Driving Under Influence: Robust Controller Migration for MEC-Enabled Platooning*

Constantine Ayimba[a,b,*,1], Michele Segata[c,2], Paolo Casari[d,3] and Vincenzo Mancuso[a,4]

[a]IMDEA Networks Institute, Avenida del Mar Mediterráneo 22 , Leganés, 28918, Madrid, Spain
[b]University Carlos III of Madrid, Av. de la Universidad, 30, Leganés, 28911, Madrid, Spain
[c]Free University of Bozen-Bolzano, piazza Domenicani, 3, Bozen-Bolzano, 39100, Italy
[d]University of Trento, Via Sommarive, 9, Povo, 38123, Trento, Italy

ARTICLE INFO

Keywords:
Reinforcement Learning
Q-Learning
Platooning
MEC
Application Relocation
Application Migration

ABSTRACT

Connected cars are becoming more common. With the development of multi-access edge computing (MEC) for low-latency applications, it will be possible to manage the cooperative adaptive cruise control (CACC, also known as platooning) of such vehicles from the edge of cellular networks. In this paper, we present a controller that carries out platooning from the network edge by adapting to varying network conditions. We incorporate a mechanism in the controller that allows vehicles to switch to automated cruise control when delays exceed safety thresholds, and switches back to platooning when the delays are sufficiently low to support it. We also formulate the problem of maintaining a low-latency connection in the presence of high mobility through migration and propose a Q-Learning algorithm to solve this problem. We finally propose an Asynchronous Shared Learning scheme that enables multiple migration agents to cooperate, in order to expedite the convergence of migration policies. Compared to state-of-the-art migration techniques, our scheme exhibits better compliance of vehicle speed and spacing values to preset targets, and ameliorates statistical dispersion.

1. Introduction

Intelligent Transportation Systems (ITS) of the future aim to increase road throughput, improve safety and reduce emissions from vehicles [1, 2]. A key proposal of ITS is platooning, the coordinated driving of vehicles with short spacing between them. To realize platooning, legacy schemes have relied so far on vehicular ad hoc networks (VANETs). These networks are however susceptible to high packet losses, as they rely on contention based media access control [3]. Given the latency-sensitive nature of platooning [4], VANETs cannot therefore safely support large platoons.

Moreover, to maintain speed-independent spacing between vehicles in such platoons, not only is it necessary that the vehicles 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 [3]. This exacerbates losses and hinders the information transfer required to maintain the platoon.

With the advent of multi-access edge computing (MEC), each vehicle in the platoon can potentially 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 [6]. There have been some recent efforts [7, 8, 9] 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 [10].

It may also be possible to simultaneously deploy controllers on multiple MEC hosts. In this case, if the platoon roams into a given region of the network, the corresponding controller may thus take over. The drawbacks of this strategy are twofold: (*i*) given that only one controller can be active a time, the other controllers occupy resources needed by other edge services; and (*ii*) this strategy necessitates an overarching synchronization among controllers in order to maintain the platoon in formation as command is passed on. However, it may be feasible to implement the individual logical controller (required by each platoon) as a hierarchy of coordinated controllers, each responsible for a segment of the platoon, as in [11]. In this work, we focus on the fundamental problem of migrating the controller logic

*Corresponding author
constantine.ayimba@imdea.org (C. Ayimba); michele.segata@unibz.it (M. Segata); paolo.casari@unitn.it (P. Casari); vincenzo.mancuso@imdea.org (V. Mancuso)
https://scholar.google.com/citations?user=v-hf3kEAAAAJ&hl=en&oi=ao (C. Ayimba); http://michele-segata.it (M. Segata); https://webapps.unitn.it/du/it/Persona/PER0221586 (P. Casari); https://networks.imdea.org/team/imdea-networks-team/people/vincenzo-mancuso/ (V. Mancuso)
ORCID(s): 0000-0003-4032-9735 (C. Ayimba); 0000-0002-6016-749X (M. Segata); 0000-0002-6401-1660 (P. Casari); 0000-0002-4661-381X (V. Mancuso)

as vehicles move and MEC loads change. Given this, we only consider a non-distributed controller.

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 in which an agent learns the action-value of a state by trial and error initially before settling on the most rewarding action in the long run [12]. We thereby substitute a controller in each platoon member with a centralized one, which can be migrated from one MEC host to another according to the policies 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, unlike 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. Moreover, (*vi*) we introduce Slow down And spLiT (SALT), a safety overlay to the working of the MEC controller as a key additional contribution to our work presented in [13]. This novel technique progressively increases the gap between vehicles and reduces the overall speed as network and/or computing conditions deteriorate persistently. When the spacing is sufficiently safe, Automated Cruise Control is triggered to keep the vehicles moving in a convoy. When the vehicles enter an area where network and compute delays are sufficiently low, the platoon is reformed and progressively moves at the desired speed and spacing.

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 [4, 8]. Our interest focuses on making the controller aware of communication issues that can occur, and how to take advantage 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 [14]. 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 [15] only requires communications 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 [16] 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 [14] 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 [17]. However, only a subset of these fit latency-critical applications. The authors of [18] propose a random start placement on the available nodes which is then refined by prediction based on collected performance metrics. In [19], 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 [6] 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 host may host, which thereby affect the experienced latency.

This limitation is however considered by the authors of [20], who create an Adaptive User-managed Service Placement (AUSP) algorithm taking into consideration the compute delay, the communication delay and the switching cost to a given candidate MEC host. Then, they resort to a multi-armed bandit approach which estimates the total delay distribution of each MEC host. At each time step, the MEC host with the lowest estimated total cost is chosen. This approach is however susceptible to unnecessary migrations.

Our proposed scheme takes all these delay elements into account and also minimizes the number of migrations. In so doing, it also cuts down the migration costs.

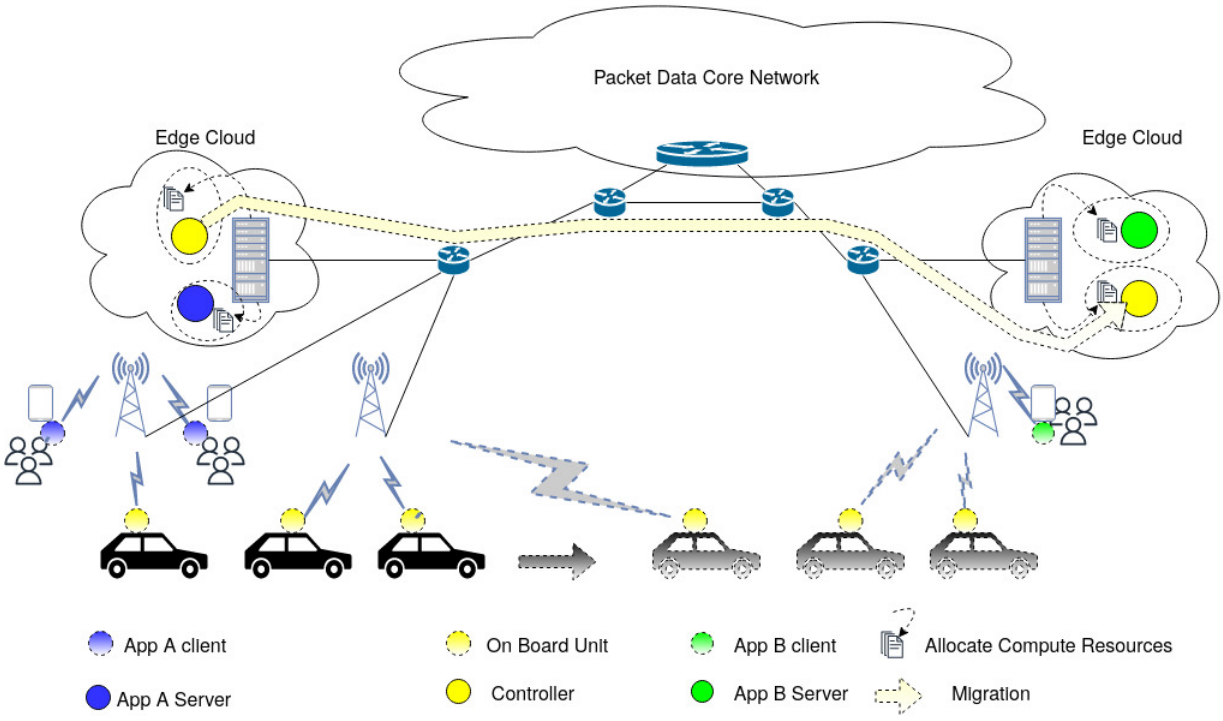


Figure 1: V2I Platooning on the network edge

3. Controller adaptations for V2I platooning

In legacy ad hoc platooning systems, the platoon leader communicates its speed and acceleration to each platoon member [21]. 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, the vehicles only need to communicate their own speed, acceleration and distance from the preceding vehicle to the controller resident in the MEC host, as depicted in Figure 1. From the figure, it is also clear that not all vehicles of the platoon may be served by the same cell tower, and hence packets will experience varying delays. Indeed, network latency and the variation thereof over time strongly affect this process.

As discussed in [22], 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 on-board unit of a 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 as suggested by the performance of automatic control in real world platooning experiments [23].

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, one 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 2, we implement a data collection window[1] to receive the packets from all members of the platoon. The controller then uses these data to compute the acceleration directive for each vehicle.

[1] This window should be long enough to allow all platoon members to send scheduled uplink transmissions, but shorter than the reporting interval to the controller. In this paper we use a collection window of 30 ms.

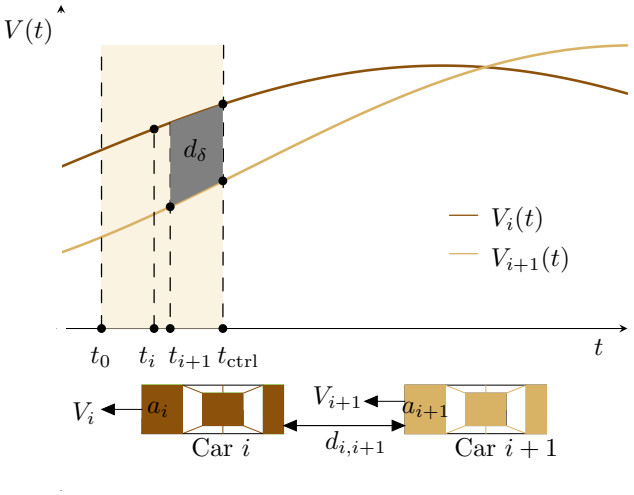


Figure 2: Controller delay compensation. At time t_i, car i generates the report packet with its own speed, acceleration and distance from vehicle $i-1$, and sends it to the MEC controller. Car $i+1$ does likewise at time t_{i+1}. At t_{ctrl}, the controller collates the data. The time from t_0 to t_{ctrl} (highlighted) is the data collection window.

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 such lags are in the order of up to several tens of milliseconds, these updates can be approximated as piecewise linear functions as depicted in Figure 2. 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:

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

3.3. Slow down And spLiT (SALT)

In some situations, the delay experienced by vehicle update messages may be too high to be satisfactorily compensated for as described in Section 3.2. In these cases, the MEC controller becomes an unreliable arbiter of the platoon, and it becomes safer to switch control back to the vehicles. In order to gracefully carry out this transition, the speed and spacing of the platoon need to be progressively adjusted as long as the untenable delay conditions persist. We do this by implementing a Slow And spLiT (SALT) overlay module inside the MEC controller. SALT leverages the total delay experienced as the trigger for its operation which consists in (*i*) adjusting the target distance used in CACC and (*ii*) switching between CACC and ACC as needed.

As network and computing conditions vary, the interval between when a vehicle transmits a packet to the controller to when it receives a packet from the controller also varies. Simply capturing the upstream delay at the controller may not account for the computing and downstream delay. In order to get a complete picture of the delay, we include the creation timestamp of the vehicle report to which the controller is responding as part of the control packet. When the vehicle receives the control directive, it computes the difference between the current time and the creation timestamp, and sends it to the controller as part of the next report.

Given that these delay values are quite noisy, we first pass them through a low-pass filter before using them as an input to our variable spacing function. If m is the number of average delays collected over an observation window[2] and $x[m]$ is the m^{th} sample, at the n^{th} observation window, the filtered delay $y[n]$ is given by

$$y[n] = \left(\frac{1}{2}\right)^{m+1} y[n-1] + \sum_{i=1}^{m} \left(\frac{1}{2}\right)^{i} x[m-i+1]. \tag{4}$$

We use this filtered delay to determine how to adjust the vehicle spacing. We choose $T_{\max}$ as reference upper bound on delay and $0 < \Psi < 1$ as the trigger point for when to adjust the spacing. The modified spacing $d[n]$ is obtained as follows

$$\Delta = \frac{1}{10} \left\lceil 10 \times \tanh\left(\frac{y[n] - T_{\max}}{T_{\max}}\right) \right\rceil, \tag{5}$$

$$d[n] = \begin{cases} d_\lambda, & \text{if } \Delta \leqslant \Psi \\ d_\lambda(1+\Delta), & \text{otherwise,} \end{cases} \tag{6}$$

where d_λ is the platooning target for the inter-vehicular spacing. Should delays exceed $T_{\max}$, the chances of successive packets with stale data arriving at the controller increases considerably. We choose the hyperbolic tangent since it has a smoothing effect on noisy inputs and its output is bounded in the interval [-1.0,1.0]. We use a simple geometric function with a ratio, $r < 1.0$, to adjust the speed at each observation window.

We choose $r \approx 1.0$ to avoid shocks that can cause string instability. The filtered delay is similarly used as the trigger. The speed at the at the n^{th} observation window $V[n]$ is

[2] We choose this to be 200 ms (twice the vehicular reporting interval), averages are collected over short intervals of 30 ms each corresponding to the data collection window. For a relatively large platoon, these values ensure that m is statistically significant.

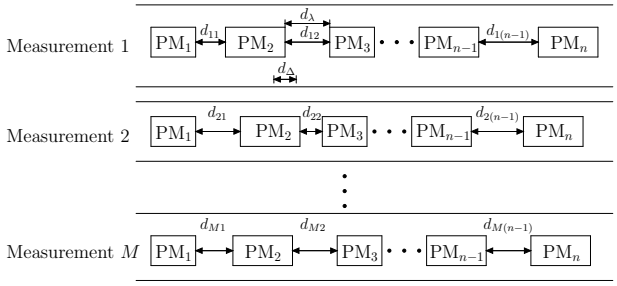


Figure 3: Platoon metrics as reported to the MEC host. "PM" denotes a platoon member; d_λ is the required vehicle spacing; d_Δ is the spacing tolerance.

given by

$$V[n] = \begin{cases} V_{\text{ptn}}, & \text{if } d[n] = d_\lambda \\ \dfrac{V[n-1]}{r}, & \text{if } V[n-1] < V_{\text{ptn}} | \Delta \leqslant \Psi \\ V_{\text{acc}}, & \text{if } d[n] \geqslant 1.2\,\text{s} \times V_{\text{acc}} | \Delta > \Psi \\ V[n-1] \times r, & \text{otherwise.} \end{cases} \tag{7}$$

When $V[n] = V_{\text{acc}}$, vehicles in the platoon switch to a standard Adaptive Cruise Control (ACC) that relies only on on-board sensors such as the radar. The ACC algorithm is provided by the Plexe framework [24], included within the SUMO mobility simulator that we use for our performance evaluation (see Section 5.1 for the details). In Equation (7), 1.2 s is the default time headway provided by Plexe, which is chosen to guarantee string-stability in the presence of actuation lags (set to 0.5 s) [25].

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 host 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 [26]. 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).[3]

4.1. Data for state context definition

A schematic showing the measurements received at the MEC is illustrated in Figure 3. 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 3, with M measurements and n cars, as:

$$\gamma^{(-)} = \frac{1}{M(n-1)} \sum_{i=1}^{M} \sum_{j=1}^{n-1} \left\lfloor \frac{d_\lambda - d_{ij}}{d_\Delta} \right\rfloor \text{ for } d_{ij} \leqslant d_\lambda \tag{8}$$

$$\Gamma^{(-)} = \frac{\gamma^{(-)}}{1 + \gamma^{(-)}} \tag{9}$$

$$\gamma^{(+)} = \frac{1}{M(n-1)} \sum_{i=1}^{M} \sum_{j=1}^{n-1} \left\lfloor \frac{d_{ij} - d_\lambda}{d_\Delta} \right\rfloor \text{ for } d_{ij} \geqslant d_\lambda \tag{10}$$

$$\Gamma^{(+)} = \frac{\gamma^{(+)}}{1 + \gamma^{(+)}}\,. \tag{11}$$

Here, d_{ij} is the inter-vehicular spacing between vehicles i and j, 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:

$$T_{\text{ddt}} = T_{\text{net}} + T_{\text{proc}}\,, \tag{12}$$

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 multiple factors influence T_{net}, including the capacity of the

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

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 $T_{\text{signaling}}$ it takes to signal the platoon members about the new location to communicate with:

$$T_{\text{ovh}} = T_{\text{signaling}} + T_{\text{migration}}. \tag{13}$$

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

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

$$T_{\text{R}} = \frac{T_{\text{ddt}} + T_{\text{ovh}}}{T_{\text{budget}}}. \tag{14}$$

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.

$$T_{\Delta} = 2\frac{T_{\text{proc}}^{(t)} - T_{\text{proc}}^{(t-1)}}{T_{\text{proc}}^{(t)} + T_{\text{proc}}^{(t-1)}}, \tag{15}$$

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 20 s. This design value is a trade-off between timely migration and adequate time to collect decision data.

Each candidate MEC with $T_R < 1$ presents a migration option characterized by a given relative migration delay

$$\theta = \frac{T_{\text{migration}}^{\text{candidate}}}{T_{\text{budget}}}, \tag{16}$$

and relative processing capability

$$\beta = \frac{T_{\text{proc}}^{\text{candidate}}}{T_{\text{proc}}^{\text{current}}}. \tag{17}$$

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.

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. Problem Formulation

We consider that there exist N possible locations on which the controller can be hosted over the epoch given by $(t+1)$. The selection options over epoch $(t+1)$ can be expressed in vector form as $\eta^{(t+1)} = [\eta_1^{(t+1)}, \eta_2^{(t+1)}, ..., \eta_N^{(t+1)}]$.

The compute node, k, chosen to host the controller has unknown background traffic (ρ_k) and its location impacts the network latency (τ_k) that is experienced. Consequently, these two elements affect the spacing errors of the platoon. The placement problem can be thus be formalized as:

$$\min \sum_{i=1}^{M}\sum_{j=1}^{n-1}\left| d_{\lambda} - \sum_{k=1}^{N} d_{ij}(\rho_k, \tau_k)\eta_k^{(t+1)} \right| \tag{18a}$$

$$\text{s.t.:} \sum_{k=1}^{N} \eta_k^{(t+1)} = 1, \tag{18b}$$

$$\eta_k^{(t+1)} \in \{0, 1\}. \tag{18c}$$

Given the stochastic nature of both the background traffic and network latency from each platoon member to each of the N locations over the entire epoch, a reliable model to solve this problem is unfeasible. We therefore use model free reinforcement learning.

4.3. Q-learning

We briefly discuss key aspects of Q-learning which are crucial to understanding the remainder of the text. 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 [12].

When the environment is in state S, the agent takes action a, obtains the reward R and the environment transitions to state S' [26]. Therefore, we have

$$Q(S, A) \leftarrow Q(S, A) + \alpha\left(R + \gamma \max_a Q(S', a) - Q(S, A)\right), \tag{19}$$

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 $\alpha = \frac{1}{k}, k > 0$. 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 later stages to make use of the policies learned. 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 [27], we design a migration agent that exploits context-awareness. A procedural depiction of this process is presented in Algorithm 1. The agent retrieves the platoon spacing data, **line 10**. The agent also obtains the processing statistics of the MEC hosts in order to estimate their capacity with reference to the current host, **line 11**. These are retrieved to derive the context, **line 12**, as specified in Equation (9) and Equation (11). It then retrieves a listing of the available MEC hosts and estimates the migration time to the new host, **line 14**, either based on any previous logs of a similar migration, or based on a default value if no recent migration targeted that 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 4. Comparing these options, according to Equation (19), the agent makes an ϵ-greedy decision as to which MEC host to migrate to, **line 15**, as shown in Figure 5. Once the migration has been carried out, the agent updates the reward, **lines 18-24**. We also note that this migration experience is stored to be shared with other parallel agents, **lines 26-31**. The details of this shared learning will be presented in Section 4.5. We now discuss what comprises the reward.

Algorithm 1: Contextual Q-learning migration

Result: Migrate Controller, update Q-Tables for n^{th} agent

1 $Q^{(n)} \leftarrow$ ReadSharedPolicyFile();
2 $\gamma \leftarrow 0.8$;
3 $i \leftarrow 0$;
4 $RunCounter \leftarrow 0$;
5 $T_{\text{mec}} \leftarrow$ DefaultMEC;
6 $InitCtx \leftarrow$ getInitialContext();
7 **while** *True* **do**
8 $Q(S, A) \leftarrow$ ReadQ($Q^{(n)}$, $InitCtx$);
9 wait_migration_interval;
10 $x_{\text{log}} \leftarrow$ getPlatoonStats();
11 $C_{\text{log}} \leftarrow$ getGetMecOpStats(T_{mec})
12 $Ctx \leftarrow$ getContext($x_{\text{log}}, C_{\text{log}}$);
13 **if** $RunCounter > 0$ **then**
14 $ArrMEC \leftarrow$ getAvailableMEC();
15 $T_{\text{mec}} \leftarrow$ MoveCtrl($Ctx, Q^{(n)}, ArrMEC$);
16 **end**
17 //Update Reward
18 $R \leftarrow$ getImmediateReward(x_{log});
19 $Q(S', a) \leftarrow$ ReadQ($Q^{(n)}$, Ctx);
20 $R' \leftarrow R + \gamma Q(S', a)$;
21 $k \leftarrow$ ReadStateVisits($Q^{(n)}$, $InitCtx$);
22 $k \leftarrow k + 1$;
23 $R_\Sigma \leftarrow \frac{1}{k}(R' - Q(S, A)) + \frac{k-1}{k}(Q(S, A))$;
24 $Q^{(n)} \leftarrow$ UpdateQ($Q^{(n)}$, $InitCtx$, R');
25 //Store visited state for experience sharing
26 $ArrShared[i] = \{R', InitCtx\}$;
27 $i \leftarrow i + 1$;
28 **if** $i == 5$ **then**
29 SharedXP_Thread($ArrShared$);
30 $i \leftarrow 0$;
31 **end**
32 $InitCtx \leftarrow Ctx$;
33 $RunCounter \leftarrow RunCounter + 1$;
34 **end**

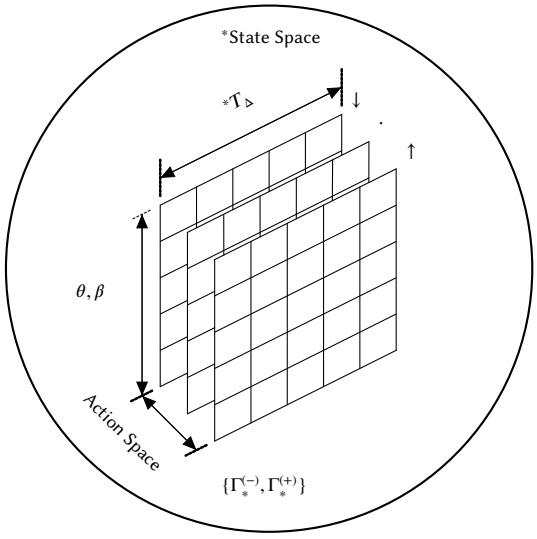


Figure 4: Migrator State and Action Spaces. The relative migration delay and relative candidate MEC power tuple $\{\theta, \beta\}$, the change in processing time between successive epochs, T_Δ and the platoon topology given by the tuple $\{\Gamma^{(-)}, \Gamma^{(+)}\}$ constitute the state space. The actions "Remain in the current MEC host" and "Migrate to any MEC host characterized by $T_R < 1$" form the action space. The table $(\cdot)$ indicates migration options to MEC hosts with the same power as the current host; $(\uparrow)$ to more powerful MEC hosts; and $(\downarrow)$ to less powerful hosts than the current one.

4.4. 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 reward 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

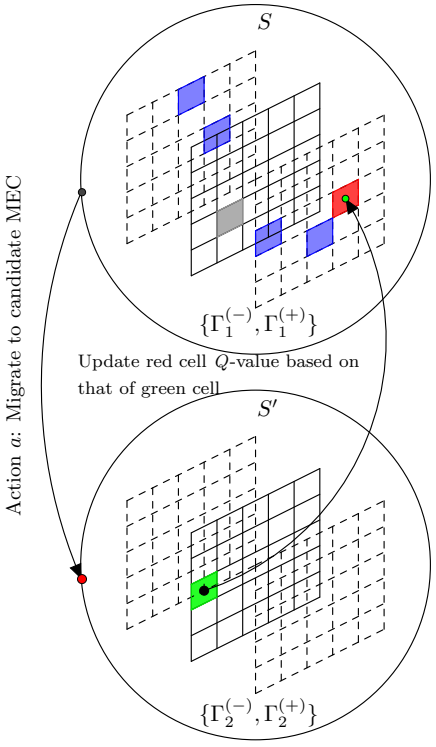


Figure 5: The Q-learning update process of the migration agent. The gray cell represents the Q-value of the MEC currently hosting the controller. The red cell represents $Q(S, A)$, the Q-value of the MEC host chosen to host the controller from the context of the current host and platoon topology specified 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.

a packet urgent and therefore requires 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

$$R_{\text{net}} = \frac{X_{\text{normal}}}{X_{\text{urgent}} + X_{\text{normal}}} - \zeta\,. \tag{20}$$

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

$$\mathcal{D} := \left\{d_{ij}\right\}_{i=1,j=1}^{i=M,j=n-1}$$

$$\mathcal{D}_\epsilon := \{d \in \mathcal{D} \;:\; d < d_\lambda - d_\Delta\}$$
$$\mathcal{D}_\Omega := \{d \in \mathcal{D} \;:\; d > d_\lambda + d_\Delta\}$$
$$\mathcal{D}_0 := \{d \in \mathcal{D} \;:\; |d - d_\lambda| \leqslant d_\Delta\}$$

$$r_\epsilon = \frac{1}{|\mathcal{D}_\epsilon|} \sum_{d_{ij} \in \mathcal{D}_\epsilon} \left| \frac{d_\lambda - d_{ij}}{d_\Delta} \right| \tag{21}$$

$$R_\epsilon = \frac{r_\epsilon}{1 + r_\epsilon} \tag{22}$$

$$r_\Omega = \frac{1}{|\mathcal{D}_\Omega|} \sum_{d_{ij} \in \mathcal{D}_\Omega} \left| \frac{d_\lambda - d_{ij}}{d_\Delta} \right| \tag{23}$$

$$R_\Omega = \frac{r_\Omega}{1 + r_\Omega} \tag{24}$$

$$r_0 = \frac{1}{|\mathcal{D}_0|} \sum_{d_{ij} \in \mathcal{D}_0} \left| \frac{d_\lambda - d_{ij}}{d_\Delta} \right| \tag{25}$$

$$R_0 = 0.5 - \frac{r_0}{1 + r_0} \tag{26}$$

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 $d_{\text{safe}} \ll d_\lambda$:

$$R_{\text{safe}} = \begin{cases} 0 & \text{if } d_{ij} \geqslant d_{\text{safe}}\ \forall d_{ij} \in \mathcal{D} \\ 1 & \text{otherwise} \end{cases} \tag{27}$$

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

$$R_\lambda = |\mathcal{D}_\epsilon| w_\epsilon R_\epsilon + |\mathcal{D}_\Omega| w_\Omega R_\Omega \tag{28}$$
$$+ |\mathcal{D}_0| w_0 R_0 - |\mathcal{D}| R_{\text{safe}}, \tag{29}$$

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

$$R = |\mathcal{D}| R_{\text{net}} + R_\lambda. \tag{30}$$

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

4.5. 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

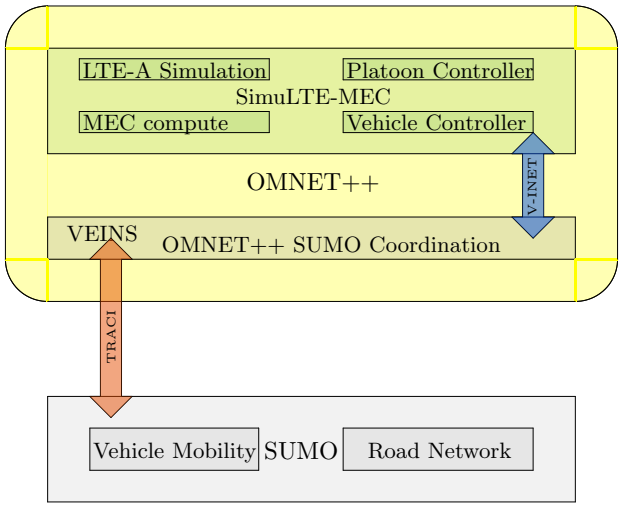


Figure 6: Simulation frameworks

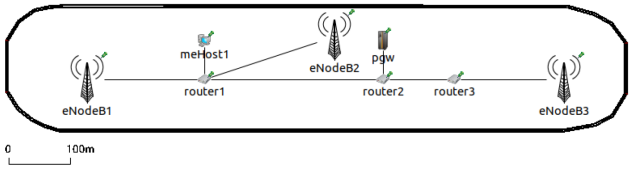


Figure 7: Road and MEC Network for evaluating SALT. The scale corresponds to the road network, eNodeBs and the MEC host are exaggerated to make them discernible.

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 to Q-values of the visited states that will only later be reflected in the shared policy.

The scheme described above can model the behavior 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 6. The first is the robust vehicular simulator Simulation of Urban MObility (SUMO) from the German Aerospace Agency DLR [28]. It is widely used in research given its highly realistic depiction of vehicular mobility. It also includes Plexe modules [24], a framework for the simulation of platooning systems featuring different control laws and actuation models. In the evaluation, we use the ACC algorithm and the first order lag that simulates actuation dynamics. The second component is the OMNeT++ Framework "Vehicles in Network Simulation (VeINS)" [29]. 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 [30], an OMNeT++ framework that realistically simulates the LTE-Advanced network with mobile edge computing extensions.

In order to test the operation of SALT, we simulate the road and MEC network as depicted in Figure 7. In this scenario, only one MEC host is available and migration is not an option. Whereas eNodeB1 and eNodeB2 have a fast connection to the MEC host, eNodeB3 does not have a LAN connection and has to rely on slower routing through the core network when it handles packets between the platoon members and the controller. We vary the average exponential routing delay through the core network between 1 ms and 10 ms. With respect to Equation (5), we choose $T_{\max} = 50$ ms which is half the reporting interval for the vehicles. In addition, with reference to Equation (6) and Equation (7), we set $\Psi = 0.3$, $V_{\text{acc}} = 10\,\text{m/s}$ and $r = 0.9$. The rest of the simulation parameters are as specified in Table 1.

In order to test the performance of the migration scheme, we use the scenario depicted in Figure 8. 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
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)	$X \sim \mathcal{U}(0\text{ ms}, 10\text{ ms})$
Core routing delay (per packet)	$X \sim \text{Exp}(1\text{ ms})$
X2 hop delay (per packet)	50 μs
Migration delay (per vehicle)	$X \sim \mathcal{U}(1\text{ ms}, 3\text{ ms})$
Migration epoch interval	20 s
Vehicular reporting interval	100 ms
Platoon speed V_{ptn} (Average)	25 m/s
Platoon speed (Oscillation)	20% V_{ptn}
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_e	-0.5
w_Ω	-0.1
w_0	0.25

Table 1 also lists the parameters used to to compute the reward function of platooning. We choose the weights such that the negative reward (i.e., the penalty) of being 20% below the target spacing or 50% above has comparable impact to 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 the 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 hosting the controller is not directly connected to the eNodeB serving a vehicle. Figure 8 depicts the road circuit that we use in SUMO to delimit a training episode, along with the communication network in SimuLTE-MEC. 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 31 s, the subsequent intervals after that are 20 s each. These are the epochs at

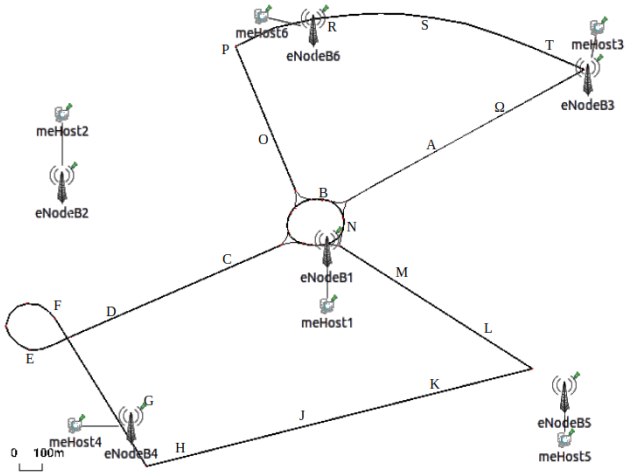


Figure 8: Road and MEC Network. The scale corresponds to the road network. The size of eNodeBs and MEC hosts (prefixed "meHost") are exaggerated to make them discernible.

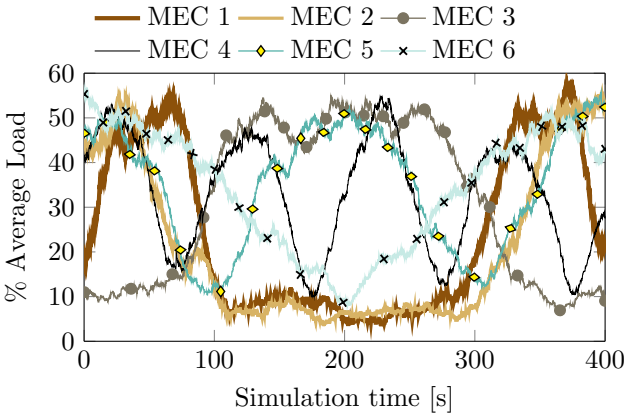


Figure 9: MEC host load due to background traffic.

which the migration agent makes its decisions. In order to make the migration scenario challenging, each eNodeB has a MEC host attached to it such that the controller may be hosted at those locations. The background traffic of the MEC hosts, independent of the platooning load, is depicted in Figure 9. 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 8.

5.2. Asynchronous shared learning extension

To test the potential of the asynchronous shared learning scheme described in Section 4.5, we use parallel simulations of platoons of cars lapping through the same circuit. We test the centralized version of our scheme, with each platoon accessing the shared migration policy every 5 epochs.

Algorithm 2: Asynchronous Shared Learning

Result: Update Shared Policy File

1 **Function** *SharedXP_Thread(xArr)*
2 //Ensure this thread has sole access to shared file LockSharedPolicyFile();
3 //Capture experiences of other parallel agents $Q^{(n)} \leftarrow$ ReadSharedPolicyFile();
4 //Update the working policy with own experiences
5 **foreach** $XP \in xArr$ **do**
6 $R_n \leftarrow XP[0]$;
7 $TmpCtx \leftarrow XP[1]$;
8 $k_n \leftarrow$ ReadStateVisits($Q^{(n)}$, $TmpCtx$);
9 $k_n \leftarrow k_x + 1$;
10 $Q(S,A) \leftarrow$ ReadQ($Q^{(n)}$, $TmpCtx$);
11 $R_x \leftarrow \frac{1}{k_x}\big(R_n - Q(S,A)\big) + \frac{k_x - 1}{k_x}\big(Q(S,A)\big)$;
12 $Q^{(n)} \leftarrow$ UpdateQ$\big(Q^{(n)}, TmpCtx, R_x\big)$;
13 **end**
14 //Publish updated policy for all agents WriteSharedPolicyFile($Q^{(n)}$);
15 //Release file for use by other agents UnLockSharedPolicyFile();
16 *return*;
17 **end**

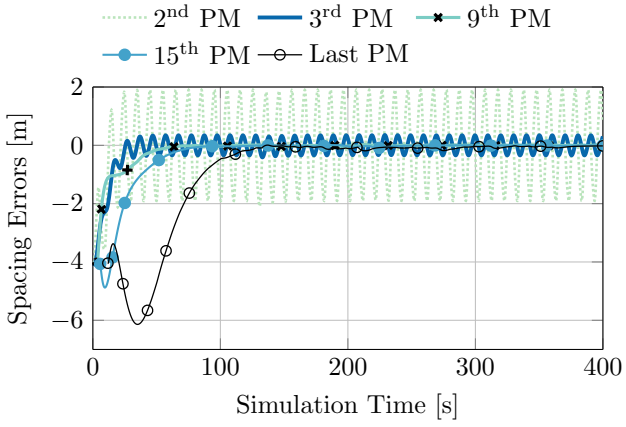
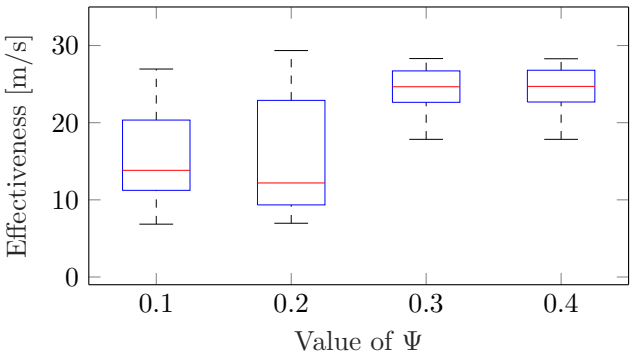


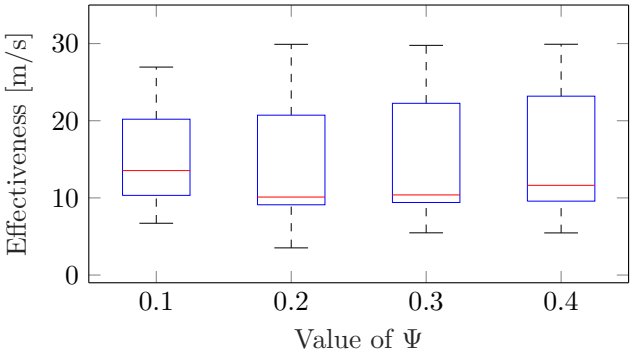
Figure 10: Spacing errors of platoon members (negative means farther).

At initialization time, the parent platooning simulation directory is cloned into a number of directories commensurate to the selected number of parallel agents. However, the overall policy file is placed in a directory accessible to all participating agents. After cloning, a shell script triggers the start of the SUMO/OMNeT++ simulation as well as that of the migration script in each directory.

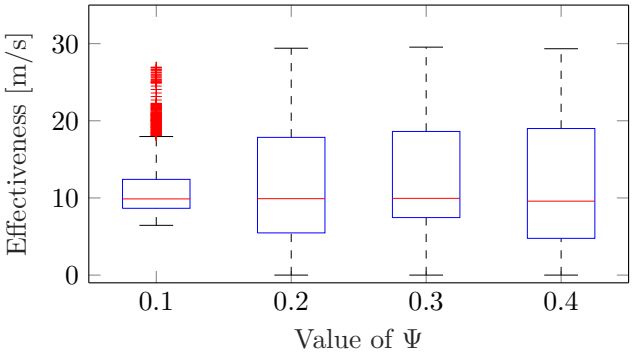
Our shared learning procedure is as shown in Algorithm 2. When the migration agent starts, it reads the overall policy file into a data structure and proceeds to update it as it makes its decisions. The start time of the simulation is different in each directory, in order to mirror the separation of 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, **lines 2-3**, 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 the states it visited, **lines 4-12**, as prescribed in Equation (19). It then overwrites the policy file with the updated data structure, **line 13**. Finally, it unlocks the policy file, **line 14**. Should an agent find the file locked by another, it waits for a random time and then retries until the file is unlocked.



(a) Core routing delay $X \sim$ Exp(1 ms)



(b) Core routing delay $X \sim$ Exp(5 ms)



(c) Core routing delay $X \sim$ Exp(10 ms)

Figure 11: Platoon effectiveness (spacing fairness × average speed) for different SALT triggers.

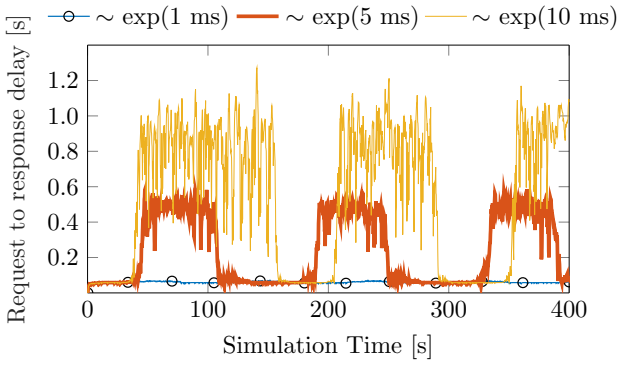


Figure 12: Filtered overall delay (interval between a vehicle sending packet to receiving a control directive from the controller) for the SALT scenario.

5.3. Evaluation results

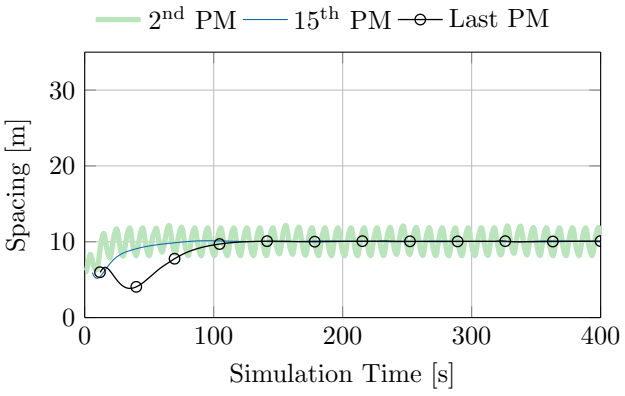
In this section we examine the results from the modifications on the controller. We then consider platoon performance resulting from the use of our migration scheme, compared to that of the state of the art scheme proposed in [6].

5.3.1. Controller modifications

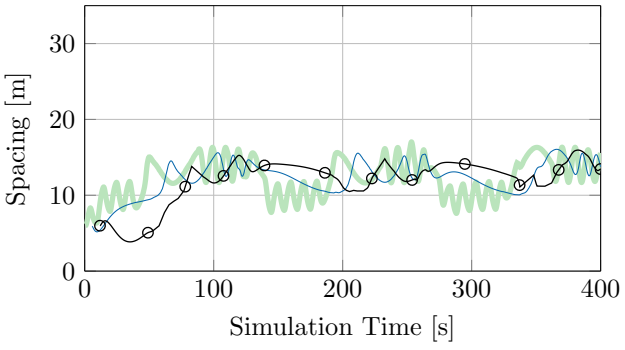
We first examine the platoon formation when the controller is hosted on the MEC. In order to set the most challenging condition for the controller [16], we add a sinusoidal perturbation to the movement speed of the platoon leader. This sinusoidal driving pattern also accounts for speed variations as 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 10. The large spacing errors at the beginning of the simulation are simply due to the transient phase and the initial positions of the vehicles, which are not injected at their steady-state spacing. We remark that platoon safety is still preserved as vehicles do not collide, and no dangerously short spacing occurs, even during the initial transient phase. The spacing between the 2nd PM (i.e., 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 [25] and thereby ensure platoon safety.

5.3.2. SALT

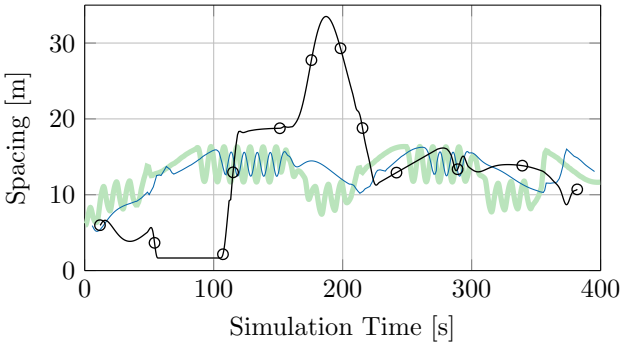
In order to compare platoon performance with regard to the safety trigger value Ψ, we derive an Effectiveness metric by multiplying Jain's fairness index [31] in spacing, x, and the average speed, $\dot{x}$ for all n platoon members over each observation window.



(a) Core routing delay $X \sim \text{Exp}(1)$ ms



(b) Core routing delay $X \sim \text{Exp}(5)$ ms

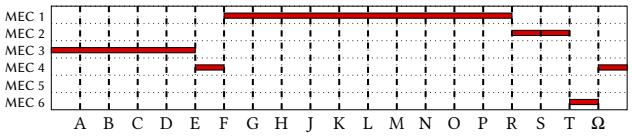


(c) Core routing delay $X \sim \text{Exp}(10)$ ms

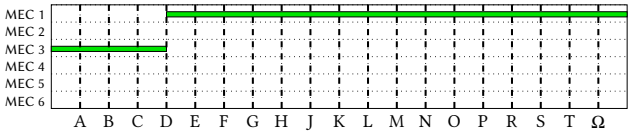
Figure 13: SALT performance for varying degrees of delay with $\Psi = 0.3$

$$\text{Effectiveness} = \frac{\left(\sum_{j=1}^{n-1} x_j\right)^2}{(n-1)\sum_{j=1}^{n-1} x_j^2} \times \frac{1}{n}\sum_{j=1}^{n} \dot{x}_j \qquad (31)$$

The distribution of this metric is depicted in Figure 11. A low value, $\Psi = 0.1$ or $\Psi = 0.2$, makes the controller too sensitive such that the platoon devolves into Automated Cruise Control even when the core routing delay is small as in Figure 11a. On the other hand a higher value, $\Psi = 0.4$, does not trigger the safety measure quickly enough when the delay values are substantially higher possibly resulting in low effectiveness overall as shown in Figure 11c. A setting of $\Psi = 0.3$, however, provides a balance between these



(a) Q-migration Policy A



(b) Q-migration Policy B

Figure 14: Sample Q-migration policies

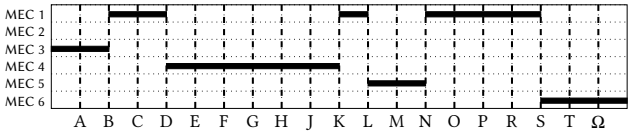


Figure 15: Follow ME migration policy [6]

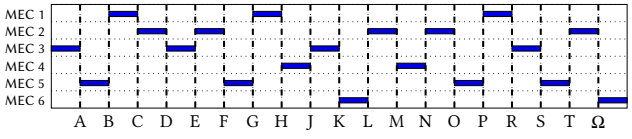


Figure 16: AUSP migration policy [20]

extremes. We remark that the setting of Ψ can be different depending on expected channel conditions. In situations where a large delay may be indicative of further channel deterioration, such as shadowing in urban environments, a lower value of Ψ is typically the best choice.

The total delay, filtered using Equation (4), between when a vehicle sends a packet to the controller to when it receives a packet from the controller is shown in Figure 12. When the core routing delay is sufficiently low, with an average exponential distribution of 1 ms, all responses from the controller arrive within the 100 ms interval between consecutive vehicular reports. As the core routing delay increases, the periods in which platooning can be safely carried out reduces.

We also evaluate the performance of SALT with increasing delay as shown in Figure 13. When the delay is low, cf. Figure 13a, ACC is not triggered and CACC platooning is maintained throughout. With a moderate increase in delay, as shown in Figure 13b, there is a smooth transition between ACC and CACC. When the delay is high (e.g., in the experiment time interval between 50 s and 150 s, see Figure 12), the vehicles are controlled by ACC with larger spacing between them. When the delay becomes tolerable (e.g., from 150 s to 190 s of simulated time, cf. Figure 12), the platoon is reformed and the vehicle spacing reverts to the required value. These transitions in spacing are more gradual. However, when the routing delay is substantially higher, Figure 13c, the switch between ACC and CACC is more drastic especially towards the tail of the platoon. There are also brief periods in which the vehicles at the tail end of the platoon almost come to a stop as ACC control re-establishes safe spacing. Overall, increasing delays lead to increasing ACC dominance over CACC and vice-versa.

In the following sections, we update the results of our previous experiments in [13] incorporating SALT into the workings of the controller.

5.3.3. *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 14. These sequences were the most recurrent in which the migration agent attained full convergence for every visited state (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 [6], forces numerous migrations as shown in Figure 15. Moreover, the other state-of-the-art scheme we consider in this article (AUSP [20]), which also takes into account the processing delay, migration cost and communication delay), results in even more migrations as it tries to minimize the total delay as depicted in Figure 16. Yet, these migrations may prove inconvenient, as we discuss next.

5.3.4. *Platoon stability*

Considering the same sinusoidal driving pattern employed so far, Figure 17 shows that using our Q-migration scheme, 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. Policy A which involves more migrations

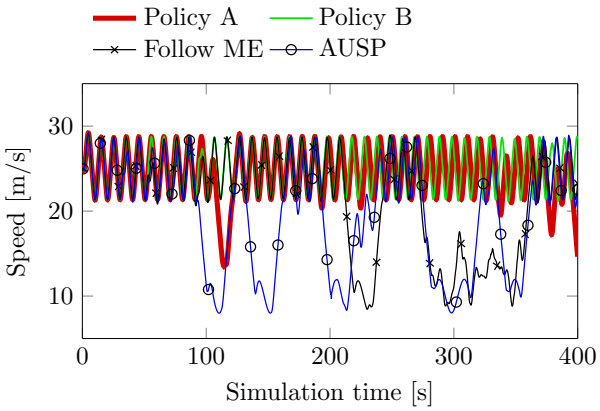


Figure 17: Speed profiles of the first follower.

compared to Policy B exhibits instances in which SALT is triggered but these are short-lived. In comparison, Follow ME [6] and AUSP [20] exhibit much greater variability given that SALT is triggered over longer intervals. AUSP which carries out a considerable number of migrations exhibits a higher number of speed deviations as a consequence of the extra delays incurred. This points to the efficacy of the minimal migrations that our algorithm decides to perform.

When compared to Follow ME and AUSP, our scheme exhibits better platoon spacing discipline. This is shown in Figure 18, 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 (primarily due to the spacing between the first follower and the platoon leader).

From Figure 18, we observe that our Q-migration algorithm achieves smaller inter-quartile ranges (Figure 18a and Figure 18b) compared to those of Follow ME (Figure 18c) and AUSP (Figure 18d). Follow ME exhibits small dispersion when it maintains the controller on MEC 4 between simulation time 100 s to 200 s (cf. 15) since it does not incur migration costs. As it resumes more migrations after 200 s, dispersion becomes significant. AUSP attempts to even out the spacing resulting in fewer outlying values from the first follower. It however also results in noticeable dispersion throughout. The gains that AUSP achieves in migrating to the lowest utilized, closest MEC host are curtailed by the large migration costs incurred in the process.

The narrow inter-quartile range achieved by our scheme 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.

Table 2
Parallel episodes until convergence

# Agents	1st Conv.	2nd Conv.	3rd Conv.	4th Conv.	5th Conv.
10	73	75	76	77	78
5	304	311	315	321	348
1	867	1293	1306	1322	1340

5.3.5. *Asynchronous shared learning*

The acceleration of convergence through the use of asynchronous shared learning is apparent from Table 2. In this table, we consider each set of parallel episodes, and examine the log of the states that each parallel agent visits. If $\epsilon = \epsilon_{\min}$ at every epoch for any one or more of the parallel agents, we consider that set of episodes as exhibiting convergence given that the policy may be accessed by all agents. The n^{th} time that this is observed, for a set of parallel episodes, 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

In this article, we have presented a context-aware Q-learning-based migration algorithm that learns the appropriate strategies to migrate a platoon controller from one MEC host to another on the network edge. Our approach achieves better adherence to platoon speed and spacing presets, compared to state-of-the-art migration schemes. We have also presented our asynchronous shared learning algorithm that leverages the presence of multiple platoons in order to learn migration policies faster.

Furthermore, we customized the CACC algorithm to enable its operation from the network edge. In addition, we have presented an overlay safety module that enables the graceful switching of control from the MEC controller (CACC) to the vehicle (ACC) in case of sustained high delays.

As part of future work we intend to explore the migration of hierarchical controllers that manage sub platoons, with fewer vehicles, that are more robust to delays. The sub platoon controllers in this case may be migrated to different hosts more often than the over-arching platoon controller, with positive implications on overall communication delays.

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

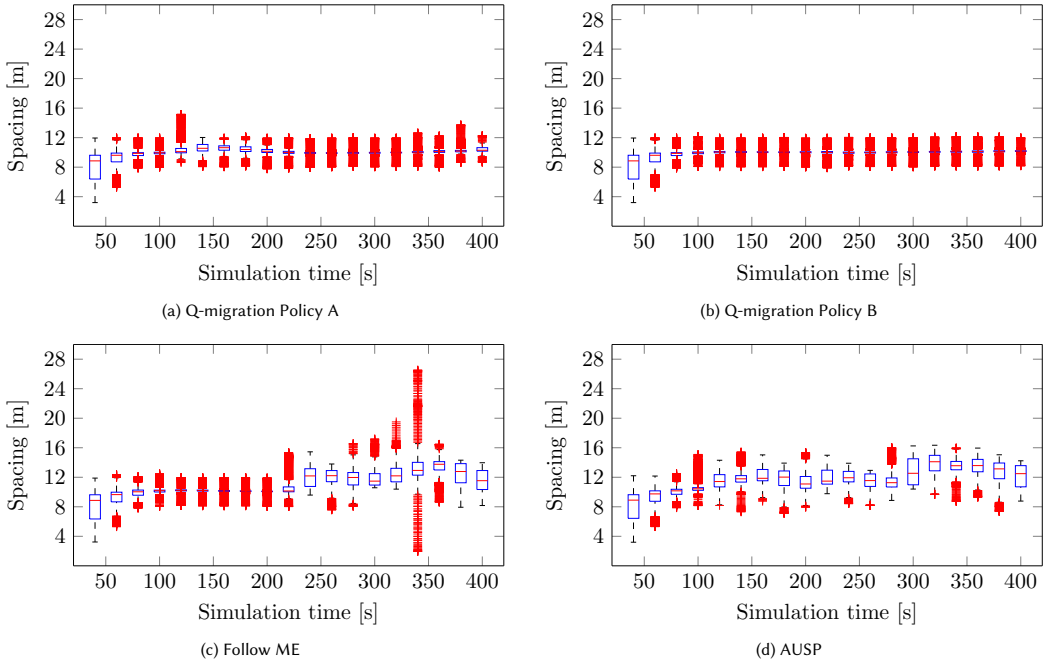


Figure 18: Distribution of spacing between platoon members. Each box plot represents data taken over 20 s windows. The last box plot represents data in the last 10 s.

(MIUR) under the initiative "Departments of Excellence" (Law 232/2016).
Michele Segata was partially supported by the Free University of Bozen–Bolzano under the SECEDA project, RTD Call 2021.

References

[1] M. Boban, A. Kousaridas, K. Manolakis, J. Eichinger, W. Xu, 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. doi:10.1109/MVT.2017.2777259.

[2] S. Barmpounakis, G. Tsiatsios, M. Papadakis, E. Mitsianis, N. Koursioumpas, N. Alonistioti, Collision avoidance in 5G using MEC and NFV: The vulnerable road user safety use case, Computer Networks 172 (2020) 107150. doi:https://doi.org/10.1016/j.comnet.2020.107150.

[3] F. A. Teixeira, V. F. e Silva, J. L. Leoni, D. F. Macedo, J. M. Nogueira, Vehicular networks using the IEEE 802.11p standard: An experimental analysis, Vehicular Commun. 1 (2) (2014) 91–96. doi:https://doi.org/10.1016/j.vehcom.2014.04.001.

[4] S. Öncü, J. Ploeg, N. van de Wouw, H. Nijmeijer, Cooperative adaptive cruise control: Network-aware analysis of string stability, IEEE Trans. Intell. Transp. Syst. 15 (4) (2014) 1527–1537. doi:10.1109/TITS.2014.2302816.

[5] F. Dressler, F. Klingler, M. Segata, R. Lo Cigno, Cooperative driving and the tactile Internet, Proc. IEEE 107 (2) (2019) 436–446. doi:10.1109/JPROC.2018.2863026.

[6] T. Ouyang, Z. Zhou, X. Chen, Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing, IEEE J. Sel. Areas Commun. 36 (10) (2018) 2333–2345. doi:10.1109/JSAC.2018.2869954.

[7] A. Virdis, G. Nardini, G. Stea, A framework for MEC-enabled platooning, in: Proc. IEEE WCNCW, 2019, pp. 1–6. doi:10.1109/WCNCW.2019.8902910.

[8] C. Quadri, V. Mancuso, M. Ajmone Marsan, G. P. Rossi, Platooning on the edge, in: Proc. ACM MSWiM, 2020, pp. 1–10. doi:10.1145/3416010.3423220.
URL https://doi.org/10.1145/3416010.3423220

[9] K. Serizawa, M. Mikami, K. Moto, H. Yoshino, Field trial activities on 5g nr v2v direct communication towards application to truck platooning, in: 2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall), 2019, pp. 1–5. doi:10.1109/VTCFall.2019.8891260.

[10] L. Lv, Y. Shi, W. Shen, Mobility-as-a-service research trends of 5g-based vehicle platooning (2021).

[11] C. Quadri, V. Mancuso, V. Cislaghi, M. A. Marsan, G. P. Rossi, From plato to platoons, in: 2021 19th Mediterranean Communication and Computer Networking Conference (MedComNet), 2021, pp. 1–8. doi:10.1109/MedComNet52149.2021.9501242.

[12] C. J. C. H. Watkins, Learning from delayed rewards, Ph.D. thesis, King's College, Cambridge, UK (May 1989).

[13] C. Ayimba, M. Segata, P. Casari, V. Mancuso, Closer than close: Mec-assisted platooning with intelligent controller migration, in: Proceedings of the 24th International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, 2021, pp. 23–32.

[14] R. Rajamani, Han-Shue Tan, Boon Kait Law, 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. doi:10.1109/87.852914.

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

[16] S. Santini, A. Salvi, A. S. Valente, A. Pescapé, M. Segata, R. Lo Cigno, A consensus-based approach for platooning with intervehicular communications and its validation in realistic scenarios, IEEE Trans. Veh. Technol. 66 (3) (2017) 1985–1999. doi:10.1109/TVT.2016.2585018.

[17] F. A. Salaht, F. Desprez, A. Lebre, An overview of service placement problem in fog and edge computing, ACM Comput. Surv. 53 (3) (Jun. 2020). doi:10.1145/3391196.
URL https://doi.org/10.1145/3391196

[18] K. Velasquez, D. Abreu, M. Curado, E. Monteiro, Service placement for latency reduction in the internet of things, Proc. IEEE 72 (2) (2017) 105–115. doi:10.1007/s12243-016-0524-9.

[19] M. Chen, W. Li, G. Fortino, Y. Hao, L. Hu, I. Humar, A dynamic service migration mechanism in edge cognitive computing, ACM Trans. Internet Technol. 19 (2) (Apr. 2019). doi:10.1145/3239565. URL https://doi.org/10.1145/3239565

[20] O. Tao, X. Chen, Z. Zhou, L. Li, X. Tan, Adaptive user-managed service placement for mobile edge computing via contextual multi-armed bandit learning, IEEE Transactions on Mobile Computing (2021) 1–1doi:10.1109/TMC.2021.3106746.

[21] M. Segata, B. Bloessl, S. Joerer, C. Sommer, M. Gerla, R. Lo Cigno, F. Dressler, Toward communication strategies for platooning: Simulative and experimental evaluation, IEEE Trans. Veh. Technol. 64 (12) (2015) 5411–5423. doi:10.1109/TVT.2015.2489459.

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

[23] H.-S. Tan, R. Rajamani, W.-B. Zhang, Demonstration of an automated highway platoon system, in: Proceedings of the 1998 American control conference. ACC (IEEE Cat. No. 98CH36207), Vol. 3, IEEE, 1998, pp. 1823–1827.

[24] M. Segata, R. Lo Cigno, T. Hardes, J. Heinovski, M. Schettler, B. Bloessl, C. Sommer, F. Dressler, Multi-Technology Cooperative Driving: An Analysis Based on PLEXE, IEEE Transactions on Mobile Computing (TMC) (2022). doi:10.1109/TMC.2022.3154643.

[25] R. Rajamani, Vehicle Dynamics and Control, 2nd Edition, Springer, 2012.

[26] R. S. Sutton, A. G. Barto, Introduction to Reinforcement Learning, 2nd Edition, MIT Press, Cambridge, MA, USA, 2020. URL http://incompleteideas.net/book/RLbook2020.pdf

[27] C. Ayimba, P. Casari, V. Mancuso, SQLR: Short-term memory Q-learning for elastic provisioning, IEEE Trans. Netw. Service Manag. 18 (2) (2021) 1850–1869. doi:10.1109/TNSM.2021.3075619.

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

[29] C. Sommer, R. German, F. Dressler, Bidirectionally coupled network and road traffic simulation for improved IVC analysis, IEEE Trans. Mobile Comput. 10 (1) (2011) 3–15. doi:10.1109/TMC.2010.133.

[30] G. Nardini, A. Virdis, G. Stea, A. Buono, SimuLTE-MEC: extending SimuLTE for multi-access edge computing, in: Proc. OMNeT++ Community summit, 2018, pp. 1–8.

[31] R. K. Jain, D.-M. W. Chiu, W. R. Hawe, et al., A quantitative measure of fairness and discrimination, Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA (1984).