Closer than Close: MEC–Assisted Platooning with Intelligent Controller Migration

Constantine Ayimba*
IMDEA Networks Institute and University Carlos III
Madrid, Spain

Michele Segata
Faculty of Computer Science, Free University of Bolzano
Bolzano, Italy

Paolo Casari
DISI, University of Trento
Trento, Italy

Vincenzo Mancuso
IMDEA Networks Institute
Madrid, Spain

ABSTRACT

The advent of multi access edge computing (MEC) will enable latency-critical applications such as cooperative adaptive cruise control (also known as platooning) to be hosted at the edge of the network. MEC-based platooning will leverage the coverage of the cellular infrastructure to enable inter-vehicular communications, potentially overcoming crucial problems of vehicular ad-hoc networks (VANETs) such as non-trivial packet loss rates. However, MEC-based platooning will require the controller to be migrated to the most suitable positions at the network edge, in order to maintain low-latency connections as the platoon moves. In this paper, we propose a context-aware Q-learning algorithm that carries out such migrations only as often as is necessary, and thereby reduces the additional delays implicit in application migration across MEC hosts. When compared to the state-of-the-art approach named FollowME, our scheme exhibits better compliance of vehicle speed and spacing values to preset targets, as well as a reduced statistical dispersion.

CCS CONCEPTS

• **Theory of computation → Multi-agent reinforcement learning**; *Sequential decision making*; • **Networks** → *Network simulations*; Network resources allocation; Cloud computing.

KEYWORDS

Platooning, MEC, Q-Learning

ACM Reference Format:
Constantine Ayimba, Michele Segata, Paolo Casari, and Vincenzo Mancuso. 2021. Closer than Close: MEC–Assisted Platooning with Intelligent Controller Migration. In *Proceedings of the 24th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM '21), November 22–26, 2021, Alicante, Spain.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3479239.3485681

1 INTRODUCTION

Future transportation systems will need to be highly automated and coordinated in order to improve safety, enhance the utility of the transportation infrastructure, and reduce the environmental impact of vehicle use, among other goals [2, 3]. A key component of the tools that enable these goals is cooperative adaptive cruise control (CACC) or platooning [23]. Most current implementations focus on managing platoons as independent vehicular ad-hoc networks (VANETs). However, these implementations are susceptible to the limitations of the networking protocols that are ubiquitous in this area such as 802.11p [19]. These limitations include high packet loss rates, often occurring in bursts, in different types of relevant vehicular scenarios. Moreover, to maintain speed-independent spacing between vehicles in such platoons, it is necessary that the vehicles not only receive status beacons from their predecessors, but all vehicles also need to receive communications from the platoon leader [5]. For long platoons, keeping all vehicles connected to the platoon leader may be challenging, owing to turns or obstacles along the road, which may prevent line-of-sight communications [19]. This exacerbates losses and hinders the information transfer required to maintain the platoon.

With the advent of multi-access edge computing (MEC) for 5G systems, each vehicle in the platoon can have a low-latency connection to a controller hosted at the network edge. The use of scheduling and robust forward error correction in 5G means that, unlike contention-based protocols used in VANETs, packet losses are not as pronounced. However, delivery delays remain a key challenge [9]. There have been some recent efforts [11, 21] to explore how the MEC architecture can be leveraged for vehicle-to-infrastructure (V2I) platooning. However, some key challenges have not been adequately addressed, including: (*i*) the development of robust controllers that can tolerate high delays; (*ii*) how to deal with out-of-order reports from platoon members; and (*iii*) how to intelligently migrate the controller across edge nodes to minimize latency as the platoon moves.

In this work, we address these challenges by modifying the platoon controller to handle V2I communication issues and develop a context-aware Q-learning migration scheme that deploys the controller in the most suitable location. Q-learning is a model free reinforcement learning technique which learns the action-value function, Q, of a state by trial and error [22].

We thereby substitute a controller in each platoon member with a centralised one which can be migrated from one MEC host to another according to the action-value function learned as the platoon moves.

Concretely, the contributions of this paper are: (*i*) we design a migration agent for the centralized control of a platoon from the MEC, based on a Q-learning algorithm that, compared to previous approaches, incorporates context-awareness; (*ii*) we modify CACC to estimate and compensate for network latency and messages delivered out of order, which is generally not required for direct Vehicle-to-Vehicle (V2V) communications; (*iii*) we define and study how multiple Q-learning agents can cooperate for rapid policy convergence with zero synchronization overhead; (*iv*) we enhance and combine existing vehicular and network simulation frameworks, through which (v) we evaluate the performance of intelligent controller migrations for MEC-assisted platooning, in the presence of single or multiple Q-learning agents.

The remainder of this paper is organized as follows: Section 2 surveys related work; Section 3 describes changes to the CACC controller; Section 4 introduces the design of the Q-learning agents that make automatic MEC migration decisions; Section 5 discusses our performance evaluation method and results; finally, Section 6 concludes the paper.

2 RELATED WORK

Platooning *per se* and the possibility to control it from the MEC have already been addressed in the literature [11, 23]. Our interest focuses on how to make the controller aware of communication issues that can occur, and how to make the most of the fluid architecture of cellular edge networks, which offer multiple options where to run and possibly migrate the platoon controller.

Controllers. Work on controllers to achieve cooperative driving dates back to the PATH project [13]. Though robust, this controller fits a peer-to-peer ad hoc network with short delays and good discipline in the order of communications among vehicles. The controller proposed in [10] only requires communication between proximate members of the platoon. While relaxing the stringent requirement of the platoon leader communicating with all members, it imposes a speed-dependent spacing between vehicles. Other works such as [15] have made controllers more delay tolerant and robust to packet errors, by taking into account the topology of the platoon and by employing speed-dependent spacing as well as communication between the platoon leader and all members. This approach is markedly string-stable, but remains limited to small platoons, and may not scale well in a MEC-driven scenario. We enhance the well-known controller presented in [13] to account for varying communication delays and disorderly reporting from platoon members. These adaptations make the controller better suited for use in V2I MEC-enabled platooning.

Service migration. Many state-of-the-art service migration schemes exist for MEC deployments [14]. However, only a subset of these fit latency-critical applications. The authors of [20] propose a random start placement on the available nodes which is then refined by prediction based on collected performance metrics. In [4], the authors propose a cognitive edge computing architecture supporting a service migration scheme. The scheme relies on repeated evaluations of the quality of experience of the users as they move in a network. Owing to its focus on human users, this work does not address the strict latency requirements of platooning. The authors of [9] design a service placement algorithm leveraging Lyapunov optimization to decompose the problem. Each subproblem is then solved by Markov approximation. The resulting scheme tracks user mobility and locates the service at the MEC host that minimizes the delay and cost. Although this approach considerably improves latency, it does not tackle the typically time-varying computing capability of the MEC hosts. This is particularly important given the myriad services that a MEC node may host, which thereby affect the latency experienced. Our proposed scheme takes into account such variations and will therefore migrate the service, should a previously selected MEC host become unreliably slow.

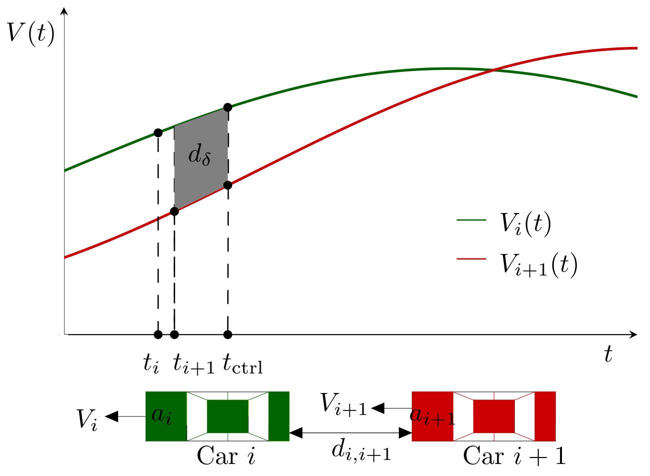


Figure 1: Controller delay compensation. At time t_i, car i generates the report packet with its own speed, acceleration and distance from vehicle $i-1$, and sends it to the MEC controller. Car $i+1$ does likewise at time t_{i+1}. At t_{ctrl} the controller collates the data.

3 CONTROLLER ADAPTATIONS FOR V2I PLATOONING

In legacy ad hoc platooning systems, the platoon leader communicates its speed and acceleration to each platoon member [16]. Each member also receives the speed and inter-vehicle spacing of its predecessor, typically from a radar system. With these data, the controller in each member calculates and applies the appropriate acceleration.

In the V2I-assisted platooning case, all vehicles need only communicate their own speed, acceleration and distance from the preceding vehicle to the controller resident in the MEC host. Figure 1 illustrates the case of two cars back-to-back in a platoon, whose data is used at the platoon controller with some random delay with respect to when the cars generate the respective reports. Indeed, network latency and the variation thereof over time strongly impact this process.

As discussed in [7], the architecture of the network greatly influences the latency of a service. Given the proliferation of small cells and network densification expected in 5G, handover delays will be of particular importance in determining the performance of 5G services. Handover latency values in the order of a few milliseconds can be tolerated. Further, owing to the fact that uplink transmission opportunities are subject to channel-dependent scheduling, there exists a non-trivial delay between the moment a packet is available for transmission at the platoon member and the time it reaches the MEC controller. Therefore, the data in this packet will be slightly stale, particularly in regard to speed and inter-vehicle spacing. Instead, we can safely assume that acceleration values will remain coherent within a transmission window.

3.1 Controller operation adaptations

When porting CACC to the V2I context, we modify it to improve controller operation and cope with two key issues. Depending on when the cellular network grants a transmit opportunity to a platoon member, either or both of the following may occur:

(1) consecutive packets from the same vehicle reach the controller in quick succession, owing to a delay that spans the interval of more than one periodic update,
(2) a packet generated at an earlier time by a leading vehicle reaches the controller later than that of a following vehicle.

In the case 1, we implement a filter that keeps only the latest packet from a given vehicle, as it represents the most updated information. In case 2, as depicted in Figure 1, we implement a data collection window to receive the packets from all members of the platoon. The controller then uses these data to compute the acceleration directive for each vehicle.

3.2 Latency compensation in the control law

The MEC controller has to account for the discussed latency values before computing the acceleration directives. Thus, we design the controller so as to update the speed assuming the previously assigned acceleration for the vehicle and update the spacing by estimating the distance travelled by the vehicle and by its corresponding predecessor. Given that these lags are in the order of (up to several tens of) milliseconds, these updates can be approximated as piece-wise linear functions as depicted in Figure 1. In particular, the speeds of vehicles i and its follower $i+1$ at control epoch t_{ctrl} can be computed based on speed and acceleration values sent with their freshest updates, at times t_i and t_{i+1}, respectively:

$$V_i(t_{\text{ctrl}}) \approx a_i(t_i) \cdot (t_{\text{ctrl}} - t_i) + V_i(t_i) \tag{1}$$

$$V_{i+1}(t_{\text{ctrl}}) \approx a_{i+1}(t_{i+1}) \cdot (t_{\text{ctrl}} - t_{i+1}) + V_{i+1}(t_{i+1})\,. \tag{2}$$

With the update generated at time t_{i+1}, the distance $\widehat{d}_{i,i+1}$ between the two vehicles is then estimated as:

$$\widehat{d}_{i,i+1} = d_{i,i+1} + \left(\int_{t_{i+1}}^{t_{\text{ctrl}}} V_i(t)\, dt - \int_{t_{i+1}}^{t_{\text{ctrl}}} V_{i+1}(t)\, dt \right)\,. \tag{3}$$

4 Q-LEARNING AGENTS FOR CONTROLLER MIGRATION

As a platoon moves, the distance between the vehicles and the controller will change. Even if we resort to network slicing and assign dedicated resources to the platoon controller, at some point such controller may have to be migrated closer to the platoon in order to guarantee responsiveness. For this, we design a migration agent and assume that its actions are either to move the controller to a candidate MEC node or to leave it in its current location. With high probability, the chosen action will influence changes in communication delays impacting the delivery of acceleration directives to the platoon members. Consequently, the speed and the spacing between platoon members will be impacted.

By defining the state as a combination of the processing capacity of a MEC host and of the platoon topology (relative spacing between platoon members), we can approximate the system as a state machine. In the following, we refer to the system as the "environment," in order to align with common reinforcement learning jargon [18]. The action of the migration agent will cause a state transition with some probability. We can therefore formulate the migration problem as a Markov Decision Process (MDP).[1]

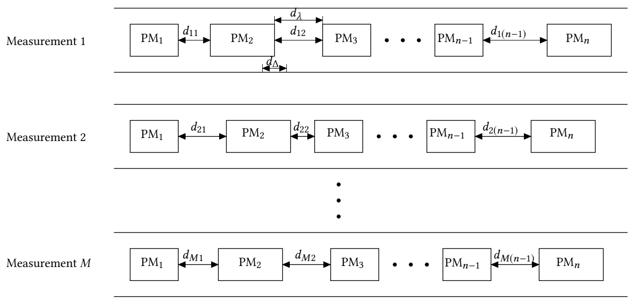


Figure 2: Platoon metrics as reported to the MEC Host. "PM" denotes a platoon member; d_λ is the required vehicle spacing; d_Δ is the spacing tolerance.

4.1 Data for state context definition

A schematic showing the measurements received at the MEC is illustrated in Figure 2. The controller communicates at regular intervals to the vehicles to issue directives for braking and acceleration. The directive to migrate the controller from one MEC host to another takes place less frequently, hence several measurements of the platoon are received between two such directives. These measurements are used by the mobile edge application orchestrator (MEAO) to obtain platoon-specific state variables (or "context") for the migration agent We derive the *platoon topology context* $\{\Gamma^{(-)}, \Gamma^{(+)}\}$ from the measurements depicted in Figure 2, with M measurements and n cars, as:

$$\gamma^{(-)} = \frac{1}{M(n-1)} \sum_{i=1}^{M} \sum_{j=1}^{n-1} \left\lfloor \frac{d_\lambda - d_{ij}}{d_\Delta} \right\rfloor \text{ for } d_{ij} \leqslant d_\lambda \tag{4}$$

$$\Gamma^{(-)} = \frac{\gamma^{(-)}}{1+\gamma^{(-)}} \tag{5}$$

$$\tag{6}$$

[1]We remark that other causes of delay may exist which are independent of migration. In this sense, our MDP is partially observable.

$$\gamma^{(+)} = \frac{1}{M(n-1)} \sum_{i=1}^{M} \sum_{j=1}^{n-1} \left\lfloor \frac{d_{ij} - d_\lambda}{d_\Delta} \right\rfloor \text{ for } d_{ij} \geqslant d_\lambda \tag{7}$$

$$\Gamma^{(+)} = \frac{\gamma^{(+)}}{1 + \gamma^{(+)}} \,. \tag{8}$$

Here, d_λ is the platooning target for the inter-vehicular spacing d_{ij}, and d_Δ is the spacing tolerance. The quantities $\gamma^{(-)}$ and $\gamma^{(+)}$ measure average deviations from the target, relative to tolerance, and group negative and positive deviations, respectively. Finally $\Gamma^{(-)}$ and $\Gamma^{(+)}$ are normalized versions of average relative deviations, which guarantee values in the range between 0 and 1, 0 being the ideal target. In particular, $\Gamma^{(-)}$ tells how well (or how bad) the controller is maintaining a safe distance, whereas $\Gamma^{(+)}$ signals whether the controller is accruing or losing the benefits of platoon compactness.

Besides, in order to make informed migration decisions, we need to obtain the network specific aspect of the context, so we estimate the delay in V2I communications. Specifically, there are two delay components due to data delivery and migration overhead.

The estimated data delivery time, T_{ddt} is calculated as:

$$T_{\text{ddt}} = T_{\text{net}} + T_{\text{proc}} \,, \tag{9}$$

where T_{net} is the estimated time for data to traverse the network and T_{proc} is the estimated time taken by the platoon controller to calculate the driving directives for the platoon members. T_{proc} depends on the capabilities of the MEC host and can be more reliably measured than T_{net}. In fact, T_{net} is influenced by multiple factors including the capacity of the wireless channel, the number of hops through routers in the wired network, and congestion on the switched links therein.

The overhead, T_{ovh}, includes the time it takes to migrate the VNF from one host to another, $T_{\text{migration}}$, and the time it takes to signal the platoon members about the new location to communicate to, $T_{\text{signaling}}$:

$$T_{\text{ovh}} = T_{\text{signaling}} + T_{\text{migration}} \,. \tag{10}$$

In case no migration takes place, $T_{\text{ovh}} = 0$.

Given a maximum permissible delay budget, T_{budget}, the resulting reference time ratio, T_{R} is calculated as:

$$T_{\text{R}} = \frac{T_{\text{ddt}} + T_{\text{ovh}}}{T_{\text{budget}}} \,. \tag{11}$$

Any candidate MEC positions yielding estimates such that $T_{\text{R}} \geqslant 1$ are not considered for migration.

Further, given that the response times of a given MEC host will vary depending on how busy it is over a given period, the change in processing time in successive epochs, T_Δ, is a crucial decision variable.

$$T_\Delta = 2\frac{T_{\text{proc}}^{(t)} - T_{\text{proc}}^{(t-1)}}{T_{\text{proc}}^{(t)} + T_{\text{proc}}^{(t-1)}}, \tag{12}$$

where (t) represents the current epoch defined as the interval $[t-1, t)$ and $(t-1)$ represents the previous epoch defined as the interval $[t-2, t-1)$. We set the duration of each epoch as 20s.

Each candidate MEC with $T_R < 1$ presents a migration option characterised by a given relative migration delay, $\theta = T_{\text{migration}}^{\text{candidate}} / T_{\text{budget}}$, and relative processing capability, $\beta = T_{\text{proc}}^{\text{candidate}} / T_{\text{proc}}^{\text{current}}$. The tuple of θ and β serves to contextualize the migration option along with the change in processing time T_Δ. We quantize θ, β and T_Δ into discrete steps.

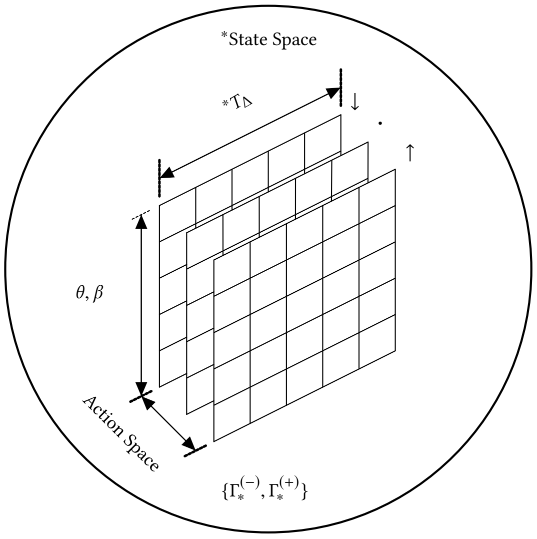


Figure 3: Migrator State and Action Spaces. The relative migration delay and relative candidate MEC power tuple $\{\theta, \beta\}$, the change in processing time between successive epochs, T_Δ and the platoon topology given by the tuple $\{\Gamma^{(-)}, \Gamma^{(+)}\}$ constitute the state space. The actions "Remain in the current MEC host" and "Migrate to any MEC host characterized by $T_R < 1$" form the action space.

The state of the environment is therefore fully specified by the values of $\{\Gamma^{(-)}, \Gamma^{(+)}\}$ computed from the last measurement set, and the values of θ, β and T_Δ for all MEC hosts in the environment. Possible actions for the migration agent are restricted to keep the controller in the current MEC host or migrate it to any MEC host with $T_R < 1$.

4.2 Q-learning

Q-learning is a Reinforcement Learning (RL) technique in which the agent approximates the optimal action value without prior knowledge of the Markov Decision Process (MDP) that underlies its environment. When the environment is in state S, the agent takes action a, obtains the reward R and the environment transitions to state S' [18]. Therefore, we have

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left(R + \gamma \max_a Q(S', a) - Q(S, A)\right), \tag{13}$$

where $\gamma \in [0, 1]$ is a discounting factor that weighs the contribution future rewards will have when starting in state S', and $\alpha \in (0, 1]$ is the learning rate, and makes it possible to tune the pace at which the policy converges towards the expected action-value $Q(S, A)$. If all states are visited with equal probability, α can be chosen as a fixed parameter. However, given that the agent will realistically observe only a subset of the states, we keep track of the number of times k that the agent was in state S and took action A and define

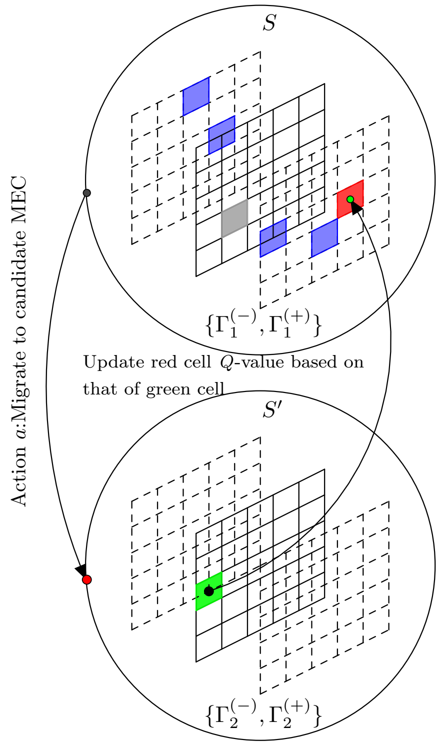

S
$\{\Gamma_1^{(-)}, \Gamma_1^{(+)}\}$
Update red cell Q-value based on that of green cell
S'
$\{\Gamma_2^{(-)}, \Gamma_2^{(+)}\}$
Action a:Migrate to candidate MEC

Figure 4: The Q-learning update process of the migration agent. The gray cell represents the Q-value of the MEC currently hosting the controller. The red cell represents $Q(S, A)$, the Q-value of the MEC host chosen to host the controller from the context of the current host and platoon topology specified by $\{\Gamma_1^{(-)}, \Gamma_1^{(+)}\}$. The green cell represents $Q(S', a)$: the Q-value of the chosen MEC in the eventual context specified by the platoon topology $\{\Gamma_1^{(-)}, \Gamma_1^{(+)}\}$. The blue cells represent Q-values of the other alternative MEC hosts not chosen for the migration.

$\alpha = \frac{1}{k}, k > 0$. As will become clear later, this choice also facilitates the weighing of the Q-values when multiple agents share their experiences in the asynchronous shared learning we use to expedite policy convergence. Actions are decided after a time window termed an epoch. The set of epochs from the beginning to when the environment encounters a terminal state, where no further actions can be taken, constitute an episode.

We also employ ϵ-greedy action selection, whereby an action is chosen randomly with probability ϵ. We initially set $\epsilon = 1$ for each state, and reduce this value progressively (down to $\epsilon_{min} = 0.01$) as more visits are made to the state. This encourages exploration in the initial phases to discover the most rewarding actions and exploitation in the latter stages to make use of the policies learnt. At convergence, action selection is largely on policy and the algorithm chooses the action procuring the highest reward.

Leveraging some aspects of our previous work on short-term memory Q-learning [1], we design a migration agent that exploits context-awareness. The agent first retrieves the platoon spacing data to derive the context as specified in (5) and (8). It then obtains a listing of the available MEC hosts and estimates the migration time to the new host, either based on any previous logs of a similar migration, or based on a default value if no recent migration targeted that host. The agent also obtains the processing statistics of the MEC hosts in order to estimate their capacity with reference to the current host. With these data it calculates the quantized levels of θ, β and T_Δ for each MEC host. The latter parameters identify the Q-values of the migration options in the decision structure shown in Figure 3. Comparing these options, according to (13), the agent makes an ϵ-greedy decision as to which MEC host to migrate to as shown in Figure 4.

4.3 Reward Functions

A key element in an MDP is the reward function which serves as a feedback to the agent to evaluate how suitable or unsuitable a decision was. The reward function we design and implement, given the cyber-physical nature of platooning, consists of two components: the network related and the vehicular positioning reward. These rewards are calculated at every epoch. Furthermore the network component takes into account the delivery intervals of control packets and a small penalty, ζ, for controller migration. The latter is to dissuade unnecessary migrations, which may perturb the platoon owing to the resulting delays.

We now consider the network component of the reward as regards packet delivery intervals. Our controller deems a packet urgent and therefore require expedited delivery if ***all*** of the following conditions are fulfilled: (*i*) its current estimate of the spacing between the recipient vehicle and its immediate predecessor is smaller than the inter-vehicle gap d_λ; (*ii*) the current speed of the recipient is higher than the speed of its immediate predecessor; (*iii*) the previous estimate of the spacing between the recipient vehicle and its immediate predecessor was smaller than the platoon gap; and (*iv*) the previous speed of the recipient vehicle was higher than that of its immediate predecessor. If the latter conditions are not fulfilled then the packet is deemed normal. If X_{urgent} and X_{normal} are the number of packets deemed urgent and normal over an epoch, respectively, the resulting network reward component is

$$R_{\text{net}} = \frac{X_{\text{normal}}}{X_{\text{urgent}} + X_{\text{normal}}} - \zeta \, . \tag{14}$$

To evaluate the second component, which accounts for platoon spacing rewards, we define the following sets and quantities in analogy to what done for the topology context with $\{\gamma^{(-)}, \gamma^{(+)}\}$ and $\{\Gamma^{(-)}, \Gamma^{(+)}\}$:

$$\begin{aligned}
\mathcal{D} &:= \{d_{ij}\}_{i=1,j=1}^{i=M,j=n-1} \\
\mathcal{D}_\epsilon &:= \{d \in \mathcal{D} \ : \ d < d_\lambda - d_\Delta\} \\
\mathcal{D}_\Omega &:= \{d \in \mathcal{D} \ : \ d > d_\lambda + d_\Delta\} \\
\mathcal{D}_0 &:= \{d \in \mathcal{D} \ : \ |d - d_\lambda| \leqslant d_\Delta\}
\end{aligned}$$

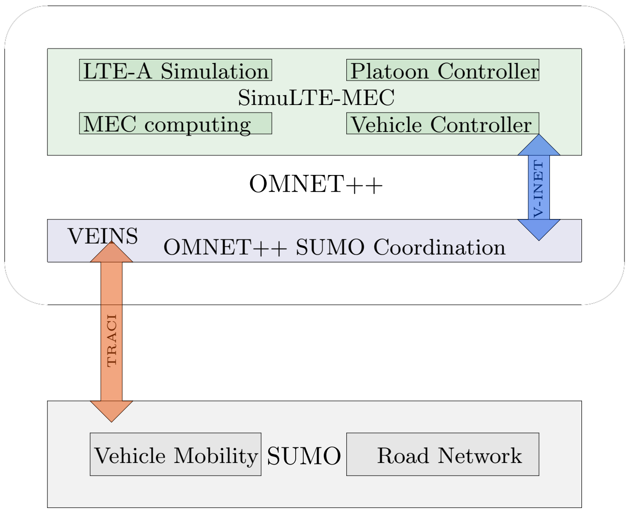


Figure 5: Simulation frameworks

$$r_\epsilon = \frac{1}{|\mathcal{D}_\epsilon|} \sum_{d_{ij} \in \mathcal{D}_\epsilon} \left| \frac{d_\lambda - d_{ij}}{d_\Delta} \right| \tag{15}$$

$$R_\epsilon = \frac{r_\epsilon}{1 + r_\epsilon} \tag{16}$$

$$r_\Omega = \frac{1}{|\mathcal{D}_\Omega|} \sum_{d_{ij} \in \mathcal{D}_\Omega} \left| \frac{d_\lambda - d_{ij}}{d_\Delta} \right| \tag{17}$$

$$R_\Omega = \frac{r_\Omega}{1 + r_\Omega} \tag{18}$$

$$r_0 = \frac{1}{|\mathcal{D}_0|} \sum_{d_{ij} \in \mathcal{D}_0} \left| \frac{d_\lambda - d_{ij}}{d_\Delta} \right| \tag{19}$$

$$R_0 = 0.5 - \frac{r_0}{1 + r_0} \tag{20}$$

Here, we have tri-partitioned inter-vehicular distances into the sets of distances within a given target interval $d_\lambda \pm d_\Delta$, or below that interval, or above. In addition, we define a safety indicator that tells whether the spacing between two cars is dangerously below the target, i.e., below $d_{\text{safe}} \ll d_\lambda$:

$$R_{\text{safe}} = \begin{cases} 0 & \text{if } d_{ij} \geqslant d_{\text{safe}} \ \forall d_{ij} \in \mathcal{D} \\ 1 & \text{otherwise} \end{cases} \tag{21}$$

With the above definitions, we can then compute the vehicular positioning component of the reward as:

$$R_\lambda = |\mathcal{D}_\epsilon| w_\epsilon R_\epsilon + |\mathcal{D}_\Omega| w_\Omega R_\Omega + |\mathcal{D}_0| w_0 R_0 - |\mathcal{D}| R_{\text{safe}}, \tag{22}$$

where $w_\epsilon, w_\Omega, w_0$ are weights assigned so as to prioritise either safety or spacing of the platoon and d_{safe} is the minimum spacing below which a crash is likely to occur. The total reward R, accrued in one epoch, is then calculated as:

$$R = |\mathcal{D}| R_{\text{net}} + R_\lambda. \tag{23}$$

A training episode, characterised by several consecutive epochs, terminates with the platoon successfully completing a given road segment (e.g., a lap in a circuit).

4.4 Asynchronous Shared Learning

We now propose an extension to the above framework that enables shared learning across different platoons. This extension makes it possible to leverage the state exploration in different platoons and thus explore the state space more extensively in less time.

We assume that multiple platoons exist and visit the same road segments. Each platoon controller runs on an independent slice in the MEC host. As such, each platoon could be transparent to the others. However, we further assume that the migration agent of each platoon can read and write a common migration policy file, from where it fetches its own policy. The agent can also update the policy by annotating what it learns. Each migration agent can only access this shared file every so often, so that we can neglect the overhead associated with sharing the experience learned by other platoons.

In this scheme, agents do not need to be synchronized, and rather could access the migration policy at any time. Between two consecutive accesses to the shared policy file, each agent keeps its own version of the file, with updates on visited state's Q-values that will be only later reflected in the shared policy.

The scheme described above can model the behaviour of a distributed learning process, in which a central entity asynchronously collates updates from all agents, e.g., when they connect to a given eNodeB, or when they migrate to a specific MEC host. The scheme can also be extended to become virtually overhead-free and distributed, if we assume that each MEC host maintains a migration policy file, and that platoon controllers migrating to a MEC host bring in all their past learned policy values. This way, the policy built at a MEC host can be spread to the other MEC hosts by simply being transferred jointly with controllers during migrations. We argue that this asynchronous scheme is potentially effective to speed up the convergence of learned migration policies.

5 PERFORMANCE EVALUATION

5.1 Method

Our simulation testbed comprises three main components as shown in Figure 5. The first is the robust vehicular simulator SUMO from the German Aerospace Agency DLR [6]. It is widely used in research given its highly realistic depiction of vehicular mobility. The second component is the OMNeT++ Framework “Vehicles in Network Simulation (VeINS)” [17]. This framework provides an API that facilitates the coordination of the vehicle simulation in SUMO with the network simulation of corresponding UE in OMNeT++. The third component of the simulator is SimuLTE-MEC [8], an OMNeT++ framework that realistically simulates the LTE-Advanced network with mobile edge computing extensions.

We implement the migration scheme as a python script that monitors the logs generated by OMNeT++ as the platoon traverses its course. The script uses the logs as input to the Q-learning migration scheme. It also monitors the simulation time and, at regular intervals, updates a reference file with the selected MEC host. In the process of generating a vehicular report to send to the MEC host, SimuLTE reads the reference file. If there is a change, it triggers a migration from the current location to the selected MEC host. Table 1 lists the key parameter choices we make in SimuLTE to better capture the envisaged 5G network capabilities. In particular, we set the handover delay to random values below 10 ms. A handover occurs when a vehicle receives 3 successive signals from a target eNodeB that have a higher RSSI compared to the one that is currently serving it. Table 1 also lists the parameters used to to compute the reward function of platooning. The chosen weights are as such that the negative reward (i.e., the penalty) of being 20% below the target spacing or 50% above is the same impact as for a migration. The rationale is that we have observed up to about 20% of variation in speed profiles with migrations, which is comparable with driving 20% below the target for what concerns safety, and with the platoon occupying about 20% more road space when driving at 15 m instead of 10 m, although the exact value depends on platoon size.

We curtail fading aspects of the channel given that network densification with small cells will lead to higher probability of line-of-sight communications. We enhance the computing infrastructure of the MEC hosts with FIFO queues, in order to simulate the variability of processing delays due to competing third party background processes. In addition we introduce queuing delays on the switching elements to account for routing delays that occur when the MEC node hosting the controller is not directly connected to the eNodeB serving a vehicle. Figure 6 depicts the road circuit that we use in SUMO to delimit a training episode, along with the communication network in SimuLTE. We remark that our scheme is independent of the placement of the MEC hosts within the network (whether closer to the eNodeB or to the core) given that it distinguishes them based on processing capacity and migration delay. The simulation time at location A is 31s, the subsequent intervals after that are 20s each. These are the epochs at which the migration agent makes its decisions. Each eNodeB has a MEC node attached to it such that the controller may be hosted at those locations. The background traffic of the MEC nodes, independent of the platooning load, is

Table 1: Simulation Parameters

Parameter	Value
Path loss model	ITU Rural macro cell
eNodeB antenna gain	18 dB
eNodeB height	25 m
eNodeB transmit range	500 m
eNodeB transmit power	43 dBm
Size of uplink and downlink packet	39 Bytes
Block error rate	1%
Handover latency (per vehicle)	$X \sim \mathcal{U}(0, 10)$ ms
Core routing delay (per packet)	$X \sim \mathrm{Exp}(1)$ ms
X2 hop delay (per packet)	50 μs
Migration delay (per vehicle)	$X \sim \mathcal{U}(1, 3)$ ms
Migration epoch interval	20 s
Vehicular reporting interval	100 ms
Platoon speed (Amplitude)	21.5 m/s - 28.9 m/s
Platoon speed (Frequency)	0.1 Hz
Number of cars (n)	30
Platoon spacing (d_λ)	10 m
Migration penalty (ζ)	0.05
d_{safe}	1 m
d_Δ	0.5 m
w_ϵ	-0.5
w_Ω	-0.1
w_0	0.25

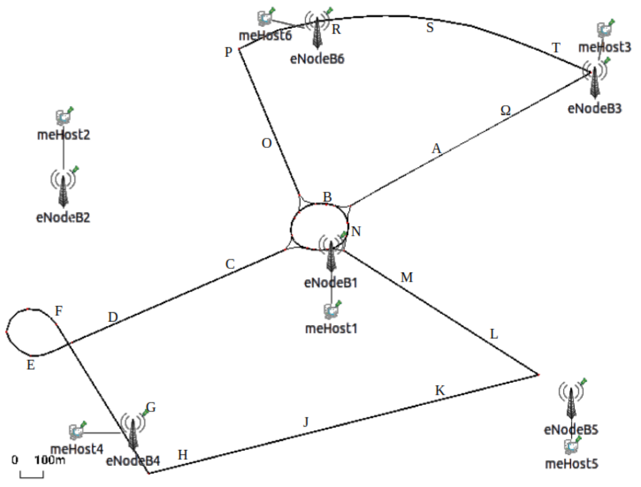


Figure 6: Road and MEC Network. The scale corresponds to the road network, eNodeBs and MEC hosts are exaggerated to make them discernible.

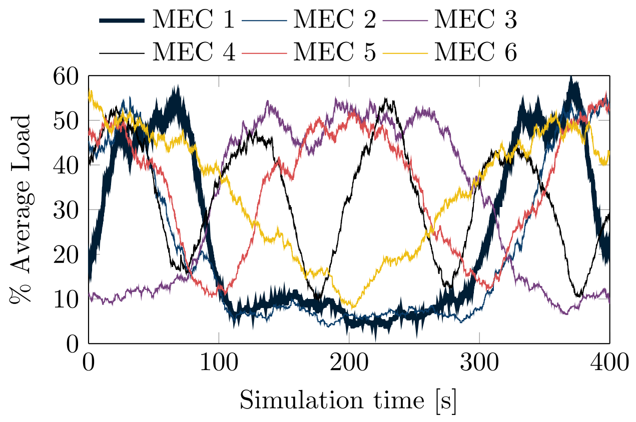


Figure 7: MEC host load due to background traffic.

depicted in Figure 7. We chose these patterns to represent realistic workload traces with different periodicity. We remark that none of the patterns is in sync with the time the platoon needs in order to complete a loop along the circuit of Figure 6.

5.2 Asynchronous Shared Learning Extension

To test the potential of the asynchronous shared learning scheme described in Section 4.4, we use parallel simulations of platoons of cars lapping through a same circuit. We test the centralized version of our scheme, with each platoon accessing the shared migration policy every 5 epochs.

At initialization, the parent platooning simulation directory is cloned into a number of directories commensurate to the number of parallel agents chosen. However, the overall policy file is placed in a directory accessible to all participating agents. After cloning, a

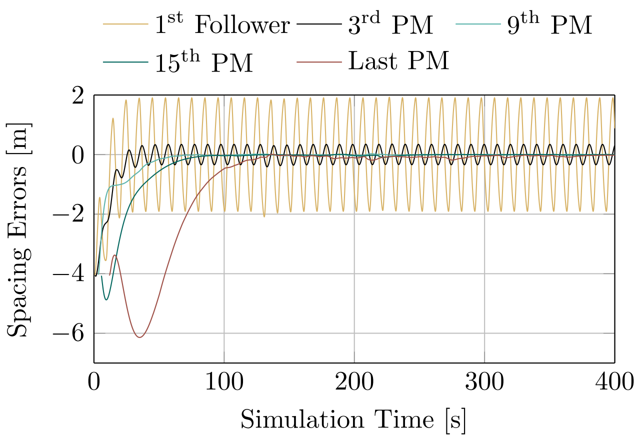


Figure 8: Spacing errors of platoon members (negative means farther).

shell script triggers the start of the SUMO/OMNeT++ simulation as well as that of the migration script in each directory.

When the migration agent starts, it reads the overall policy file into a data structure and proceeds to update it as it makes its decisions. The start time of the simulation is different in each directory, in order to mirror the separation of the platoons in the real world. After about 100 s (which corresponds to about five migration decisions), the migration agent locks the policy file and reads the overall policy into its data structure, so as to capture any updates by the other participating agents. The agent then proceeds to update the data structure with Q-values and immediate rewards of just the states it visited as prescribed in Eq. (13). It then overwrites the policy file with the updated data structure. Finally, it re-reads the overall policy into its data structure and unlocks the policy file. Should an agent find the file locked by another, it waits for a random time and then retries until the file is unlocked.

5.3 Evaluation results

In this section we examine the results from the modifications on the controller. We then consider platoon performance resulting from the use of our migration scheme, compared to that of the state of the art scheme proposed in [9].

5.3.1 Controller Modifications. We first examine the platoon formation when the controller is hosted on the MEC. In order to set the most challenging condition for the controller [15], we add a sinusoidal perturbation to the movement speed of the platoon leader. This sinusoidal driving pattern also accounts for speed variations as the platoon may need to slow down or speed up depending on driving conditions. The platoon speed oscillates between 21.5 m/s (77 km/h) and ≈29 m/s (104 km/h). The 30 vehicles of the platoon fully stabilize in about 100 s, as shown in Figure 8. The spacing between the first follower and the platoon leader exhibits the highest variation, as expected. The effect on the rest of the followers is progressively damped down towards the tail of the platoon. From this result, we conclude that our modifications on the MEC-hosted controller preserve string stability [12] and thereby ensure platoon safety.

5.3.2 Migration Strategies. The proximity to an eNodeB and the load on a given MEC host may be contrasting objectives for the migration agent to pursue. For example, the agent may run on the MEC host connected to the eNodeB closest to the platoon (hence it perceives a low air interface communication latency). However, if the load on this MEC host increases, the agent may need to migrate to a different MEC host, and the resulting routing and switching delays may nullify the advantage of being connected to a near eNodeB. Another important consideration is that the migrations should be kept at a minimum given the extra delays involved in switching from one MEC host to another.

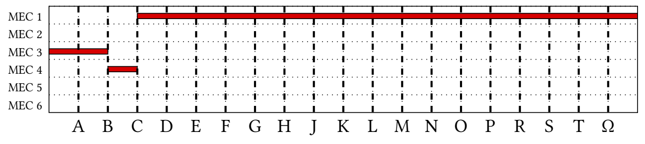
A subset of the migration strategies obtained by our algorithm are depicted in Figure 9. These sequences were the most recurrent in which every visited state was fully converged (i.e., $\epsilon = \epsilon_{\min}$). We omitted less frequently observed sequences for brevity. By design, we initially position the controller at MEC 3 (towards the top-right section of the track). This avoids any biases that might give undue advantage to one migration policy over the other.

Our algorithm learns policies that exploit both the proximity and the computing capability of the MEC host. This reflects in the agent migration patterns, which initially involve different MEC servers, but then prefer stabilizing the controller on the same server despite the server load and the additional routing delays. Notably, different policies attain the same conclusions, and elect a MEC server on which they remain until the end of our simulations. In contrast, the state-of-the-art Follow ME algorithm [9], only considers proximity when selecting the preferred MEC host, and forces numerous migrations, as shown in Figure 10. Yet, these migrations may prove inconvenient, as we discuss next.

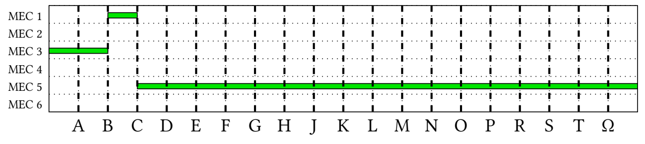
5.3.3 Platoon stability. Considering the same sinusoidal driving pattern employed so far, Figure 11 shows that the speed of the first follower adapts very well to that of the platoon leader. This takes place despite the sinusoidal perturbations on the platoon leader speed, and confirms that the learned migration policies show very good robustness. In comparison, Follow ME [9] exhibits much greater variability. This points to the effectiveness of the minimal migrations that our algorithm decides to perform.

When compared to Follow ME, our scheme exhibits better platoon spacing discipline. This is shown in Figure 12, which employs box-plots to convey the distribution of vehicle spacing across the platoon at different simulation times. Each blue box extends from the 1st to the 3rd quartile of the distribution, the red bar denotes the median, and whiskers cover the 10th-90th percentile range. Red "+" markers denote the data along the tails of the distribution. From Figure 12, we observe that our Q-migration algorithm achieves smaller inter-quartile ranges (Figure 12a and Figure 12b) compared to those of Follow ME (Figure 12c). This implies higher string stability and a generally better driving experience. In particular, the occurrences of long whiskers are due to the the spacing between the first follower and the platoon leader, which varies the most under the sinusoidal motion of the leader.

5.3.4 Asynchronous Shared Learning. The acceleration of convergence through the use of asynchronous shared learning is apparent



(a) Q-migration Policy A



(b) Q-migration Policy B

Figure 9: Sample Q-migration policies

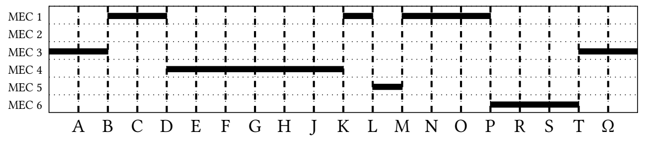


Figure 10: Follow ME migration policy [9]

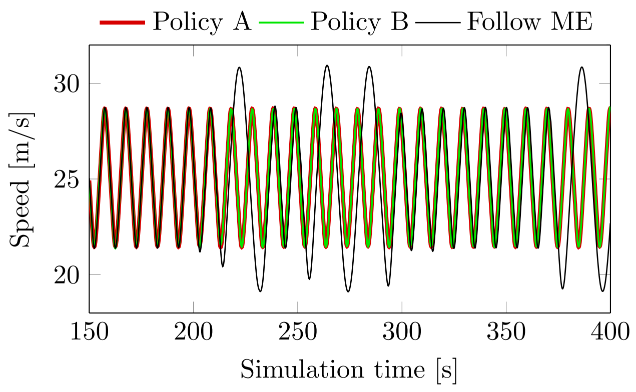
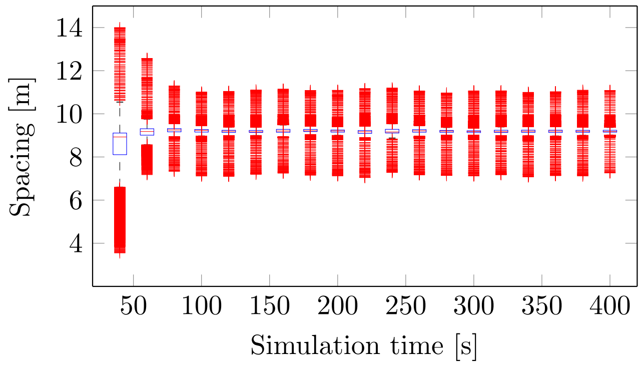


Figure 11: Speed profiles of the first follower

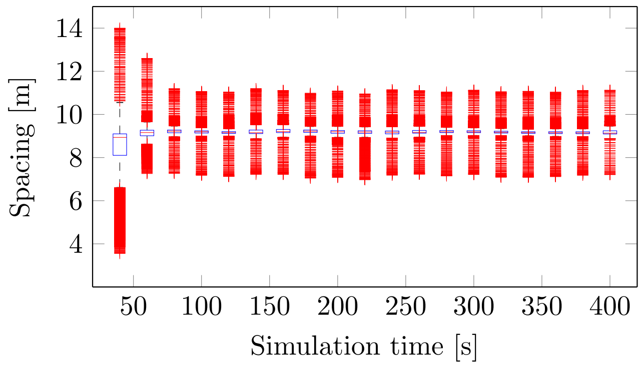
from Table 2. In this table, we take each consecutive set of parallel episodes, and we examine the log of the states that each agent visits. If $\epsilon = \epsilon_{\text{min}}$ at every epoch for any of the agents, we consider that episode as exhibiting convergence. The n^{th} time that this is observed is termed the n^{th} convergence. The results prove that having more agents sharing their own experience with other agents greatly increases the speed at which migration policies fully converge. However, after the initial significant gain, the advantage of having more parallel agents decreases. This points to a diminishing return in the value of parallel agents after the first convergence.

6 CONCLUSIONS AND FUTURE WORK

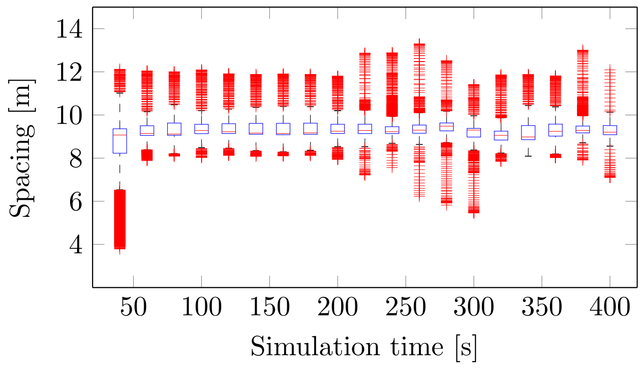
We presented a context-aware Q-learning-based migration algorithm for vehicle platoon controllers. The algorithm learns the appropriate strategies to migrate the controller from one MEC host to another in a cellular network context. Our approach reduces the number of migrations required for the controller to keep pace with the platoon and maintain it in formation. Our approach also makes it possible to easily and effectively leverage the presence of multiple platoons in order to learn migration policies faster. In particular, compared to the state-of-the-art scheme FollowME, our



(a) Q-migration Policy A



(b) Q-migration Policy B



(c) Follow ME

Figure 12: Distribution of spacing between platoon members. Each box plot represents data taken over 20 s windows. The last box plot represents data in the last 10 s.

Table 2: Parallel episodes until convergence

# Agents	1st Conv.	2nd Conv.	3rd Conv.	4th Conv.
10	27	28	29	31
5	63	67	75	77
1	389	482	495	499

approach complies better with vehicle speed and spacing presets, and achieves a reduced statistical dispersion of spacing values.

Because sub-optimal migration decisions may occur in the initial learning phases of our algorithm and may lead to string instability, we intend to explore controllers with speed-dependent spacing as a means to enhance safety during the training phase. We will also investigate the ideal number of agents that achieves the best trade-off between policy exploration and exploitation.

ACKNOWLEDGMENT

Project funded by the Spanish State Research Agency (AEI) PID2019-109805RB-I00/AEI/10.13039/501100011033, and by the Italian Ministry for University and Research (MIUR) under the initiative "Departments of Excellence" (Law 232/2016).
Michele Segata was partially supported by the Free University of Bolzano–Bozen under the SECEDA project, RTD Call 2021.

REFERENCES

[1] Ayimba, C., Casari, P., and Mancuso, V. SQLR: Short-term memory Q-learning for elastic provisioning. *IEEE Trans. Netw. Service Manag. 18*, 2 (2021), 1850–1869.

[2] Barmpounakis, S., Tsiatsios, G., Papadakis, M., Mitsianis, E., Koursioumpas, N., and Alonistioti, N. Collision avoidance in 5G using MEC and NFV: The vulnerable road user safety use case. *Computer Networks 172* (2020), 107150.

[3] Boban, M., Kousaridas, A., Manolakis, K., Eichinger, J., and Xu, W. Connected roads of the future: Use cases, requirements, and design considerations for vehicle-to-everything communications. *IEEE Veh. Technol. Mag. 13*, 3 (2018), 110–123.

[4] Chen, M., Li, W., Fortino, G., Hao, Y., Hu, L., and Humar, I. A dynamic service migration mechanism in edge cognitive computing. *ACM Trans. Internet Technol. 19*, 2 (Apr. 2019).

[5] Dressler, F., Klingler, F., Segata, M., and Lo Cigno, R. Cooperative driving and the tactile Internet. *Proc. IEEE 107*, 2 (2019), 436–446.

[6] Krajzewicz, D., Hertkorn, G., Rössel, C., and Wagner, P. SUMO (simulation of urban mobility) - an open-source traffic simulation. In *4th Middle East Symposium on Simulation and Modelling* (2002), A. Al-Akaidi, Ed., pp. 183–187.

[7] Lauridsen, M., Gimenez, L. C., Rodriguez, I., Sorensen, T. B., and Mogensen, P. From LTE to 5G for connected mobility. *IEEE Commun. Mag. 55*, 3 (2017), 156–162.

[8] Nardini, G., Virdis, A., Stea, G., and Buono, A. SimuLTE-MEC: extending SimuLTE for multi-access edge computing. In *Proc. OMNeT++ summit* (2018).

[9] Ouyang, T., Zhou, Z., and Chen, X. Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing. *IEEE J. Sel. Areas Commun. 36*, 10 (2018), 2333–2345.

[10] Ploeg, J., Scheepers, B., van Nunen, E., van de Wouw, N., and Nijmeijer, H. Design and experimental evaluation of cooperative adaptive cruise control. In *Proc. IEEE ITSC* (2011), pp. 260–265.

[11] Quadri, C., Mancuso, V., Ajmone Marsan, M., and Rossi, G. P. Platooning on the edge. In *Proc. ACM MSWiM* (2020).

[12] Rajamani, R. *Vehicle Dynamics and Control*, 2nd ed. Springer, 2012.

[13] Rajamani, R., Han-Shue Tan, Boon Kait Law, and Wei-Bin Zhang. Demonstration of integrated longitudinal and lateral control for the operation of automated vehicles in platoons. *IEEE Trans. Control Syst. Technol. 8*, 4 (2000), 695–708.

[14] Salaht, F. A., Desprez, F., and Lebre, A. An overview of service placement problem in fog and edge computing. *ACM Comput. Surv. 53*, 3 (June 2020).

[15] Santini, S., Salvi, A., Valente, A. S., Pescapé, A., Segata, M., and Lo Cigno, R. A consensus-based approach for platooning with intervehicular communications and its validation in realistic scenarios. *IEEE Trans. Veh. Technol. 66*, 3 (2017), 1985–1999.

[16] Segata, M., Bloessl, B., Joerer, S., Sommer, C., Gerla, M., Lo Cigno, R., and Dressler, F. Toward communication strategies for platooning: Simulative and experimental evaluation. *IEEE Trans. Veh. Technol. 64*, 12 (2015), 5411–5423.

[17] Sommer, C., German, R., and Dressler, F. Bidirectionally coupled network and road traffic simulation for improved IVC analysis. *IEEE Trans. Mobile Comput. 10*, 1 (2011), 3–15.

[18] Sutton, R. S., and Barto, A. G. *Introduction to Reinforcement Learning*, 2nd ed. MIT Press, Cambridge, MA, USA, 2020.

[19] Teixeira, F. A., e Silva, V. F., Leoni, J. L., Macedo, D. F., and Nogueira, J. M. Vehicular networks using the IEEE 802.11p standard: An experimental analysis. *Vehicular Commun. 1*, 2 (2014), 91–96.

[20] Velasquez, K., Abreu, D., Curado, M., and Monteiro, E. Service placement for latency reduction in the internet of things. *Proc. IEEE 72*, 2 (2017), 105–115.

[21] Virdis, A., Nardini, G., and Stea, G. A framework for MEC-enabled platooning. In *Proc. IEEE WCNCW* (2019), pp. 1–6.

[22] Watkins, C. J. C. H. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, May 1989.

[23] Öncü, S., Ploeg, J., van de Wouw, N., and Nijmeijer, H. Cooperative adaptive cruise control: Network-aware analysis of string stability. *IEEE Trans. Intell. Transp. Syst. 15*, 4 (2014), 1527–1537.