PlaaS) *service architecture*, compliant with the ETSI MEC standard, to be used by subscribers of multiple mobile network operators. In other words, we devise a service providing mechanisms for discovery, subscription, dynamic join/leave, etc., to run in an ETSI MEC framework. We describe an architecture, comprising placement, behavior and interaction of the various functions, that allows this service to be used by vehicles, and we provide a reference open-source implementation for the popular Simu5G simulator [1], [27], that researchers and practitioners can use, e.g., to test their own control logic in a lifelike environment, comprising both 3GPP-compliant 5G network access and an ETSI-compliant MEC architecture. We explicitly design our PlaaS framework so that it can run *across different MEC systems*. Current mobile deployments show that mobile networks, owned by different operators, are co-located. Under the assumption of single ownership of both network and MEC infrastructure, it would be overly constraining to limit platooning services to the subscribers of the same mobile network operator. Just as subscribers of different networks can communicate with each other – because network owners have peering agreements, composed of both a technical and a business side – they should be allowed to run MEC-based cooperative distributed applications such as platooning. To this aim, we leverage the ongoing discussion on ETSI *MEC federation* [5] as a way for different MEC systems to enable shared use of MEC services and MEC apps. There are unique challenges to designing PlaaS in a multi-operator context, such as defining the required entities and their respective interfaces, collecting information (e.g., positions) of vehicles belonging to different MEC system owners' domain, and keeping the status of platoons updated in all the involved MEC systems. Moreover, multi-operator PlaaS presents unique challenges from a performance standpoint, such as the impact of the delay on the inter-federation link and the different computational load on entities of different MEC systems, – which could not be addressed in works [9], [10], [11], [12], [13], [14], dealing with single MEC systems. We highlight these challenges, explore their impact, and propose ways to overcome them. To the best of our knowledge, this is the first work addressing the above concerns.

The main contributions of this paper are:

- we design, for the first time, a *complete* PlaaS framework, defining roles and interactions, to be used in a multi-operator, ETSI-MEC-endowed mobile network;
- we prove its feasibility by evaluating it in scenarios that include all the components that can influence the performance, i.e., ETSI-compliant MEC architecture, computation overhead, 5G network transport, vehicular mobility;
- we integrate all our code within Simu5G and release it publicly. This allows researchers to evaluate their platooning algorithms (e.g., CACC, lane change, join/leave maneuvers, etc.) in the above-described environment, virtually without any implementation effort.

The rest of this paper is organized as follows: Section II reports background information, whereas Section III discusses the related works. We present our framework in Section IV, and evaluate it in Section V. Section VI draws conclusions and highlights directions for future work.

II. BACKGROUND

This section provides an overview of platooning and ETSI MEC.

A. PLATOONING

A platoon is a convoy of vehicles that travel on the same lane. With reference to FIGURE 1, the head-of-line vehicle is called *platoon leader* (PL), and the other *platoon members* (PMs) strive to keep a constant (and possibly short) distance from the vehicle ahead of them, adjusting their speed. This is realized in practice by *controlling* the PMs with two components, namely the *longitudinal* and *lateral* controllers [19]. The longitudinal controller computes the acceleration commands for the PMs in order to let them keep a constant, target distance with respect to the preceding vehicle, whereas the lateral controller issues the commands that allows the vehicles to keep the current lane and stay in-line with the preceding one. The higher-level commands generated by the controllers are translated into lower-level commands (i.e., throttle or brake) for the actuation system of the PMs. The objective is to maintain *platoon stability*, i.e., to keep the inter-vehicle distance constant, while minimizing fuel consumption. The target inter-vehicle distance will be a function of the cruising speed. A shorter distance for the same speed will reduce air friction, hence fuel consumption, as well as road occupancy.

Adaptive Cruise Control (ACC) systems are based on PMs adjusting their own speed based on onboard sensors such as radar and lidars. However, ACC is affected by the so-called *string instability* problem, i.e., errors in the inter-vehicle distances are amplified towards the tail of the platoon due to the unavoidable actuation delay required by each PM to detect the input from sensors, compute the new acceleration value and actuate the command [20]. Cooperative ACC (CACC) improves ACC by enabling information exchange among the vehicles of the platoon, such as their position, speed and acceleration. As a result, the information about the status of the vehicles can be (almost) instantly shared among the vehicles themselves, and the platoon control logic can promptly adapt their speed/acceleration, hence minimizing shockwave effects. CACC requires network support in order to let the vehicles *cooperate* towards the above objective. During the past few years, both IEEE 802.11p and 3GPP cellular networks have been considered as viable technologies to support platoon control [22], [23]. Cellular Vehicle-to-everything (C-V2X) has been standardized by 3GPP since Release 14 (LTE), and is now an integral part of the 5G specifications. With respect to IEEE 802.11p, it has the distinctive advantage that it allows applications to run on licensed spectrum, with ubiquitous coverage, controlled interference and reserved resources. As far as the control logic is concerned, it can be executed in either a distributed or a centralized manner [21]. In a distributed way, each PM runs its own control logic – i.e., both the longitudinal and lateral controllers –, which is usually based on its own position, speed, acceleration and that of the preceding vehicle. With centralized control, all the PM information (e.g., current speed and distance from the preceding vehicle) is sent by PMs to a central controller, which computes commands for all of them and sends them back. The central controller can reside on the PL or on an external host. On one hand, centralized control can take advantage of the global knowledge about the status of both the PL and all the PMs to optimize the speed and acceleration of all the vehicles at once, hence reducing convergence time. On the other hand, all the information needs to be conveyed to a single entity, which can make the required latency non-negligible and affect the freshness of the information [21]. However, more effective communication technologies – such as 5G (and beyond) – can easily overcome the above limitation. Moreover, the advent of MEC makes it easier to onboard the centralized controller at the edge of the network, e.g., as a MEC app [9], further reducing the communication latency.

Besides controlling the platoon stability, a complete platooning framework must take care of all the operations required to form and manage a platoon, namely discovery, join and leave a platoon, merge and split existing platoons. In particular, join/leave and merge/split operations require to perform maneuvers involving both longitudinal and lateral controllers [24]. For example, when a PM wants to leave a platoon, the lateral controller of the PM must be instructed to make the PM change lane and, if the latter was in the center of the platoon, the longitudinal controller of the remaining PMs must be commanded such that they can close the gap left by the vehicle.

B. ETSI MULTI-ACCESS EDGE COMPUTING

MEC is a paradigm that allows network users to exploit applications running in a virtualized environment at the edge of the network, hence it allows end users of the network to leverage the services provided by such applications with smaller latency compared to, e.g., a remote cloud deployment. The Industry Specification Group of ETSI has standardized the MEC reference framework [2], by specifying its functional entities, their interactions and the set of APIs they provide. With reference to FIGURE 2, an ETSI MEC system includes a *MEC host level* and a *MEC systemlevel*.

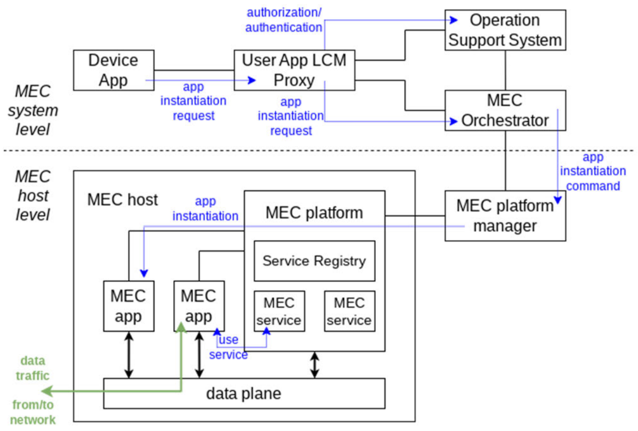


FIGURE 2. Main entities of the ETSI-MEC reference framework.

The MEC host level includes the virtualization infrastructure running the MEC apps and the MEC platform. The latter provides a set of MEC services that can be exploited by MEC apps to fulfill their objective. ETSI has specified several standard APIs for relevant MEC services such as, e.g., the Radio Network Information Service (RNIS) [3] and the Location Service (LS) [4], and new ones are currently being standardized. MEC apps can offer services themselves: such applications are called *producers* and expose an API to other MEC apps, called *consumers*. Whenever a MEC app wants to consume the functionalities provided by either a MEC service or a producer MEC app, it obtains the location of that entity from the Service Registry. In fact, MEC apps and MEC services can be hosted by different MEC hosts under the domain of a same MEC system. Once a MEC app knows the (local or remote) endpoint of a MEC service or MEC app, they can exchange data through the operator's core network. Note that data-plane communications are not standardized by ETSI.

The MEC system level maintains a global view of the MEC hosts in a MEC system and their resources, and handles the lifecycle of MEC apps, i.e., instantiation, relocation and termination. Its main element is the MEC orchestrator: after a MEC app instantiation request has been authenticated and authorized by the Operation Support System (OSS), the

MEC orchestrator selects an appropriate MEC host – based on application requirements (e.g., latency) and availability of resources (e.g., CPU, memory, mass storage) and MEC services – and instructs its MEC platform manager to deploy the container or virtual machine that will run the MEC app.

A MEC app is made available to a MEC system through a bundle of files provided by the application provider to the MEC system, which uses them to instantiate the application in a MEC host upon the request from a User Equipment (UE) or the MEC system owner via management interfaces.

At the UE side, the UE app is the local application that performs the data-plane communication with a MEC app. However, when the UE wants to discover, instantiate or terminate a MEC app, it must interact with the Device App, a MEC system-level entity that interfaces with the User Application Lifecycle Management Proxy (UALCMP). The latter is responsible for forwarding lifecycle-related requests coming from the UE to the MEC orchestrator.

A MEC system includes several MEC hosts under the control of the same MEC orchestrator, and it is deployed by the MEC system owner. Although not mandatory, the MEC system owner is often the operator that owns the network infrastructure too. Obviously, the geographical area covered by MEC systems from different owners can overlap, similar to what happens with mobile networks of different operators. This originated the need to collaborate for MEC systems belonging to different MEC owners. As a consequence, ETSI MEC started considering *MEC federation* as a way for different MEC systems to enable shared use of MEC services and MEC apps [5]. V2X services have been identified as major use cases for MEC federations, since vehicular applications are supposed to work in heterogeneous environments involving vehicles from different manufacturers, connected to different network operators. The interactions between federated MEC systems are illustrated in FIGURE 3. A Federation Manager is included next to the MEC orchestrator in the MEC system level, which interfaces with Federation Managers of other MEC systems. The Federation Manager is responsible for discovering federated MEC systems and the MEC services they offer – including performing authentication – as well as for establishing data-plane communication between MEC apps/services in different MEC systems. Note that the actual data exchange between MEC apps/services *does not* involve the Federation Managers and is *not* standardized by ETSI. This framework allows MEC apps to consume the services provided by a MEC app/service in a different MEC system (e.g., of another MEC system owner).

III. RELATED WORKS

Several recent research works have investigated the synergy between platooning applications and the MEC infrastructure.

Some papers [6] and [7] suggest to employ platoons of vehicles as distributed computing infrastructures, whose processing capabilities can be enhanced by MEC. Work [6] proposes a scheme to run collaborative applications, distributed among the platoon members, where the tasks composing the

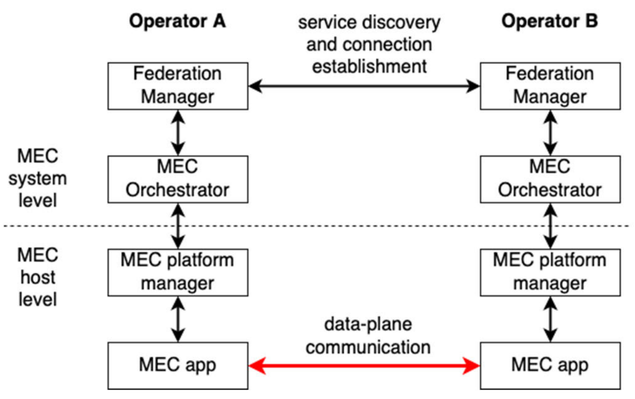


FIGURE 3. MEC Federation.

application can be offloaded to a MEC host in order to meet latency constraints. The selection of the tasks to be offloaded is designed to minimize the cost of purchasing MEC computing resources. Reference [7] tackles the same problem, with the objective of minimizing the energy consumed by the vehicles. Although the above works show how members of a platoon and MEC can interact to offer new solutions, MEC is not regarded as a technology to *create* and *control* platoons, which instead rely on distributed control and inter-vehicle communications. In [15] a mobile edge platoon cloud (MEPC) is proposed, where idle resources at the PMs can be allocated to tasks from surrounding vehicles. The authors models the interactions between the vehicle generating the task and the PMs as a Stackelberg game, and propose to use the blockchain technology to protect the privacy of exchanged data. According to [8], the above works can be included in the broader category of the offloading mechanisms for V2X services.

Other works [9], [10], [11], [12], [13], [14], instead, advocate using MEC as an infrastructure, e.g., to run platooning control algorithms. Our previous paper [9] presented a preliminary platooning framework based on an abstraction of MEC, where the latter runs the logic to form platoons and the centralized longitudinal controller. The focus of that paper was to prove the feasibility of a MEC-enabled platooning framework via realistic mobile network simulations. Work [10] presents a MEC-based platoon control architecture and provides a thorough analysis of the impact of latency and packet loss on the platoon stability. Although the work shows the feasibility of edge-controlled platooning in different network conditions, a description of the protocols involved to create and maintain the platoon is missing. Moreover, results were obtained by modeling network latencies as probability distributions, and a realistic radio network scenario was not actually simulated. In [11], authors propose a two-layer framework aiming at coordinating multiple platoons, i.e. forming a multi-platoon by coordinating a chain of independent platoons. That paper evaluates multi-platoon stability, as well as the communication overhead needed to control a multi-platoon. In this case, too, the focus of the paper is on the logic required to control the platoon, and details about MEC

TABLE 1. Comparison of related works.

Ref.	Subject
[6],[7]	Platoon as distributed computing environment, task offloading to MEC.
[15]	Platoon as mobile edge cloud architecture for task offloading from surrounding vehicles.
[9]	MEC runs centralized longitudinal controller only.
[10]	MEC-based platoon control architecture.
[11]	MEC-based Control logic for coordinating/merging platoons.
[12]	Platoon control framework based on MEC with Docker.
[13]	MEC-based leader-follower platoon control.
[14]	Three-tier cloud-based architecture for platoon control.
[21]	Evaluates platoon stability under uncertainty for both distributed and centralized control strategies.
[25]	Service continuity of MEC-based V2X applications in a multi-operator scenario.
[16]	MEC-based wireless towing in emergency scenario.
[17]	Platoon controller migration in a MEC-based environment.
[18]	Contention mechanisms for 802.11p-based platoons, with controller hosted at the edge of the network.
This paper	ETSI-MEC compliant software framework. Allows discovery, join, leave, CACC, multiple platoons, with arbitrary algorithms in a multi/operator context.

protocols required to manage the entire lifecycle of a platooning service (e.g., join of a new vehicle) are not described. Like in [10], realistic simulations of the radio network were not performed. Reference [12] presents a platoon control framework implemented by Docker containers in the MEC. In the proposed framework, information coming from the platoon members is conveyed to the platoon leader, which forwards it to its counterpart application at the MEC, called Virtual Platoon App. The latter may complement the information coming from the platoon leader with new data, e.g. from road-side infrastructure. Such information is then passed to another MEC application, namely the Virtual Control App, acting as the longitudinal controller for all the vehicles of the platoon. The output of the latter is then sent to the platoon leader, which in turn forwards it to the other vehicles through V2X communications. Although the proposed framework fits well the ETSI MEC architecture, support for dynamic addition/removal of platoon members is not considered, and the coexistence of multiple platoons is not discussed. Moreover, by routing all the communications through the platoon leader, the round-trip latency between a platoon member and the controller may be non-negligible. In [13], coordination of a platoon is delegated to the platoon leader, which gathers platoon members' information, computes their updated acceleration values, and notifies them to other vehicles. The MEC system is only used by the platoon leader to retrieve information about the surrounding environment, like road traffic, to deal with possible shockwave effects due to slower preceding vehicles on the road. In [14], the authors propose to split the functional blocks of a platoon control system into a three-layer architecture involving vehicles, road-side units and the cloud. The paper does not discuss the communication protocols within the framework of ETSI MEC and does not evaluate the performance of the proposed architecture. Work [16] proposes a wireless towing service for emergency scenarios and evaluates it on a 5G network endowed with MEC, simulated with Simu5G. MEC is used to host the app that performs the various services (including AI-based driver selection). The control logic manages a single towed vehicle. Paper [17] assumes MEC-based cruise control and proposes a Q-learning strategy to decide when to migrate the control application in order to enhance performance. In [18], the authors consider the problem of regulating the contention window for platoons whose PMs use 802.11p to share data with a edge-based controller.

None of the above works consider the case of platoons involving vehicles from different network operators, which is instead a common situation in practical scenarios. Thus, the solutions proposed in the existing literature neglect the peculiarities that a multi-operator MEC environment introduces, in terms of both architectural requirements and performance. To the best of our knowledge, the work in [25] is the only work that considers a multi-operator environment. It focuses on the issue of service continuity when a generic MEC-based V2X application needs to be moved from one MEC host to another, possibly belonging to another operator. Platooning is taken as one exemplary use case for this issue. Although the authors consider the problem of synchronizing information among different MEC host during mobility, they do not consider the architectural implications of vehicles in the same platoon being connected to different network operators and different MEC systems. Also, solutions are proposed only theoretically, and details are not discussed.

In light of the above analysis (summarized in TABLE 1), we can state that our work is the first to design functions and protocols to run PlaaS, including not only cruise control, but also all the required functionalities to discover, join, and leave platoons; our PlaaS framework is fully compliant with the ETSI-MEC standard; it allows a service operator to run platoons consisting of vehicles whose MEC apps run on

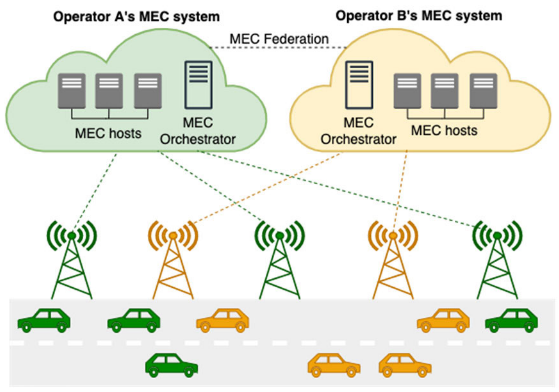


FIGURE 4. System model.

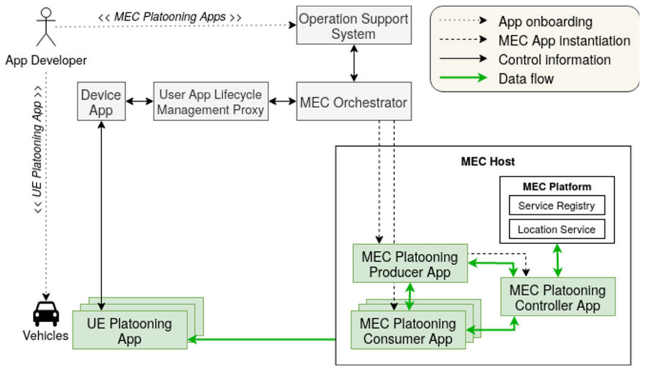


FIGURE 5. Proposed MEC-based platooning framework.

different federated MEC systems. Our PlaaS is presented in Section IV, is proved feasible in Section V, after evaluation in scenarios that model all the relevant components (network transit, computational load, vehicular mobility, physical constraints).

IV. PROPOSED FRAMEWORK

In this section, we describe our PlaaS framework. We begin by stating the working hypotheses, and then discuss PlaaS. The latter has been designed from the start having in mind its operation in a multi-operator scenario. However, for ease of exposition, we describe it first in the framework of a single operator, and then in a multi-operator context, involving federation of ETSI MEC systems.

A. SYSTEM MODEL

With reference to FIGURE 4, we consider a road environment (e.g., a highway) where vehicles may form several platoons. Network coverage along the road is provided by multiple mobile network operators, whose base stations are deployed at the side of the road, possibly at co-located sites. Vehicles are equipped with a transceiver module for cellular connectivity, and they are subscribed to the mobile network of one operator. This means that a vehicle is a UE that can communicate *only* with the base stations of the mobile network operator it is subscribed to.[1]

We assume that each operator owns its MEC system, which is compliant with the ETSI MEC framework described in Section II-B. Each MEC system is composed of multiple MEC hosts, interconnected via the operator's core network. Applications running on a vehicle can communicate with MEC apps located on the MEC hosts belonging to the operator the vehicle is connected to. In order to provide a multi-operator platooning service, MEC systems belonging to different operators form a MEC Federation as envisaged by the ETSI standard described in Section II-B.

[1]We make no hypotheses as to whether the underlying cellular network is 4G or 5G, since both can be used for the purpose of this paper – at least in a line of principle. In Section V we will setup 5G scenarios for the evaluation, for the sake of concreteness.

Our goal is to define an ETSI MEC-based PlaaS framework that can be leveraged by vehicles to discover which platoons are available in their neighborhood and possibly join them, in order to have their cruise speed regulated accordingly. Our PlaaS framework must be able to run on different, federated MEC systems, so that platoons may be formed by vehicles connected as UEs of different network operators. The above framework is realized by a software package that includes applications to be run at both the vehicle and MEC sides. UE applications are installed on the vehicle, whereas MEC applications will be onboarded to all the involved MEC systems. Designing the applications according to the ETSI MEC specifications ensures that standard APIs are employed, hence portability of the MEC apps is guaranteed, as well as the interoperability between MEC apps and services running on different MEC systems.

In the remainder of this section, we first describe the main components of the framework and their interactions in the simpler case of a single network operator/MEC system. Then, we generalize it to the multi-operator case.

B. MAIN COMPONENTS OF THE FRAMEWORK

FIGURE 5 shows the main logical entities of a MEC system involved in our PlaaS framework. Green boxes represent the applications that realize the service, whereas the other boxes are standard ETSI MEC entities that interact with the applications.

The *UE Platooning App* runs on the vehicles and interacts with the MEC side of the PlaaS framework, by sending requests to discover, join or leave a platoon and receiving inputs from the platoon controller, such as the desired acceleration value to maintain the desired inter-vehicle distance and speed. Locally, the UE Platooning App needs to interface with the lower-level controller of the vehicle, which actuates the control inputs received from the platooning controller.

On the MEC side, the PlaaS is realized through a triple of MEC apps, namely the *MEC Platooning Consumer App*, the *MEC Platooning Producer App* and the *MEC Platooning Controller App,* hereafter *Consumer App*, *Producer App* and *Controller App,* respectively, for short.

The Consumer App acts as the counterpart of the UE Platooning App on the MEC, and interacts with the Producer

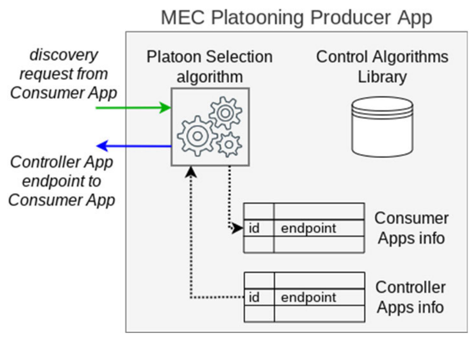

MEC Platooning Producer App
discovery request from Consumer App
Controller App endpoint to Consumer App
Platoon Selection algorithm
Control Algorithms Library
id
endpoint
Consumer Apps info
Controller Apps info

FIGURE 6. Internal structure of the MEC Platooning Producer App.

```
platoonSelectionAlgorithm(pos, dir)
begin:
   candidate_platoon = 0
   candidate_dist = +infty
   foreach running platoon p
      dir[p] = get_direction(p)
      if (dir == dir[p])
         pos[p] = get_last_member_pos(p)
         dist = compute_distance(pos, pos[p])
         if (dist < candidate_dist)
            candidate_dist = dist
            candidate_platoon = p
         endif
      endif
   endfor
   return candidate_platoon
end
```

LISTING 1. Pseudocode of an examplary platoon selection algorithm.

App and – when associated to a platoon – with the Controller App. The Consumer App keeps track of the current state of the vehicle (e.g., discovery request forwarded, waiting to join a platoon, maneuvering, cruising, etc.), conveys requests coming from the UE Platooning App (e.g., requests to discover or leave a platoon) to the Producer App, and forwards control inputs from the Controller App to the UE Platooning App.

The Producer App is the core of the PlaaS framework. Its main task is to maintain the list of Consumer Apps (hence, of UEs) that are using the PlaaS and to track the status of all the active platoons, e.g. which vehicles are associated to which platoon. Its internal structure is shown in FIGURE 6. It stores the relevant data structures that maintain the endpoints (IP address/port) of both the subscribed Consumer Apps and the Controller Apps that manage the platoons, and it handles the platoon discovery requests from the Consumer Apps by identifying the platoon the latter may request to join. To accomplish this, whenever a discovery request is received, the Producer App runs a platoon selection algorithm that selects the most suitable platoon for the requesting vehicle among the active ones. The selection algorithm can factor in all the information available, such as the current position of both the vehicle and the platoons, the state of already active platoons (e.g., their length, direction or target speed), context data (road layout or regulations), user preference and so on. The framework is designed to be flexible enough to allow one to add different platoon selection algorithms and switch between them whenever needed. The design of a platoon selection algorithm is outside the scope of this paper. Nonetheless, for the sake of concreteness, we briefly describe a sample algorithm that can be used. With reference to the pseudocode reported in LISTING 1, whenever the Producer App receives a discovery request, it scans the list of currently running Controller Apps and, for each of them, obtains the direction of the corresponding platoon. If the direction is the same of the car that issued the discovery request, the distance between the car and platoon member at the tail of the platoon is computed. Finally, the algorithm selects the platoon with the smallest distance.

Each active platoon is managed by one Controller App. The internal architecture of the latter is shown in FIGURE 7.

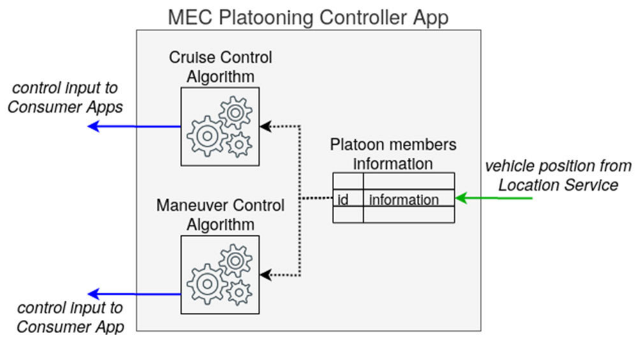

MEC Platooning Controller App
control input to Consumer Apps
Cruise Control Algorithm
Platoon members information
id
information
vehicle position from Location Service
Maneuver Control Algorithm
control input to Consumer App

FIGURE 7. Internal structure of the MEC Platooning Controller App.

It includes a Cruise Control Algorithm and a Maneuver Control Algorithm. Both algorithms generate inputs for both the longitudinal and lateral controllers of the vehicles. The former periodically computes the next control inputs for all the platoon members (i.e., new acceleration values to be enforced) in order to maintain the platoon stability and keep the vehicles in the same lane, whereas the latter runs upon the reception of a join (leave) request from a Consumer App and issues the control inputs to allow the corresponding vehicle to approach (detach from) the platoon and change lane, if needed. The control algorithms exploit information such as the Consumer App identifier and the vehicle's last available position, speed and acceleration. Such information is obtained by querying the LS of the MEC platform, hence saving the communication latency and overhead that would otherwise be required to obtain the same information directly from the vehicle through the radio interface. It is important to note that different Controller Apps are independent of each other, hence they can run *different* control algorithms. This gives the PlaaS framework the flexibility to decide which control algorithm suits better which platoon: for example, it might be the case that platoons with different lengths, or requirements in terms of target speed or inter-vehicle distance, or onboard processing capabilities of the vehicles, require different control algorithms. Thus, the Producer App

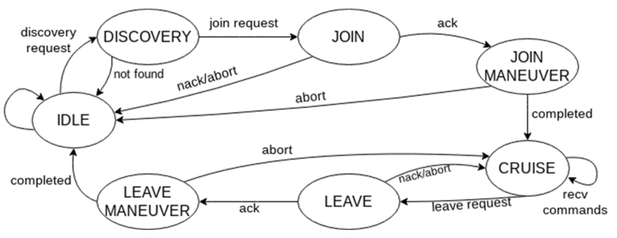


FIGURE 8. MEC Consumer App state diagram.

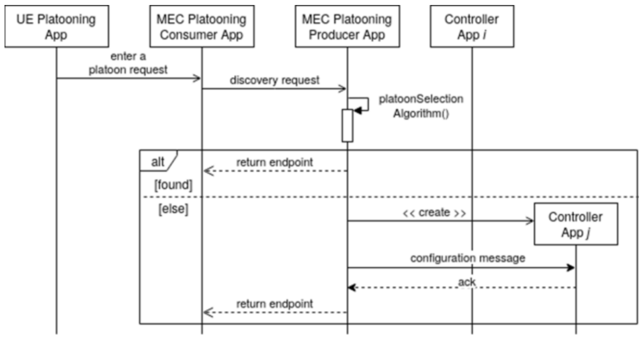


FIGURE 9. Discovery protocol.

includes a library of available control algorithms and can select the most suitable one for each Controller App instance during the instantiation of the latter. The Producer App can also change the control algorithm enforced by one platoon dynamically, if needed, without affecting the other platoons. New algorithms can be easily added to the library anytime.

As shown in FIGURE 5, the UE Platooning App is made available to vehicle manufacturers, which onboard it to their vehicles. The Consumer, Producer and Controller Apps, instead, are provided to the MEC system owner as virtual machines or container images, which will then be instantiated on the MEC virtualized infrastructure according to the standard mechanisms discussed in Section II-B. In particular, the Producer App will be instantiated by the MEC system owner through direct interaction with the OSS, whereas the instantiation of the Consumer App will be dynamically requested by the UE Platooning App via the Device App and the UALCMP, upon an explicit request from the driver of the vehicle, i.e., when she decides to exploit the services offered by the PlaaS. The instantiation of a Controller App, instead, is triggered by the Producer App when a new discovery request from a Consumer App issues the creation of a new platoon.

Note that software updates for the UE Platooning App can be issued over-the-air, without requiring the vehicle to be taken to the car-maker maintenance centers. Likewise, updates to MEC apps (e.g., providing new control algorithms) can easily be done at the MEC hosts without service interruption and without involving the vehicles.

C. COMMUNICATION PROTOCOLS ENABLING THE FRAMEWORK

We now describe the communication protocols implemented by the entities presented above to realize the PlaaS framework. To do this, we take the perspective of the Consumer App and consider its state machine, shown in FIGURE 8. We recall that once a Consumer App has been created, it completely represents a vehicle in the MEC system, and interacts with the Producer App on its behalf.

Initially, once the UE has instantiated the Consumer App, the latter is in the IDLE state, meaning that the vehicle is not a member of any platoon. It remains in such state until the vehicle (e.g., the driver) requests to access the platooning service. The latter operation brings the Consumer App in the DISCOVERY state, where it asks the Producer Apps to select the most suitable platoon to join. If a suitable platoon is found, the Consumer App receives the endpoint of the Controller App managing such platoon and sends a *join* request to it. Since we assume that only one vehicle at a time can be involved in a join/leave maneuver, join requests are queued at the Producer App until no other join/leave maneuver were previously queued. Then, once the vehicle is selected to join in the platoon, the Consumer App receives a notification and enters the JOIN MANEUVER state, where it receives the commands to approach the selected platoon and take on the desired position (either the head, tail or the middle of the platoon). If the join maneuver completes successfully, the Consumer App moves to the CRUISE state. This represents the steady state, where the vehicle is instructed by the Cruise Control Algorithm in the Controller App with the control inputs necessary to maintain the desired position and speed within the platoon. When the vehicle is not interested anymore in being a member of the platoon (e.g., the driver wants to change direction), the Consumer App issues a *leave* request and enters the LEAVE state, while the request is enqueued at the Producer App. Once the Controller App has acknowledged the request (i.e., the request has been selected from the leave queue), the Consumer App transitions to the LEAVE MANEUVER state, during which the vehicle performs the action to abandon the platoon. Upon completion, the Consumer App gets back to IDLE and it is ready to join a new platoon if needed. In our framework, leave requests have precedence over queued join requests (because a leave operation may impinge on already scheduled join maneuvers, by changing the state of the platoon itself).

In the following, we describe in detail the message exchange among the entities when the Consumer App is in DISCOVERY, JOIN MANEUVER, CRUISE and LEAVE MANEUVER states. The actions that vehicles need to perform when executing the join and leave maneuvers are orthogonal to the PlaaS framework, hence outside the scope of this paper. Interested reader can find details about these in [26].

FIGURE 9 shows the sequence of messages that realize the DISCOVERY protocol. Firstly, the UE manifests its interest in accessing platoon services to the Consumer App that it previously instantiated. The message contains information useful to find the best available platoon, such as current vehicle's position, speed, direction and destination. Such information is then forwarded by the Consumer App to the Producer

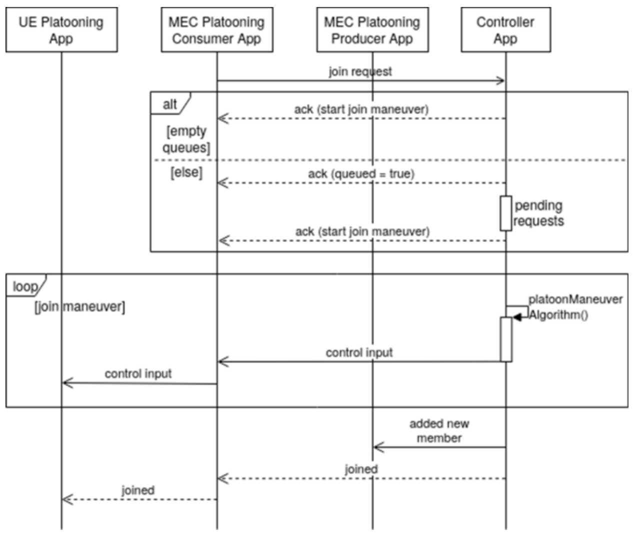


FIGURE 10. Join protocol.

App through the *discovery* message. The Producer App, upon receiving the message, runs the Platoon Selection Algorithm to retrieve the most suitable platoon among the ones tracked by the Producer App itself. This choice can possibly be made also based on information included in the discovery message. For example, the algorithm may select the closest platoon ahead traveling in the same direction. If a suitable platoon is found, then the endpoint of the Controller App handling it is returned to the Consumer App. Otherwise, the Producer App can either reject the request, or create a *new* Controller App instance – controlling a one-member platoon – and return its endpoint. The choice in this case depends on the value of the *create* flag in the discovery message: if it is set to 1, then the Producer App will instantiate a new Controller App. In this case, after the instantiation, the Producer App sends a configuration message to the new Controller App to configure the platoon, including the control algorithms to be used, the control period, the direction of the platoon, the maximum number of vehicles allowed, etc. A one-member platoon may seem pointless, at first sight: however, the vehicle forming it enjoys the advantage of having its own cruise regulated, and acts as a platoon leader for others to join later.

The sequence diagram depicted in FIGURE 10 describes the JOIN protocol. Once the Consumer App gets the endpoint of the Controller App, it sends a *join request* message. The latter, besides already described geo-related information, contains a unique identifier of the vehicle, namely the Consumer App Id. Upon receiving the request, the Controller App checks its own state: if there is already a vehicle involved in a join or leave maneuver, then the join request is enqueued and the Controller App updates the Consumer App with an ACK message having the *queued* flag set. The latter just informs the Consumer App that its join request will be served after other pending ones. As soon as the join request is popped from the queue, the *join maneuver* starts and, again, the Consumer App is notified about the update. During the *join maneuver* the Controller App periodically runs the Maneuver

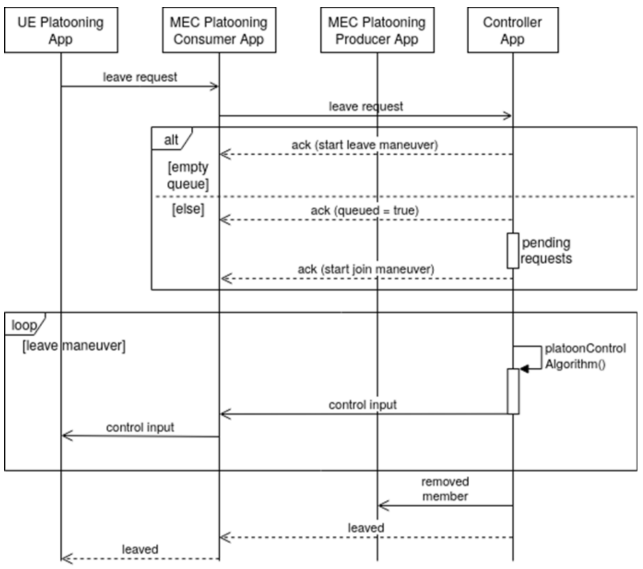


FIGURE 11. Leave protocol.

Control Algorithm to compute the acceleration (in both the longitudinal and lateral directions) of the joining vehicle. Such acceleration values are calculated based on the position and speed of the vehicle. The latter are retrieved by querying the LS running in the MEC platform: the LS provides a REST API that allows the Controller App to obtain the geographical coordinates and speed of a set of UEs [4]. The new acceleration value is then sent to the UE via its Consumer App. When the controller-specific criterion for joining a platoon is met (e.g., when the vehicle reaches a target distance to its predecessor), the Controller App inserts the Consumer App Id into the data structure representing the platoon, and sends two messages: one to notify the Producer App about the new platoon member, and one to inform the Consumer App that the join maneuver has terminated. The Consumer App in turn notifies the UE Platooning App.

The LEAVE protocol, shown in FIGURE 11, mirrors the JOIN one. It is triggered by a *leave request* message sent by the UE Platooning App, which reaches the Controller App via the Consumer App. When the request is extracted from the leave request queue, the maneuver control algorithm starts computing the acceleration values to disconnect the vehicle from the platoon. Once the leave maneuver is completed, the Consumer App Id is removed from data structures at both the Controller App and the Producer App. The latter receives a message that notifies the event. Moreover, when the last member of a platoon left, the Producer App may decide to terminate the Controller App managing the zero-member platoon.

When in CRUISE state, the Consumer App receives control inputs (i.e., acceleration values) from the Controller App and forwards them to the UE Platooning App, which in turn enforces them to the lower-level controller. FIGURE 12 shows the operations performed by one Controller App instance to generate those commands. We assume

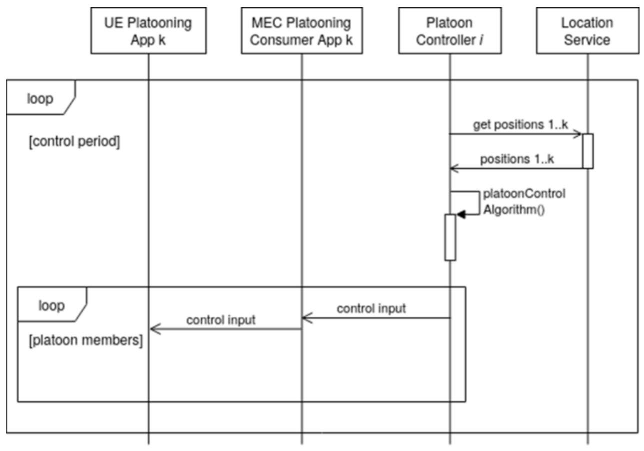


FIGURE 12. Control loop for one Controller App.

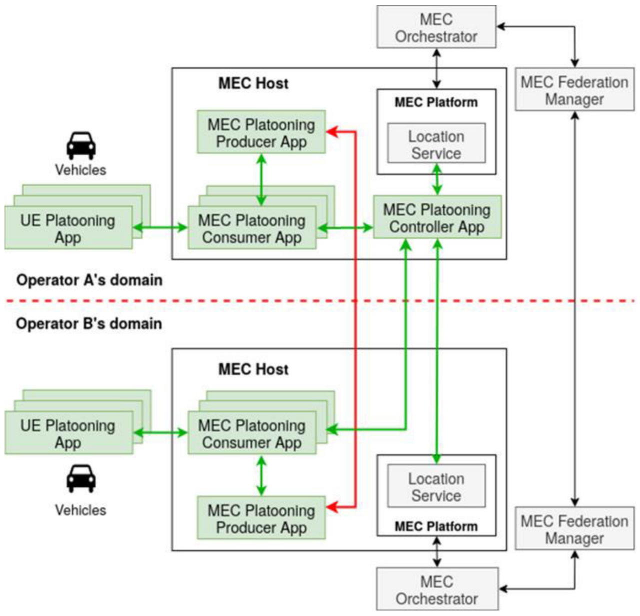


FIGURE 13. Multi-operator interactions. The red arrow highlights the (logical) federation link.

that new control inputs must be generated periodically to allow vehicles maintaining the correct position and speed. At each round, the Controller App must first update the position of current platoon members. To do so, it queries the LS again. Based on the latest positions and speeds, the Cruise Control Algorithm running in the Controller App computes a list with the new acceleration value for each member and communicates each value to the corresponding Consumer App, which forwards it to the UE Platooning App.

Last, the proposed framework needs a *keep-alive* mechanism to notify platoon status updates from the Controller App to the Producer App. The Controller App periodically sends a message with all the information about the position and speed of each member of the platoon. This serves two purposes: the first one is to keep the information about a platoon updated on the Producer App, so that a platoon can be correctly selected during the discovery phase. The second one is to provide information on the operating status of the Controller App: if a *keep-alive* message is not received by the Producer App for some time, e.g. three periods, it may mean the Controller App failed and the Producer App should take appropriate actions, e.g., hiding the related platoon in further discovery procedures and/or requesting the Consumer App to issue a warning to the driver of the vehicle.

D. GENERALIZATION TO THE MULTI-OPERATOR SCENARIO

As already discussed, a realistic road scenario includes vehicles connected to different network operators. Generally, each network operator may implement its own MEC system, hence vehicles may use applications and services provided by MEC systems managed by different operators. In order to be effective, a platooning service must support formation and management of platoons involving vehicles bound to different MEC systems. As discussed in Section II-B, MEC federation is a new paradigm introduced by ETSI for supporting inter-operator MEC applications [5]. In this section, we generalize the PlaaS framework described previously by bringing MEC federation into the picture and specifying the necessary adaptations to allow platooning to operate in a multi-operator environment. For the sake of simplicity, we refer to the case of a MEC federation composed of two MEC systems, although the same framework is valid for the case of more than two MEC systems, too. In the latter case, the MEC federation can be realized using either a mesh or a star topology (i.e., using a MEC Federation Broker). Note that such topology only affects the way the MEC systems authorize, authenticate and discover MEC apps/services in other MEC systems – which is addressed by the ETSI MEC standard, hence is outside the scope of this paper –, whereas it does not affect the flow of information required by PlaaS. In fact, once the logical connection between two MEC apps/services has been established, the communication occurs independently, in a peer-to-peer fashion.

FIGURE 13 shows the interactions between MEC entities of two different operators, namely operator A and operator B. Operator A's Producer App keeps track of the platoons managed within operator A's MEC system. This is only a partial view, as operator B's Producer App may manage other platoons in the same geographical area. Thus, the two Producer Apps must share some information to obtain a complete picture of all the active platoons. For example, when a new vehicle asks operator A's Producer App to discover a platoon, the most suitable platoon can be selected among the ones controlled locally *and* the ones controlled by operator B's Producer App. Moreover, a platoon managed by operator A's Producer App may include vehicles associated to operator B's MEC system. This means that a Controller App in operator A's MEC system may need to retrieve the position information of vehicles from the LS of operator B (or vice versa). Thus, Producer Apps expose an interface to allow a Producer App in another MEC system to acquire global knowledge

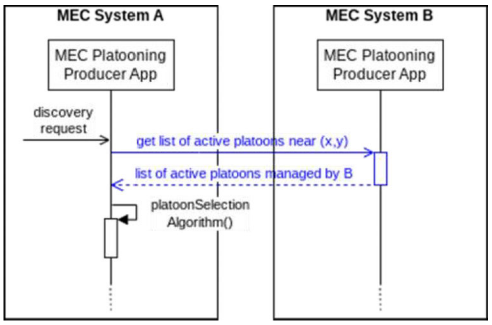

MEC System A
MEC System B
MEC Platooning Producer App
MEC Platooning Producer App
discovery request
get list of active platoons near (x,y)
list of active platoons managed by B
platoonSelection Algorithm()

FIGURE 14. Modifications to the discovery protocol (w.r.t. FIGURE 9) in a multi-operator scenarios.

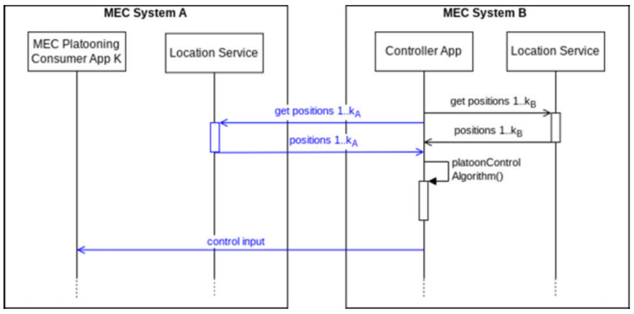


FIGURE 15. Modifications to the control loop protocol (w.r.t. FIGURE 12) in a multi-operator scenarios.

about active platoons. This information exchange is made possible by the MEC Federation Manager, which establishes an authenticated connection between the two endpoints, i.e. the two Producer Apps.

We describe hereafter how the protocols of the previous subsection need to be modified. For ease of exposition, we adopt the point of view of operator A, which is federated with B. The extension to a multi-party federation is straightforward and is left to the interested reader.

When a vehicle wants to initiate a DISCOVERY procedure at operator A, and before starting the platoon selection, A's Producer App queries B's Producer App to obtain its list of active platoons and the endpoint of the relative Controller Apps. This new interaction is depicted in blue in FIGURE 14. To reduce the amount of data exchanged, the request may specify a filter, for example based on the geographical position and travel direction of the vehicle requesting to join. After that, A's Producer App behaves like in the single-operator scenario, by executing the Platoon Selection Algorithm and returning the endpoint of the selected platoon's Controller App to the Consumer App, even if the Controller App is managed by B's Producer App and resides in operator B's MEC system. The Consumer App will then contact the Controller App, in order to do the JOIN and LEAVE operations directly. This procedure is exactly the same as in the single-operator scenario, except that the communication occurs over the MEC federation link, rather than internally to one MEC system. The only needed modification for multi-operator scenarios is that the Consumer App must insert the Id of the MEC system it belongs to in every message it sends. This allows the Controller App to select the right LS when it has to retrieve the car positions. At configuration time, in fact, the Producer App puts in the configuration message for the Controller App a list of the Producer Apps (i.e., MEC systems) involved in the federation. If, instead, a platoon is not found and the *create* flag in the discovery request is set, then the Producer App instantiates a Controller App in the operator A's MEC system. As far as the CRUISE procedure is concerned, the federation features provided by ETSI are exploited again. As shown in FIGURE 15, the control protocol is executed the same way as in the single-operator scenario, with the only difference that vehicles' positions need to be obtained from multiple LSs. For example, when the Controller App is in operator B's MEC system and the platoon includes operator A's vehicles, then the Controller App must query operator A's LS in order to retrieve the position of operator A's vehicles.

Although the modifications required to adapt the protocols to the multi-operator scenario are limited, we observe that multiple LSs with possibly different response times are now involved, and that the link(s) connecting different operators' MEC systems may introduce non-negligible delays. The effects of the above phenomena on platoon stability will be thoroughly investigated in the next section.

V. VALIDATION AND PERFORMANCE EVALUATION

In this section, we first validate our PlaaS framework, and then evaluate its performance. We have implemented it in Simu5G [1], [28], a well-known open-source 5G simulation library. Simu5G provides 3GPP-compliant 4G/5G access, and – being based on OMNeT++ [29] and INET [30] – is interoperable with other similar libraries, e.g., Veins [31], providing vehicular mobility. Simu5G allows one to simulate arbitrary 5G network layouts, including all the protocols from the application to the physical layer. Moreover, it includes a model of the ETSI MEC architecture, described in [27], allowing users to simulate MEC-based applications. Simu5G's MEC model includes the RNIS and LS, the latter being useful to support PlaaS. All the software used in this paper is publicly available at [32].

We simulate the scenario shown in FIGURE 16. A group of seven cars moves along the road in a straight line. The stretch of motorway is served by two Mobile Network Operators (MNOs) – namely A and B –, each one having four gNodeBs (gNBs, i.e., 5G New-Radio base stations) co-located, at 500m of distance to each other. Three cars, named *A[∗]*, belong to MNO A, whereas the other four, *B[∗]*, belong to MNO B. The two MNOs use different carrier bands (2GHz and 4GHz, respectively) and cars perform intra-MNO handovers based on the Received Signal Strength Indicator (RSSI). We assume that each gNB is configured in Frequency Division Duplexing (FDD) mode, and both downlink and uplink spectra are allocated 10 MHz-bandwidth with numerology index 0, resulting in subframe duration of 1ms. UEs periodically report their channel status information every 40 subframes. As far as the

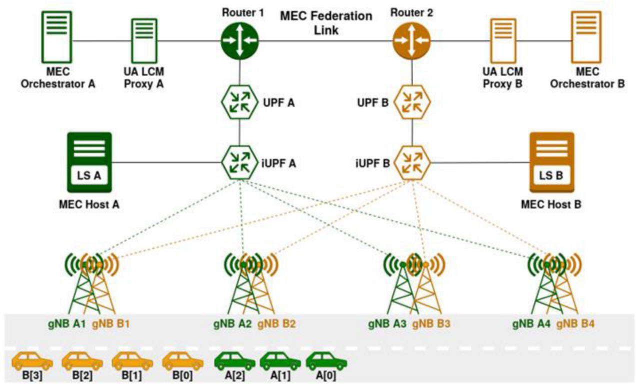


FIGURE 16. Scenario used for validation and evaluation.

TABLE 2. Main 5G radio access simulation paramaters.

Parameter Name	Value
Carrier frequency	MNO A: 2 GHz MNO B: 4 GHz
Bandwidth	10 MHz
Numerology index	0
Downlink/Uplink duplexing	Frequency Division Duplexing
Tx power	gNBs: 46 dBm; UEs: 26 dBm
Antenna gain	gNBs: 8 dBi; UEs: 0 dBi
Noise figure	gNBs: 5 dB; UEs: 7 dB
Chan. status reporting period	40 TTIs
Path loss model	Urban Macro (UMa) [33]
Fading model	Jakes
Shadowing model	Log-normal distribution

physical layer of the radio channel is concerned, gNBs and UEs transmit at 46 and 26dBm, respectively, and the channel effects are modeled according to the Urban Macro scenario defined by 3GPP specifications [33]. The parameters of the 5G networks are summarized in TABLE 2.

Each MNO has its own MEC system, including a single MEC host. A MEC federation allows the communication between the two MEC systems, so that platooning controllers can manage platoons having cars connected to different MNOs. All the cars will be associated, by the relative MEC Producer Apps, to the same Controller App to form a single platoon. Cars perform a *join request* in sequence, i.e., from A[0] to B[3], so that the join maneuver is always performed at the tail of the platoon. For the sake of validation and evaluation, our PlaaS uses the same control algorithm for both the cruise and maneuver controllers, i.e., [34]. This is widely used, and it has been implemented in our framework as one of the available controllers.

A. VALIDATION

To validate the framework, we simulated the above scenario by placing the cars at different distances from each other (between 8 and 30 meters) and running at the same speed of 25 m/s. The control algorithms have been configured as shown in TABLE 3. The Controller App requests the updated values of cars' position to the ETSI MEC LS and, after 30ms,

TABLE 3. Configuration of the control algorithms.

Parameter	Value
Position update period	100 ms
Control algorithms period	100 ms
Offset between position update and control period	30 ms
Target inter-vehicle distance	10 m
Target speed	25 m/s
Information type sent to the car	Acceleration vector

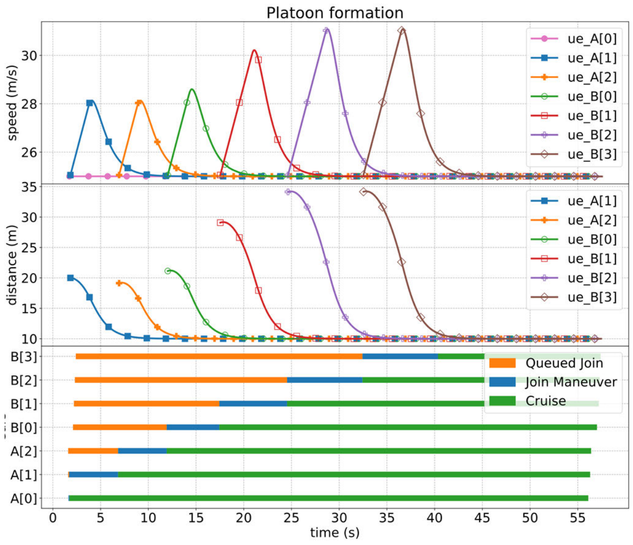


FIGURE 17. Validation of the *Join maneuver.*

it executes the control algorithm that produces new control inputs for the cars, i.e., new acceleration values. This procedure is repeated every 100ms. Concerning the control algorithm, we configure it in order to let cars reach a target speed of 25m/s and inter-vehicle distance of 10m.

FIGURE 17 shows the speed (top), the distance from the preceding car (middle) and the state of the cars (bottom) during the formation of the platoon. We can see that the platoon is completely formed after about 41 seconds (when all the cars are in CRUISE state), at which point in time the speed of each car is 25 m/s and the distance between the cars is 10m, as configured in the Controller App. During the initial stages, i.e., the first two seconds, cars request the Producer App to be associated to the best platoon. Since no platoons are already available, the first car, i.e., A[0], triggers the instantiation of a Controller App that will manage the platoons by means of the Producer App. The following *discovery* requests coming from other cars are then queued up until the Controller App is created. Once a car knows the Controller App Id and the corresponding endpoint, it sends a *join request* to it. The latter is queued (orange line in the bottom graph) if another car is already in the JOIN MANEUVER state. After a car moves to the CRUISE state, it is controlled by the Cruise Control Algorithm run by the Controller App.

FIGURE 18 shows what happens when a car in the middle of the platoon leaves. Around second 51, car A[2] requires to

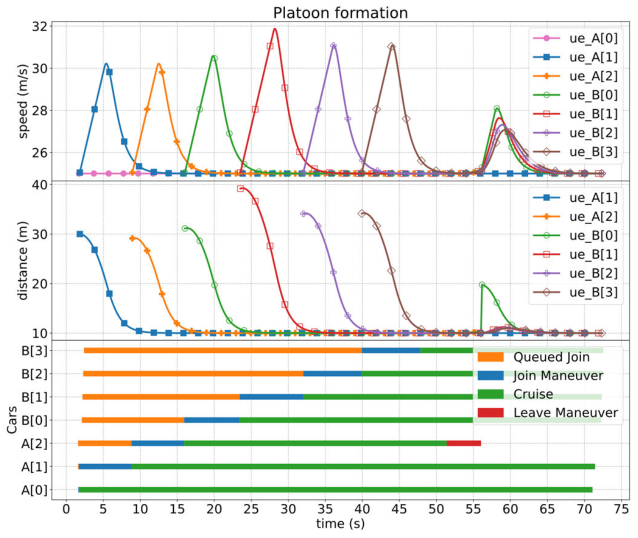


FIGURE 18. Validation of the *Leave maneuver.*

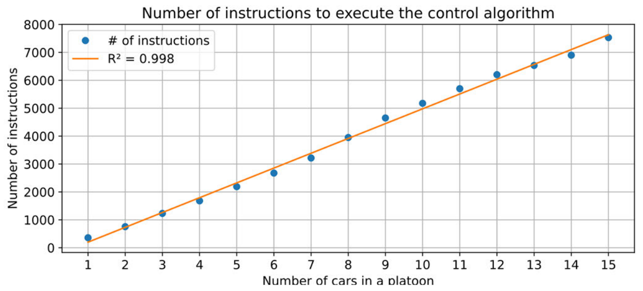


FIGURE 19. Number of machine instructions run by controller [34] as a function of the number of vehicles (dots) and linear interpolation (solid line).

leave the platoon. As shown by the red bar in the bottom chart in FIGURE 18, it enters the LEAVE MANEUVER state until the Maneuver Control Algorithm terminates the operation, i.e. until the car completed a line change operation and moved to the side of the platoon. At the end of the operation, car A[2] is detached from the platoon and cars B[∗] following it briefly accelerate (see the top chart, between 55 and 65 seconds) to cover the gap to A[1]. Around 67 seconds, the platoon reached stability again, with each car following its predecessor at the target distance.

B. PERFORMANCE EVALUATION

Our performance evaluation is aimed at proving that platoon control through our framework is feasible. In order to do so, we need to show two properties: first, that it is scalable. Second, since this system has a closed feedback loop, that potential sources of delay (i.e., computations and network transit, especially in a federated MEC environment) do not significantly affect its operation. We are able to do this since our simulation environment includes credible, detailed models of both 5G network transit and the ETSI MEC environment. Our results do depend on the particular Cruise Control Algorithm used in our evaluation, i.e., [34]. On one hand, this is one of the most widely studied controllers, which makes our analysis relevant – we are not aware of any such evaluation in the literature. On the other hand, our methodology is entirely general, and can be followed by researchers interested in evaluating their own control algorithms – which they can easily plug in our software framework.

1) IMPACT OF COMPUTATION TIMES OF THE CONTROLLER LOGIC

We first discuss the impact of the computations run by the control algorithms. Note that in our scenario both the Cruise and Maneuver Control Algorithms are the same. Ideally, we should evaluate *how long* it takes for the algorithm to complete a control loop, based on the number of cars. However, the Controller App runs in a virtualized environment (a MEC host), hence the actual running time of its algorithm will heavily depend on the amount of resources that the virtual machine or container running it is allocated, as well as on the speed of the physical infrastructure. For this reason, we instead measure the *number of machine instructions* run by the algorithm. That number can be (roughly) translated to an execution time, given an estimate of the number of instructions per second that the Controller App is allocated in the infrastructure. Our simulator runs computations instantly – in *simulated* time – but allows one to insert *computation delays* at the end of such computations, to delay the response of a MEC App. These delays are computed based on the number of machine instructions of the computation, the MEC host CPU speed and the fraction of CPU allocated to the MEC App [27].

We ran simulations with an increasing number of cars, and counted the number of machine instructions executed by the Cruise Control Algorithm. Such number has been calculated by using the *perf_event_open()* system call, which allows one to measure CPU cycles count, machine instructions, cache misses, etc.[2] Note that the number of machine instructions depends on the language compiler and the CPU architecture. Our code runs on an i5-5300U Intel CPU, and has been compiled with the *clang* compiler using the -o3 optimization option. The results reported in FIGURE 19 show that the number of machine instructions increases linearly with the number of cars, which makes this control algorithm quite scalable. With 15 cars, the control algorithm executes on average 7531 machine instructions (results are very consistent across replicas, hence no confidence intervals are reported). On today's processors, that can execute hundreds of million instructions per second (MIPS),[3] this number translates to negligible times (microseconds), even assuming that the Controller App is allocated a very low CPU percentage. It is easy to see (and we will also show it later) that μs-scale delays have no impact on the performance. For this controller, the computation times of the Controller App will not be a scalability bottleneck, unless a *very large* platoon is envisaged (in the order of hundreds of vehicles).

[2]https://man7.org/linux/man-pages/man2/perf_event_open.2.html
[3]https://en.wikipedia.org/wiki/Instructions_per_second

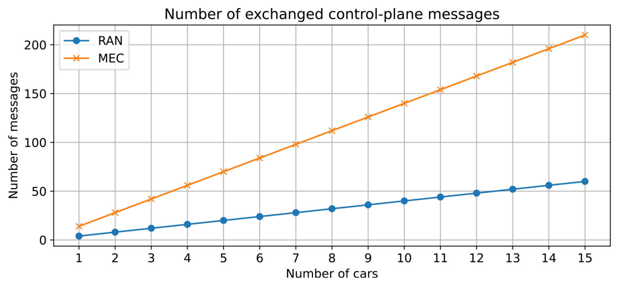


FIGURE 20. Number of messages required to discover, join and leave a platoons in radio interface and MEC with increasing number of cars.

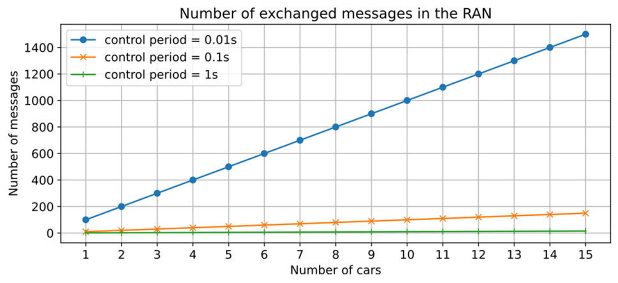


FIGURE 21. Number of messages required in the radio interface to control the cars with increasing number of cars and control period.

2) IMPACT OF COMMUNICATION OVERHEAD

We now analyze the impact of the communication between the involved entities. As PlaaS introduces a new set of messages to discover, join, and leave a platoon, we first consider how much overhead such messages introduce in the network, both over the radio interface and in the core network (i.e., communication within a MEC system or between different MEC systems). FIGURE 20 shows the number of messages that are required to manage the platoons with increasing number of cars. In other words, we consider the protocols shown in FIGURE 9, FIGURE 10, and FIGURE 11, hence excluding the messages required to issue acceleration/brake commands to the cars, which would be required nevertheless even without our PlaaS. Since most of those messages involved intra- or inter-MEC communications, the number of messages sent over the radio is minimal. Likewise, the messages exchanged within the core network are in the order of the few hundreds, which can be easily supported by high-bandwidth connections available in the core network. Note that most of these messages could actually be exchanged among entities that reside in the same MEC host. Thus, the communication overhead introduced by PlaaS can be safely considered negligible.

As far as the control loop of the platoon members is concerned, FIGURE 21 shows the number of messages sent in one minute from the PlaaS to the cars over the radio interface, with different values of the control period. When the control period is as fast as 0.01s and the number of cars is high, the number of (downlink) messages can reach large values. However, we observed in Section V-A that a control period of 0.1s is enough to control the platoon satisfactorily, and in this case the number of messages over the radio interface is manageable.

Finally, we compare the number of control commands sent over the radio interface with PlaaS against other approaches, namely the one presented in [10] and a baseline without MEC. In the latter, the platoon leader receives updates from other platoon members, runs the controller logic locally and sends commands back to the platoon members. FIGURE 22 shows that our PlaaS consumes less radio resources than both the other approaches. In fact, in [10] it is assumed that at every control cycle the cars also send their status update in the uplink, whereas PlaaS exploits the ETSI MEC LS. When MEC is not used at all, instead, each message

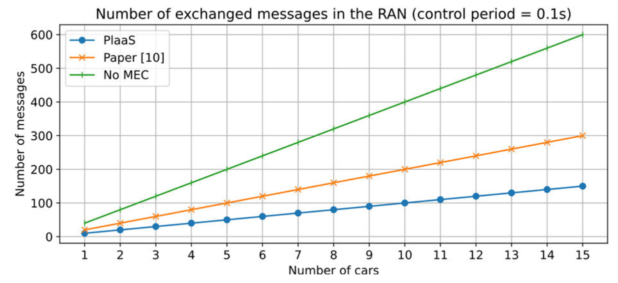


FIGURE 22. Comparison of the number of messages required in the radio interface to control the cars against the ones required in [10] and in a scenario without MEC (centralized control performed by the platoon leader).

(both status update from the platoon members and control commands from the platoon leader) involves two radio transmissions: one in the uplink to the base station and one in the downlink to the final destination.

3) IMPACT OF THE DELAYS AT THE LOCATION SERVICE

Our PlaaS obtains position information from the ETSI MEC LS, using a request-response pattern. Note that the Controller App makes *one request* per period to the LSs of MEC systems A and B. That request includes queries for the positions of *all* the cars, which are returned by the LSs in a single response. This may incur some delay, due to, for example, a high load of requests at the LS server, or to the time needed by it to retrieve the positions (e.g., because the position database is far from the server, or MNO's policies managing user sensitive data). To evaluate the impact of these delays, we simulated the same scenario with a platoon of seven cars, already formed. The LS response times are calculated according to an exponential distribution with rate λ. We now assume *instantaneous* communication between the MEC systems of MNOs A and B – we will evaluate the impact of delays on the federation link later. We analyze what happens when one of the LSs has a much higher response time than the other, and when both have a relatively high response time. We consider the three different scenarios reported in TABLE 4, where each row reports the mean response time of the two LSs involved in the corresponding scenario. Results related to the inter-vehicle distance are depicted in FIGURE 23. A non-obvious fact is that only *one* car is significantly affected by the (heavy) mismatch between the two response times, i.e., B[0] – the first

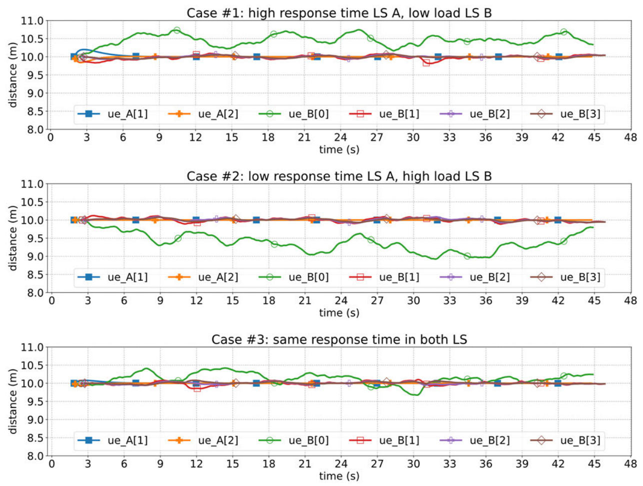


FIGURE 23. Inter-vehicle distance when the LS response time changes.

TABLE 4. Response times at the LS of the two MNOs.

Case	Response time	λ^{-1} LS A	λ^{-1} LS B
#1	LS A: high; LS B: low	30ms	0.5ms
#2	LS A: low; LS B: high	0.5ms	30ms
#3	LS A: high; LS B: high	15ms	15ms

one connected to a different MNO (and MEC system) with respect to the preceding cars –, whose distance has an offset of up to ±1m with respect to the target. All other cars – from both MNOs – are unaffected. This shows that LS delays are not a scalability problem: they would be if these offsets were *increasing* along the platoon, which they are not. On the other hand, we need to understand better the relationship between LS delays and the inter-vehicle distance offset experienced by car B[0], which we can do by focusing on B[0] itself, the only one tangibly affected.

We run the same scenario varying the mean response time of both LSs from 0.5 ms to 30 ms. FIGURE 24 and FIGURE 25 show, respectively, the average and 95th percentile of the absolute value of the deviation w.r.t. the target distance between B[0] and A[2] (95% confidence intervals, calculated over 30 independent replicas, are negligible, hence not reported). Both heatmaps show a similar pattern: the offset increases with the mismatch in the LS response times. This is because, when the response times of the LS are different, the positions of the cars of MNO A and B *are fetched at different times*. This leads the Cruise Control Algorithm to use inaccurate positions when calculating the acceleration values that are meant to keep a car at its target distance and speed. Note that a vehicle moving at 25 m/s covers 0.75 m in 30 ms, which is comparable to the offsets that we have observed in FIGURE 23. This also confirms that μs-scale computation delays (referred to in subsection V-B.1) are in fact immaterial.

The above problem occurs at B[0], i.e. the only car whose predecessor is of a different MNO. This tells us that a PlaaS service should strive to form platoons so that *cars of the same MNO occupy consecutive positions*. This non-trivial result

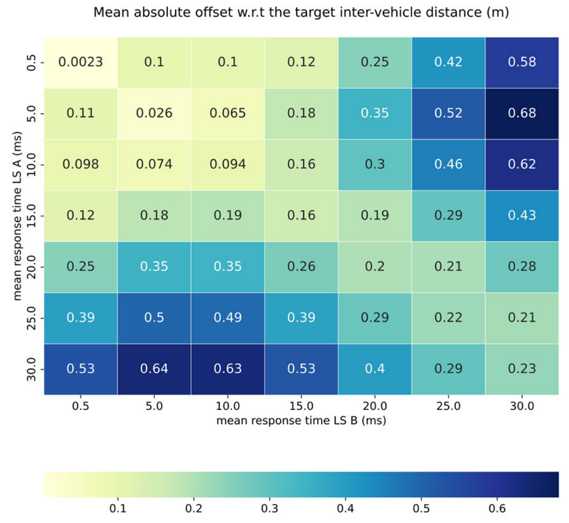


FIGURE 24. Mean absolute offset with respect to the target inter-vehicle distance of car B[0] as a function of the mean response time of the LS at MEC systems A and B.

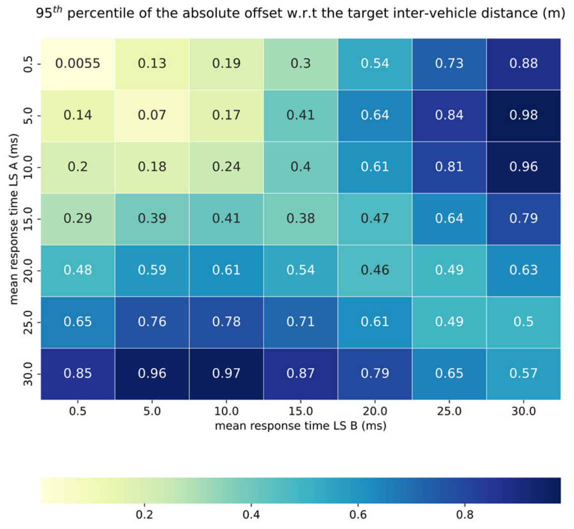


FIGURE 25. 95th percentile of the absolute offset with respect to the target inter-vehicle distance as a function of the mean response time of the LS at MEC systems A and B.

can drive the *design of the join maneuver logic*, in order to select the most suitable position where a car must join the platoon. By instructing the car to join the platoon right behind (or in front of) another car connected to the same MNO, the number of cars affected by the above instability problem will be minimized.

We provide more solid ground to this observation by simulating a scenario with four MNOs – namely A, B, C and D –, each of them serving four cars – with index ranging from 0 to 3. All the cars belong to the same platoon and A[0] is the platoon leader. We assume that the Controller App resides on MNO A's MEC system, and that the LS response time is 0.5ms for MNO A's cars and 30ms for other MNOs' cars. We observe the inter-vehicle distance, comparing the case where cars belonging to the same MNO occupy consecutive

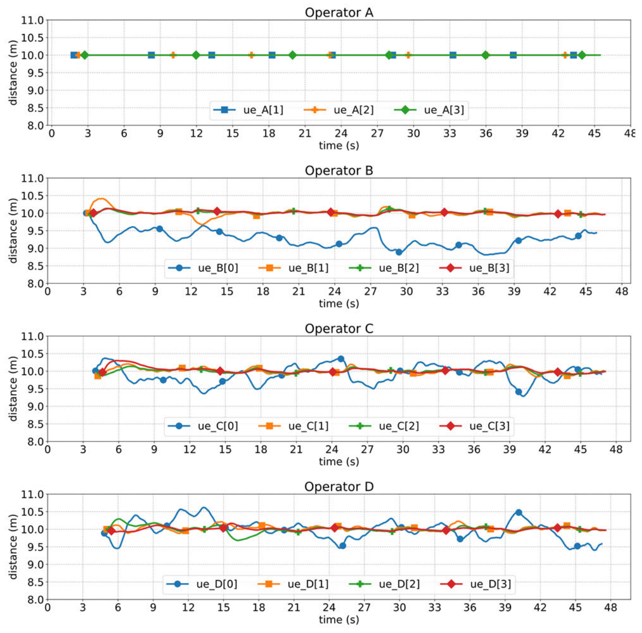


FIGURE 26. Inter-vehicle distance when cars of the same MNO occupy consecutive positions in the platoon.

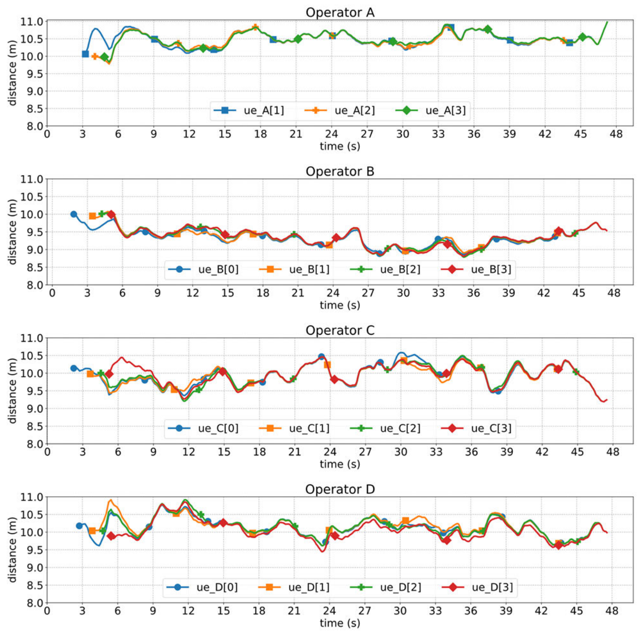


FIGURE 27. Inter-vehicle distance when cars of the same MNO occupy non-consecutive positions in the platoon.

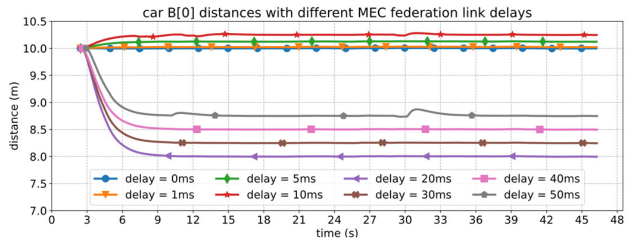


FIGURE 28. Inter-vehicle distance of car B[0] as a function of the delays on the federation link.

positions in the platoon (FIGURE 26) against the case where two back-to-back cars always belong to different MNOs (FIGURE 27). The latter case may occur in a situation where the platoon enforces a simple join maneuever logic that makes every new car join, e.g., at the tail of the platoon regardless the MNO of the cars. In both cases, we group vehicles of an MNO in the same subfigure, for better readability. FIGURE 26 shows that only the *first car* of every MNO (i.e., the ones with index 0) experiences some instability, whereas the other ones have a near-constant inter-vehicle distance. Even the very last car of the platoon, i.e. D[3], shows stable inter-vehicle distance, hence demonstrating that our framework is valid also for platoons composed of a large number of cars. Instead, FIGURE 27 shows that when the order of cars in the platoon is not carefully managed the inter-vehicle distance is highly unstable for all the cars.

From the above charts, the offset of the first car of a MNO in the platoon can be around 1m, which may or may not be tolerable. We will discuss later a simple way to mitigate this problem. We choose to discuss the impact of delays on the federation link first, because it has similar effects, and our solution will address both.

4) IMPACT OF DELAYS BETWEEN THE TWO MEC SYSTEMS

MEC systems communicate via a MEC federation link, established between two peering points. Inter-MEC system communications may incur some delay, for several reasons: the load on the MEC federation link may be high, or the link may be rate-limited because of peering Service Level Agreements; external traffic may be subject to encapsulation/decapsulation, reshaping, cyphering/deciphering, etc. All these may add some delay, whose impact needs to be assessed.

We analyze the system in the same scenario, with *small* response times at both LSs A and B (0.5 ms) and a delay on the federation link varying between 1 and 50 ms. Again, we only focus on the car B[0] because we ascertained – after a preliminary investigation, not shown for the sake of conciseness – that it is the most affected one. The results showing the inter-vehicle distance are shown in FIGURE 28.

The figure shows a threshold behavior: when the delay is below 20 ms (top four lines), there is a *minor* increase in the inter-vehicle distance. As soon as the delay reaches 20 ms, the distance drops remarkably, to values below 9 m. Note that, in this range of delays, the relationship between the delay and the offset is also *non-monotonic*. This unexpected behavior is due to the superimposition of two different effects. One is the fact that the Cruise Control Algorithm works with stale positions for B[∗] cars, this time due to the round-trip time (RTT) on the federation link, which adds to the response time of LS B. This, as we discussed in the previous evaluation, creates an offset in the inter-vehicle distance A[2]-B[0].

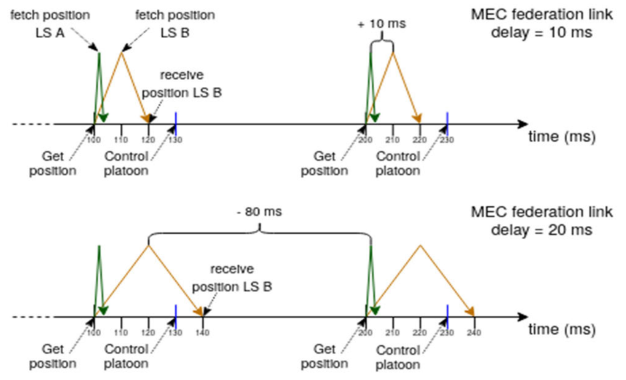


FIGURE 29. Interplay between control periods and position requests, with short delay (top) and large delay (bottom) on the federation link.

Moreover, and uniquely to this example, there is an interplay between the response times and the control loop period, which requires a careful explanation. As per FIGURE 29, both position requests and platoon control loops are run every 100 ms, and there is a 30 ms offset between the first and the second. FIGURE 29 shows two consecutive instances of position requests, at 100 and 200 ms, together with their RTT on LS A and B, as well as two instances of control loops, at 130 and 230 ms.

When the federation link has a (one-way) delay below 15 ms, (top timeline in FIGURE 29), the RTT of LS B is smaller than the 30 ms offset, hence the *most recent* position acquired for B[0] is always used by the Cruise Control Algorithm. Note that this position is also *more recent* than the one of A[2] (by about 10 ms, in the example). In this case the algorithm will presume that B[0] is closer to A[2] than it should, and will decelerate it, moving it further back until the longer distance compensates the RTT. In fact, for delays smaller than 15 ms on the federation link, B[0] stabilizes to a distance just above 10 m (see FIGURE 28).

When, instead, the one-way delay on the federation link exceeds 15ms (bottom timeline in FIGURE 29), then the algorithm (e.g., at $t = 230ms$) will use a *stale* position for B[0], i.e. one that was requested at $t = 100ms$ and reflects where B[0] was more than one control period before. This will prompt the algorithm to *accelerate* car B[0] instead, thus stabilizing it on a (much) shorter distance. This also explains why – counterintuitively – the error *decreases* with the federation link delay in this case: with higher delays, B[0]'s position is paradoxically *more recent*, hence the acceleration mismatch will stabilize at a distance closer to the ideal value. For example, with a federation link delay of 20 ms, the timestamp difference between cars B and A is 80 ms, which at 25 m/s creates a 2-meters gap. With a 50 ms delay, instead, the difference is 1.25 m, as confirmed by FIGURE 28.

5) COUNTERING THE EFFECTS OF NETWORK DELAYS

The previous subsections show that network/system delays may be harmful, because they mislead the Cruise Control Algorithm into assuming that vehicles are not where they actually are, making it compute wrong accelerations. A very simple workaround for this problem is to use the *timestamps* within the responses of the LSs, and feed them to the control algorithm together with the positions. This way, the control algorithm can compute a more realistic estimate of the *actual* vehicle position when it computes accelerations. Assume that the control algorithm runs at time t, and receives from the LS a position computed at time $t - \Delta t$. It then can easily compute the distance covered by the vehicle in Δt, based on the speed and acceleration computed in the previous cycle.

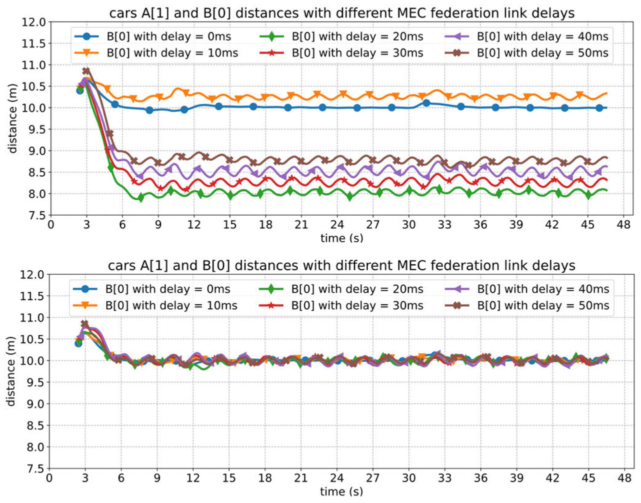


FIGURE 30. Inter-vehicle distance of B[0] before (top) and after (bottom) timestamp-based position adjustment.

We now show what happens when delays are compensated by the control algorithm using these timestamps. To better highlight the ensuing improvements, we need to setup slightly different simulation conditions: more specifically, we need the platoon leader *not* to have a constant speed. Otherwise, since all cars in our simulations start as an already formed platoon, the position correction would lead to zero acceleration and the cars would not change their behavior. Accordingly, the leader has been configured to vary its speed[4] according to a sinusoidal pattern with values between 23.6 and 26.4 m/s [10].

FIGURE 30 shows the distance of car B[0] to its predecessor, before and after timestamp-based position adjustment. The correction described above greatly reduces imperfections. Only a small, residual oscillation around the target distance is still visible, which is due to the fact that the acceleration commands that need to reach cars B[∗] will still have to pass through the federation link, hence will get to their destination later than those meant for cars A[∗]. However, this does not create undue problems. It is in fact easy to observe that this technique – which we just proved effective against the federation link delay – actually compensates position delays whatever their origin, and will thus have the same effect on all the others mentioned in this evaluation.

[4]Note that algorithm [34] also controls the acceleration of the leader. For this experiment, we tweaked it so that it takes the actual speed of the leader as an *input* datum.

C. DISCUSSION

We now briefly address a few items that should be taken into account when defining and provisioning a PlaaS.

The first is the communication overhead of our PlaaS. This is not really an issue here: the period and content of messages in both directions is such as to impose a very light load already on a 4G network, let alone on a 5G one. Moreover, message exchange between MEC apps and MEC services occurs on a wired network infrastructure, which can be expected not to represent a bottleneck.

The second is the accuracy of the LS. We have seen in the previous sections that the accuracy of platoon control depends on the (temporal) accuracy of position information. One may then wonder if the *spatial*(in)accuracy of the LS, due to inherent constraints of the employed localization algorithms, may add up to similar – or worse – undesirable consequences. Currently, the location information provided by a 5G network allows one to identify UEs with an uncertainty in the order of a few decimeters [35]. Ongoing projects target uncertainties smaller than one centimeter for the near future [36]. These inaccuracies are clearly tolerable, given the speeds and timings involved.

We also observe that the results reported in this section show little, if any, dependence on the *type* of vehicles involved. We repeated the experiments with different maximum acceleration, to model e.g. trucks or vans, and we obtained qualitatively similar results, which we omit for the sake of conciseness.

As far as possible limitations are concerned, we highlight that enabling the proposed PlaaS in a multi-operator scenario requires that all the involved multiple operators support ETSI MEC Federation, hence implementing and exposing the required APIs to allow other operators' MEC systems to discover and use the respective services, such as Producer Apps and LS. Moreover, service-level agreements among the involved operators might need to be in place in order to, e.g., allow a Producer App to exploit the LS exposed by another operator's MEC system. In this regard, incentives mechanisms may be required to foster interactions among different MEC operators.

VI. CONCLUSION AND FUTURE WORK

In this paper we have presented – for the first time, to the best of our knowledge – a comprehensive software architecture to implement ETSI MEC-based platooning in a multi-operator environment. In the latter, operators leverage MEC federation to share, in a controlled way, the information that allows users to locate, join, cruise along with, and leave, platoons formed by vehicles subscribed to multiple operators. We have described the main software components and their interactions and protocols, which are coherent with the ETSI-MEC industrial standard. Furthermore, we have thoroughly investigated the practical feasibility of MEC-based platoon control in a federated context. We have shown that computation overhead is never an issue, nor is communication latency on a 5G network. What may create instability and/or suboptimalities is the temporal inaccuracy in position reporting, due to different system delays. We have shown the extent of these problems and proposed an inexpensive way to solve them. Our study demonstrates the viability of MEC-based platooning in a multi-operator context, with control information carried across a 5G network.

We have been able to do this by experimenting on an open-source simulator, Simu5G, which includes 5G network access, ETSI MEC, vehicular mobility and platooning. All our code is made available to the community, including sample platoon control algorithms found in the literature. We believe that this represents a contribution for the scientific community, enabling researchers to test their findings in a common, comprehensive and validated environment. Moreover, a very recent paper [16] shows that, thanks to the modular nature of Simu5G, it is possible to connect it to the CARLA urban driving simulator [37], designed for autonomous driving research and capable of modeling, sensing and controlling the physical environment in great detail. This adds value to the software framework presented in this paper, making it usable to setup experiments in complex scenarios.

As a future work, we plan to extend our framework by implementing the operations to split and merge platoons, as well as new control algorithms specifically designed for centralized platooning. Moreover, we plan to follow up on [16], exploiting the opportunities opened by this work – e.g., to test our PlaaS framework in connection with CARLA. Blockchain technologies may also be investigated as a viable option to incentivize the collaboration among multiple MEC operators, hence to overcome potential reluctance from operators to expose their services to other operators, as discussed in Section V-C.

REFERENCES

[1] G. Nardini, D. Sabella, G. Stea, P. Thakkar, and A. Virdis, "Simu5G—An OMNeT++ library for end-to-end performance evaluation of 5G networks," *IEEE Access*, vol. 8, pp. 181176–181191, 2020, doi: 10.1109/ACCESS.2020.3028550.

[2] *Multi-Access Edge Computing (MEC); Framework and Reference Architecture*, Standard ETSI GS MEC 003, Version 2.2.1, Dec. 2020.

[3] *Mobile Edge Computing (MEC); Radio Network Information API*, Standard ETSI GS MEC 012, Version 1.1.1, Jul. 2017.

[4] *Multi-Access Edge Computing (MEC); Location APIdocument*, Standard ETSI GS MEC 013, Version 2.1.1, Sep. 2019.

[5] *Multi-Access Edge Computing (MEC); Study on Inter-MEC Systems and MEC-Cloud Systems Coordination*, Standard ETSI GS MEC 035, Version 3.1.1, Jun. 2021.

[6] X. Fan, T. Cui, C. Cao, Q. Chen, and K. S. Kwak, "Minimum-cost offloading for collaborative task execution of MEC-assisted platooning," *Sensors*, vol. 19, no. 4, p. 847, Feb. 2019.

[7] Y. Hu, T. Cui, X. Huang, and Q. Chen, "Task offloading based on Lyapunov optimization for MEC-assisted platooning," in *Proc. 11th Int. Conf. Wireless Commun. Signal Process. (WCSP)*, Oct. 2019, pp. 1–5, doi: 10.1109/WCSP.2019.8928035.

[8] A. B. De Souza, P. A. L. Rego, T. Carneiro, J. D. C. Rodrigues, P. P. R. Filho, J. N. De Souza, V. Chamola, V. H. C. De Albuquerque, and B. Sikdar, "Computation offloading for vehicular environments: A survey," *IEEE Access*, vol. 8, pp. 198214–198243, 2020, doi: 10.1109/ACCESS.2020.3033828.

[9] A. Virdis, G. Nardini, and G. Stea, "A framework for MEC-enabled platooning," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshop (WCNCW)*, Apr. 2019, pp. 1–6, doi: 10.1109/WCNCW.2019.8902910.

[10] C. Quadri, V. Mancuso, M. A. Marsan, and G. P. Rossi, "Edge-based platoon control," *Comput. Commun.*, vol. 181, pp. 17–31, Jan. 2022, doi: 10.1016/j.comcom.2021.09.021.

[11] C. Quadri, V. Mancuso, V. Cislaghi, M. A. Marsan, and G. P. Rossi, "From PLATO to platoons," in *Proc. 19th Medit. Commun. Comput. Netw. Conf. (MedComNet)*, Jun. 2021, pp. 1–8, doi: 10.1109/MedComNet52149.2021.9501242.

[12] S. Dabbene, C. Lehmann, C. Campolo, A. Molinaro, and F. H. P. Fitzek, "A MEC-assisted vehicle platooning control through Docker containers," in *Proc. IEEE 3rd Connected Automated Vehicles Symp. (CAVS)*, Nov. 2020, pp. 1–6, doi: 10.1109/CAVS51000.2020.9334658.

[13] R.-H. Huang, B.-J. Chang, Y.-L. Tsai, and Y.-H. Liang, "Mobile edge computing-based vehicular cloud of cooperative adaptive driving for platooning autonomous self driving," in *Proc. IEEE 7th Int. Symp. Cloud Service Comput. (SC2)*, Nov. 2017, pp. 32–39, doi: 10.1109/SC2.2017.13.

[14] U. Montanaro, S. Fallah, M. Dianati, D. Oxtoby, T. Mizutani, and A. Mouzakitis, "Cloud-assisted distributed control system architecture for platooning," in *Proc. 21st Int. Conf. Intell. Transp. Syst. (ITSC)*, Nov. 2018, pp. 1258–1265, doi: 10.1109/ITSC.2018.8569295.

[15] T. Xiao, C. Chen, Q. Pei, and H. H. Song, "Consortium blockchain-based computation offloading using mobile edge platoon cloud in Internet of Vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 10, pp. 17769–17783, Oct. 2022, doi: 10.1109/TITS.2022.3168358.

[16] C. Ayimba, V. Cislaghi, C. Quadri, P. Casari, and V. Mancuso, "Copy-CAV: V2X-enabled wireless towing for emergency transport," *Comput. Commun.*, vol. 205, pp. 87–96, May 2023, doi: 10.1016/j.comcom.2023.04.009.

[17] C. Ayimba, M. Segata, P. Casari, and V. Mancuso, "Driving under influence: Robust controller migration for MEC-enabled platooning," *Comput. Commun.*, vol. 194, pp. 135–147, Oct. 2022.

[18] Q. Wu, Z. Wan, Q. Fan, P. Fan, and J. Wang, "Velocity-adaptive access scheme for MEC-assisted platooning networks: Access fairness via data freshness," *IEEE Internet Things J.*, vol. 9, no. 6, pp. 4229–4244, Mar. 2022, doi: 10.1109/JIOT.2021.3103325.

[19] T. Sturm, C. Krupitzer, M. Segata, and C. Becker, "A taxonomy of optimization factors for platooning," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 10, pp. 6097–6114, Oct. 2021, doi: 10.1109/TITS.2020.2994537.

[20] A. Vinel, L. Lan, and N. Lyamin, "Vehicle-to-vehicle communication in C-ACC/platooning scenarios," *IEEE Commun. Mag.*, vol. 53, no. 8, pp. 192–197, Aug. 2015, doi: 10.1109/MCOM.2015.7180527.

[21] M. R. Hidayatullah and J. Juang, "Centralized and distributed control framework under homogeneous and heterogeneous platoon," *IEEE Access*, vol. 9, pp. 49629–49648, 2021, doi: 10.1109/ACCESS.2021.3068968.

[22] V. Vukadinovic, K. Bakowski, P. Marsch, I. D. Garcia, H. Xu, M. Sybis, P. Sroka, K. Wesolowski, D. Lister, and I. Thibault, "3GPP C-V2X and IEEE 802.11p for vehicle-to-vehicle communications in highway platooning scenarios," *Ad Hoc Netw.*, vol. 74, pp. 17–29, May 2018, doi: 10.1016/j.adhoc.2018.03.004.

[23] G. Nardini, A. Virdis, C. Campolo, A. Molinaro, and G. Stea, "Cellular-V2X communications for platooning: Design and evaluation," *Sensors*, vol. 18, no. 5, p. 1527, May 2018, doi: 10.3390/s18051527.

[24] S. Badnava, N. Meskin, A. Gastli, M. A. Al-Hitmi, J. Ghommam, M. Mesbah, and F. Mnif, "Platoon transitional maneuver control system: A review," *IEEE Access*, vol. 9, pp. 88327–88347, 2021, doi: 10.1109/ACCESS.2021.3089615.

[25] T. Ojanperä, H. van den Berg, W. IJntema, R. de Souza Schwartz, and M. Djurica, "Application synchronization among multiple MEC servers in connected vehicle scenarios," in *Proc. IEEE 88th Veh. Technol. Conf. (VTC-Fall)*, Aug. 2018, pp. 1–5, doi: 10.1109/VTCFall.2018.8691039.

[26] M. Segata, B. Bloessl, S. Joerer, F. Dressler, and R. L. Cigno, "Supporting platooning maneuvers through IVC: An initial protocol analysis for the JOIN maneuver," in *Proc. 11th Annu. Conf. Wireless On-Demand Netw. Syst. Services (WONS)*, Apr. 2014, pp. 130–137, doi: 10.1109/WONS.2014.6814733.

[27] A. Noferi, G. Nardini, G. Stea, and A. Virdis, "Rapid prototyping and performance evaluation of ETSI MEC-based applications," *Simul. Model. Pract. Theory*, vol. 123, Feb. 2023, Art. no. 102700, doi: 10.1016/j.simpat.2022.102700.

[28] *Simu5G*. Accessed: May 2023. [Online]. Available: http://simu5g.org

[29] *OMNeT++*. Accessed: May 2023. [Online]. Available: https://omnetpp.org

[30] *INET Library*. Accessed: May 2023. [Online]. Available: https://inet.omnetpp.org

[31] C. Sommer, D. Eckhoff, A. Brummer, D. S. Buse, F. Hagenauer, S. Joerer, and M. Segata, "Veins: The open source vehicular network simulation framework," in *Recent Advances in Network Simulation* (EAI/Springer Innovations in Communication and Computing), A. Virdis and M. Kirsche, Eds. Cham, Switzerland: Springer, 2019.

[32] *Simu5G GitHub Repository*. Accessed: Jun. 2023. [Online]. Available: https://github.com/Unipisa/Simu5G/tree/Plaas_framework

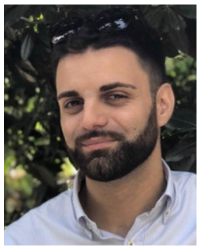
[33] *Study on Channel Model for Frequencies From 0.5 to 100 GHz*, document TR 38.901, Version 16.1.0, 3GPP, Jan. 2020.

[34] R. Rajamani, H.-S. Tan, B. Kait Law, and W.-B. Zhang, "Demonstration of integrated longitudinal and lateral control for the operation of automated vehicles in platoons," *IEEE Trans. Control Syst. Technol.*, vol. 8, no. 4, pp. 695–708, Jul. 2000, doi: 10.1109/87.852914.

[35] *Service Requirements for the 5G System*, document TS 22.261, Version 16.14.0, 3GPP, Apr. 2021.

[36] *Localisation and Sensing Use Cases and Gap Analysis*, Hexa-X Deliverable 3.1, Dec. 2021. [Online]. Available: https://hexa-x.eu/wp-content/uploads/2022/02/Hexa-X_D3.1_v1.4.pdf

[37] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun, "CARLA: An open urban driving simulator," in *Proc. 1st Conf. Robot Learn. (CoRL)*, Oct. 2017, pp. 1–16.



GIOVANNI NARDINI (Member, IEEE) received the Ph.D. degree in information engineering from the University of Pisa, in 2017. He is currently an Assistant Professor with the University of Pisa. His research interests include resource allocation algorithms in 4G/5G/B5G networks, multi-access edge computing, simulation, and performance evaluation of computer networks. In these fields, he has coauthored six patents and more than 50 papers. He has been involved in EU-funded and industrial research projects. He is serving as an Editorial Board Member and a member of the Technical Committee for the *Computer Communications* (Elsevier) and the *Wireless Networks* (Springer) as an Editor. He has served as the TPC Chair for IEEE SmartSys 2023.

ALESSANDRO NOFERI received the M.Sc. degree in computer engineering from the Department of Information Engineering, University of Pisa, Italy. He has been a Research Fellow with the Department of Information Engineering, until 2022. He is currently with Rete Ferroviaria Italiana (RFI).

GIOVANNI STEA received the Ph.D. degree from the University of Pisa, Italy, in 2003. He is currently a Full Professor with the Department of Information Engineering, University of Pisa. His current research interests include quality of service and resource allocation in networks, performance evaluation, and multi-access edge computing. In these fields, he has coauthored more than 120 peer-reviewed papers and 17 patents. He has been involved in national and EU research projects and he has led joint research projects with industrial partners. He has served as a member of the Technical and/or Organizing Committee for several international conferences, including SIGCOMM, WCNC, and VTC.

Open Access funding provided by 'Università di Pisa' within the CRUI CARE Agreement