Dynamic Controller Assignment in Software Defined Internet of Vehicles through Multi-Agent Deep Reinforcement Learning

Tingting Yuan, Wilson da Rocha Neto, Christian Esteve Rothenberg, Katia Obraczka, Chadi Barakat, Thierry Turletti

Dynamic Controller Assignment in Software Defined Internet of Vehicles through Multi-Agent Deep Reinforcement Learning

Tingting Yuan*, Wilson da Rocha Neto†, Christian Esteve Rothenberg†, Katia Obraczka‡, Chadi Barakat*, Thierry Turletti*
*Inria, Université Côte d'Azur, France
†University of Campinas, Brazil
‡UC Santa Cruz, CA, USA

***Abstract*—In this paper, we introduce a novel dynamic controller assignment algorithm targeting connected vehicle services and applications, also known as Internet of Vehicles (IoV). The proposed approach considers a hierarchically distributed control plane, decoupled from the data plane, and uses vehicle location and control traffic load to perform controller assignment dynamically. We model the dynamic controller assignment problem as a multi-agent Markov game and solve it with cooperative multi-agent deep reinforcement learning. Simulation results using real-world vehicle mobility traces show that the proposed approach outperforms existing ones by reducing control delay as well as packet loss.**

***Index Terms*—Internet of Vehicles (IoV), Software Defined Networking (SDN), multi-agent deep reinforcement learning, controller assignment.**

I. Introduction

Connected, autonomous, or semi-autonomous vehicles, also known as Internet of Vehicles (IoV) has become an important component of Smart Cities [1]. IoV applications are manyfold and their quality-of-service requirements are quite diverse, ranging from bandwidth-hungry infotainment to latency stringent driving messages [2], e.g., 200 ms for cooperative traffic applications, 100 ms for active road safety applications, and less than 10 ms for automation and augmented reality applications [3], [4]. Leveraging the Software Defined Networking (SDN) paradigm [5], Software-Defined IoV (SD-IoV) [6] has been proposed as a way to improve IoV efficiency and simplify their management through increased network programmability achieved via the decoupling of the control and data planes.

However, employing SDN's logically centralized control plane paradigm for IoV [7] will likely increase control plane operation latency as experienced by control messages exchanged between centralized controllers, typically located in the network provider's cloud infrastructure and forwarding devices in the vehicles. Recall that control message latency is a function of both propagation delay which depends on where the controller is placed and controller response delay at the controller which depends on the controller's current load. As such, IoV applications call for a logically distributed control plane, where controllers are placed at the network's edge in close proximity to vehicles. Distributing the network control plane and placing controllers at the edge enable meeting IoV applications' latency requirements by reducing both control message propagation and controller response delays. This can be achieved by adequate (*1*) controller placement and (*2*) controller assignment. Here we define controller placement as the selection of devices that can act as controllers whereas controller assignment refers to choosing which controller will control which forwarding device.

Prior work on controller placement has advocated deploying multiple controllers at different locations to reduce control delay [8]. In particular, hierarchically distributed control architectures [9]–[12] have been proposed as a way to deploy edge controllers near vehicles or users. Most previous studies assume that controllers are placed on dedicated devices in the infrastructure (e.g., data centers, Roadside Units (RSUs), and Base Stations (BSs), etc) and are therefore stationary [13]. To further reduce control delay, existing approaches have explored controller assignment optimization [14]–[16]. Some studies propose to assign controllers statically [10]. IoVs call for dynamic controller assignment approaches that account for variations in load due to IoV's highly dynamic topology and data traffic patterns [16].

However, the dynamic network controller assignment problem is NP-hard [15]. Most existing solutions tackle the problem by employing heuristics to reduce computation cost, which may compromise performance. Deep Reinforcement Learning (DRL) [17], [18] is a machine learning technique that can efficiently tackle complex problems [19] such as dynamic resource allocation [20] by building a higher-level understanding of the target system using deep neural networks. Besides, it does not require a-priori knowledge of the target system as DRL models train online as the system operates. As such, DRL is as a promising solution for dynamic network control assignment. Furthermore, optimized controller assignment using centralized algorithms that require global knowledge [10], [16], [21] is effective but time-consuming and sometimes not viable as they need to gather global information and send back assignment rules to controllers. Motivated by this, we propose a distributed cooperative DRL approach based on a hierarchically distributed network control architecture, that, given the location of controllers and vehicles, aims at reducing control delay through dynamic controller assignment.

The proposed DRL approach combines decentralized decision making by controllers located at the edge of the network and thus closer to vehicles and users, with centralized training using global information in order to achieve adequate tradeoff between control latency and global convergence. Our main contributions can be summarized as follows:

- We formulate the dynamic controller assignment problem in SD-IoVs with the goal of minimizing control delay given vehicle location and control traffic load.
- We propose a real-time distributed cooperative assignment approach, in which controllers make local decisions and coordinate with neighboring controllers. Compared to centralized algorithms, the proposed controller assignment approach improves control latency by not relying on a remote, centralized controller.
- We propose a centralized training approach using global information to attain optimal local assignment yet ensuring global convergence. It uses off-policy with experience replay which allows learning without having to necessarily obtain real time feedback. In other words, the time cost in obtaining and storing global data (usually in several seconds) has negligible impact on training.
- Finally, we evaluate the performance of the proposed dynamic controller assignment approach through simulations driven by real-world vehicle mobility traces. When compared to distance-based assignment, non-cooperative deep reinforcement learning-based assignment, and centralized control, our results indicate that our approach yields reduced control delay and lower data loss.

The remainder of the paper is organized as follows. Section II provides an overview of related work. Section III describes the system model we use in our design, which includes the SD-IoV hierarchically distributed control plane architecture. In Section IV, we formulate the dynamic controller assignment problem in SD-IoV as a multi-agent Markov Decision Process (MDP), and Section V introduces the proposed multi-agent DRL algorithm. Section VI describes our experimental methodology and presents results from the comparative performance evaluation study we conducted. Finally, Section VII concludes the paper with some directions for future work.

II. Related Work

Most related work on controller assignment focus on wired networks, including data centers [14], [15] and wide-area networks [22]–[24]. They typically consider different metrics and goals, e.g. distance between forwarding devices and controllers [24], balancing the load across controllers [23], and response delay [14], [15], [22]. More recently, some proposals considered controller assignment in wireless mobile networks, where node mobility and frequent topology changes impose unique challenges to the controller assignment problem. For example, some approaches try to assign controllers to stationary devices in the communication infrastructure, such as RSUs [10], and LTE BSs (i.e., eNBs) [16]. The work in [10] statically assigns local controllers to RSUs in software defined vehicular networks to minimize the distance as well as to balance load across the set of controlled RSUs. However, this assignment strategy does not adapt to load variation on the RSUs due to vehicle mobility and control traffic load. The work in [16] assigns remote controllers to eNBs accounting for user request rates at the cost of increased propagation time to remote controllers. Additionally, dynamic controller assignment as well as vehicle mobility may lead to control delegation between edge controllers, which has been explored in our previous work [11].

To solve the dynamic controller assignment problem, different classes of algorithms can be considered, including: (1) Optimization algorithms, such as linear programming used in [16], [25], and integer quadratic programming [10]; (2) Heuristic algorithms, such as matching-based [14], [26]; (3) Greedy algorithms, such as [27]; (4) Goal-oriented learning algorithms, such as DRL [23], Although DRL has been used in SDNs for routing and resource allocation [28], it has not yet been explored to perform dynamic controller assignment in decentralized control plane architectures, especially in the context of IoVs.

Motivated by the highly dynamic nature of IoVs coupled with their stringent message delivery latency requirements and leveraging a hierarchically distributed control plane, we propose a distributed cooperative controller assignment algorithm based on Multi-Agent Deep Reinforcement Learning (MADRL). [29], [30].

Using a distributed multi-agent approach such as MADRL to solve this problem avoids centralized single-agent decision making which can be prohibitively expensive in terms of the time to gather and assimilate global data, as well to deploy assignment policies from the central agent to the edge. Besides, a hierarchically distributed SD-IoV architecture [9] allows pushing intelligence to the edge of the network and can inherently accommodate a multi-agent system. As such, our MADRL-based approach for dynamic controller assignment in SD-IoV is based on deploying agents at edge controllers near vehicles and users. As previously noted, we follow a centralized training approach with decentralized execution to attain optimal local assignment yet ensuring global convergence. In other words, in our approach, each agent can make local decisions with local information and cooperate with each other to improve the overall goal of optimizing control latency. Although local agents make decisions without knowing others' local information, there is a tacit understanding of long-term cooperation achieved via centralized training using global datasets.

III. System Model

The main parameters and variables used in our system model are summarized in Table I.

A. Hierarchically Distributed SD-IoV

Fig. 1 illustrates the hierarchically distributed SD-IoV network that we use as a basis for the proposed dynamic controller assignment approach.

TABLE I
SYSTEM MODEL PARAMETERS AND THEIR DESCRIPTIONS

Notation	Description
I	Set of forwarding devices, $I = \{V, R\}$.
C, N	Set and the number of edge controllers.
V, R, B	Set of vehicles, RSUs and BSs.
$\eta_{f,c}$	Boolean parameter that denotes whether a cell f is in the coverage of edge controller c or not.
$\kappa_{f,c}$	Maximum distance between cell f and controller c.
$\kappa_{i,c}$	Round-trip distance between the forwarding device i and the controller c.
$F, \|F\|$	Set and the number of all cells in the scheduled region.
$F_M, \|F_M\|$	Set and the number of all cross-coverage cells in the scheduled region.
$F_c, \widetilde{F}_c$	Set of MCA cells and NCA cells of controller c.
$\tilde{A}_{c_i,c_j}$	Set of cross-coverage cells between two edge controllers.
C_c^*	Neighbors of the edge controller c.
D_f	Mean control delay of a cell f.
D_i	Delay between one forwarding device i and its controller.
$\zeta_{i,f}$	Boolean parameter, denotes whether forwarding device i is in cell f or not.
$d_{tr}(i)$	Transmission delay of forwarding device i.
$d_p(i,c)$	Propagation delay between forwarding device i and controller c.
$d_c, d_q(c)$	Response delay and queuing delay of controller c.
$\pi_{f,c}$	Boolean variable that denotes whether cell f is under the control of the controller c or not.
l_i, l_f	Control load of device i and small area $f \in F$. .
L_c	Load of controller c.
$L_{F_c}, L_{\widetilde{F}_c}$	The load of of controller c in MCA F_c and in NCA $\widetilde{F}_c$.
ϱ_c	Packet loss rate of controller c.
D_t^{max}	Maximum delay of cells at time t.

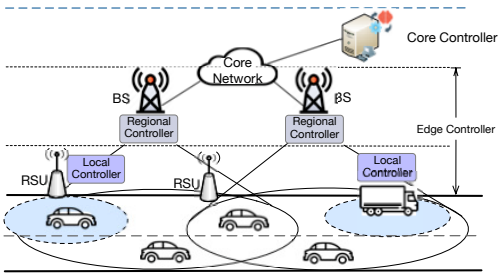


Fig. 1. Hierarchical distributed SD-IoV architecture.

1) Data Plane: The data plane is built around two layers: a static layer and a mobile layer. The static layer consists of a series of stationary communication infrastructure nodes, e.g., RSUs or BSs. The mobile layer contains moving nodes, in particular vehicles that feature on-board units to communicate with RSUs, BSs, and other vehicles. The set of forwarding devices can be defined as $I = \{V, R\}$.

2) Control Plane: The control plane is designed to be a hierarchical distributed architecture. The lower tier consists of multiple edge controllers defined as C, which are deployed near to vehicles. Instead of relaying application requests to remotely deployed controllers, edge controllers can process them locally, enabling therefore quick response time while reducing the computational overhead. According to coverage, the edge controllers are divided into two categories: local controllers in some nearby devices (e.g., RSUs, vehicles), and regional controllers in some relatively remote devices, e.g.,

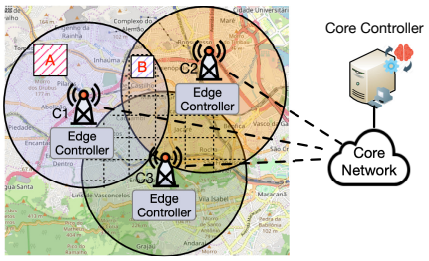


Fig. 2. Use case for edge controller coverage.

BSs, with a larger coverage, thus ensuring the continuity of the control between the areas covered by the local controllers. The top tier contains core controllers located remotely (e.g., in the cloud). They are responsible for supervising the edge controllers and offering a higher view of information.

The rationale behind designing the control plane based on a hierarchically distributed architecture is as follows. In highly dynamic environments, obtaining current and real-time state of the global network is not only a cost-intensive and time-consuming task, but is also likely to compromise data freshness due to long-distance communication and, sometimes, due to lack of connectivity. To reduce control latency between vehicles and their corresponding controller, controllers should be placed closer to the data plane elements. For this purpose, the proposed architecture extends the control plane down to the RSU and BS levels. We can also envision cases where vehicles themselves can also be used as local controllers. However, such fully distributed control plane introduces challenges, such as ensuring controllers are synchronized amongst themselves. Furthermore, some applications may need global information, which a local or regional controller cannot provide. This is why core controllers with a global view are still needed.

B. Edge Controller Coverage

We assume that edge controllers have a certain coverage in terms of geographical radius identical to the coverage of the device it is connected to. Some coverage overlap between edge controllers may exist. These areas are called multi-choice areas, or MCAs for short, in which forwarding devices (e.g., vehicles) can be assigned to one of the multiple controllers covering the area. At the same time, there exists some non-multi-choice areas (called NCAs for short), in which only one controller provides coverage. For instance, in Fig. 2, area B can be either controlled by controller C1 or controller C2 and is thus classified as an MCA. Area A, however, is an NCA since it is covered by C1 only.

Regarding the data plane's mobile tier in which vehicles are included, the number of vehicles in one region is constantly changing. As such, the model should be able to scale to accommodate arbitrary number of vehicles.

For scalability reasons, instead of individual vehicles, the assignment problem of the mobile tier considers areas. At the same time, in order to provide fine-grained performance, we divide the region into cells and assign all vehicles within a cell to the same controller. We use a Boolean parameter $\eta_{f,c}$

to denote whether a cell f is within coverage of edge controller c or not, which is defined as

$$\forall f \in F, c \in C: \ \eta_{f,c} = \begin{cases} 1, & \text{if } \kappa_{f,c} \leq \omega_c \\ 0, & \text{if } \kappa_{f,c} > \omega_c \end{cases}, \tag{1}$$

where F is the set of all cells in the scheduled region, $\kappa_{f,c}$ is the maximum distance between the cell f and the controller c, and ω_c is the coverage radius of controller c.

A cell is an MCA only if $\sum_{c \in C} \eta_{f,c} > 1$, and it is an NCA if $\sum_{c \in C} \eta_{f,c} = 1$. With the definition of η, the set of MCA cells of the controller c can be denoted as $F_c = \left\{ f | f \in F, \ \eta_{f,c} = 1, \ \sum_{c \in C} \eta_{f,c} > 1 \right\}$, and the set of NCA cells of controller c is defined as $\widetilde{F}_c = \left\{ f | f \in F, \ \eta_{f,c} = 1, \ \sum_{c \in C} \eta_{f,c} = 1 \right\}$. The set of cross-coverage cells of two edge controllers is defined as $\tilde{A}_{c_i,c_j} = \left\{ f | f \in F, \ \eta_{f,c_i} = 1, \eta_{f,c_j} = 1 \right\}$. The two edge controllers are neighbors only if $\tilde{A}_{c_i,c_j} \neq \varnothing$. The neighbors of an edge controller are defined as $C_c^* = \left\{ c' | c' \in C_c^-, \tilde{A}_{c,c'} \neq \varnothing \right\}$, where C_c^- denotes the set of edge controllers except the controller c.

C. Control Delay

The mean delay of cells, which is a critical factor to show the performance of assignment, can be defined as:

$$D_f = \frac{\sum_{i \in I} D_i \zeta_{i,f}}{\sum_{i \in I} \zeta_{i,f}}, \ \forall f \in F, \tag{2}$$

where $\zeta_{i,f}$ is a Boolean parameter, which denotes whether forwarding device i is in cell f or not, and D_i is the control delay of the forwarding device i, which is function of the delay generated over routing, queuing, and processing of the control messages between the device and the associated controller. We define D_i to be the sum of the transmission delay d_{tr}, the propagation delay d_p, and the response delay d_c. We can safely assume the transmission rate of different controllers to be the same under the same power and channel gains, thus it is independent of the controller assignment policy $\pi_{f,c}$. The propagation and controller response delays depend, in turn, on $\pi_{f,c}$. Thus, the control delay of a forwarding device i is set equal to

$$D_i = d_{tr}(i) + \sum_{c \in C, f \in F} (d_p(i,c) + d_c) \pi_{f,c} \zeta_{i,f}, \ \forall i \in I, \tag{3}$$

where $\pi_{f,c}$ is a Boolean variable of controller assignment, which is equal to 1 when the cell f hosting the forwarding device is assigned to the controller c. In the following, we will introduce the transmission delay, propagation delay, and the response delay of the controller in detail.

1) Transmission Delay: The transmission delay refers to the time taken to transmit a complete packet on the channel. In the round-trip communication between forwarding devices and the corresponding controller, the transmission delay includes the uplink transmission from forwarding devices to the corresponding controller and downlink transmission from the controller to forwarding devices. The transmission delay depends on the packet size (G bytes) and the data rate of the interface (γ) as shown:

$$d_u(i) = \frac{G}{\gamma_u(i)}, \ \ d_d(i) = \frac{G}{\gamma_d(i)}, \ \forall i \in I, \tag{4}$$

where $\gamma_u(i)$ and $\gamma_d(i)$ correspond to bit rate on uplink and downlink interfaces, respectively. The definition of transmission delay can thus be represented as

$$d_{tr}(i) = d_u(i) + d_d(i), \ \forall i \in I. \tag{5}$$

2) Propagation Delay: The propagation delay depends on the distance between the controller and the forwarding device, and the speed of the signal propagation ϑ. It is defined as:

$$d_p(i,c) = \frac{\kappa_{i,c}}{\vartheta}, \ \forall i \in I, c \in C, \tag{6}$$

where $\kappa_{i,c}$ models the round-trip distance between the forwarding device i and the controller c.

3) Controller Response Delay: The controller response delay depends on the load and the processing rate of the controller. The load of an edge controller is mainly generated by its controlled elements (e.g., OpenFlow Packet-In events, Statistic Requests, Flow Eviction events). We define the load generated by forwarding device i on the control plane as l_i. We assume that controller assignment schemes do not have a big effect on the control load of forwarding devices, thus making l_i independent of the assignment schemes. The load of a cell f can be denoted as $l_f = \sum_{i \in I} l_i \zeta_{i,f}$ by summing over all devices hosted by the cell.

The load of the edge controller includes two parts, one is the load of the its NCA cells assigned to it by default as there is no other choice, and the other one is the load of its MCA cells assigned to it. Let L_c denote the load of an edge controller c, which can be expressed as:

$$L_c = L_{\widetilde{F}_c} + L_{F_c}, \ c \in C. \tag{7}$$

The load of the NCA cells of an edge controller can be denoted as $L_{\widetilde{F}_c} = \sum_{f \in \widetilde{F}_c} l_f$. The load of the MCA cells of an edge controller can be denoted as $L_{F_c} = \sum_{f \in F_c} l_f \pi_{f,c}$, determined by the assignment variables.

Packets arriving at controllers need to be queued for services like routing path computation. Each controller has a finite capacity of service, which can be modeled as an M/M/1/K queuing with a single finite queue size K. The packets arriving at a controller are processed according to the First-Come-First-Served order. For tractability reasons, we assume such packets to follow a Poisson arrival process whose rate at the controller is given by Equation (7). The definition of d_c can thus be denoted as:

$$d_c = d_q(c) + \frac{1}{\mu_c}, \ \forall c \in C, \tag{8}$$

which includes the queuing time $d_q(c)$ and processing time ($\frac{1}{\mu_c}$), and μ_c is the controller service rate in terms of packets/s.

Since data traffic is far more than the control traffic, the control load influence on communication infrastructures is negligible. The delay caused by queuing is represented as

$$d_q(c) = \begin{cases} \frac{\rho_c^2 + K\rho_c^{K+2} - K\rho_c^{K+1} - \rho_c^{K+2}}{L_c(1-\rho_c)(1-\rho_c^K)} & , \rho_c \neq 1 \\ \frac{K-1}{2\mu_c} & , \rho_c = 1 \end{cases}, \forall c \in C, \quad (9)$$

where $\rho_c = \frac{L_c}{\mu_c}$ denotes the ratio of the arrival and service rate.

Due to the limited queue in M/M/1/K model, the system suffers packet loss, especially when the arrival rate is more than the processing rate. The packet loss rate of one controller can be defined as:

$$\varrho_c = \begin{cases} \frac{(1-\rho_c)\rho_c^K}{1-\rho_c^{K+1}} & , \rho_c \neq 1 \\ \frac{1}{K+1} & , \rho_c = 1 \end{cases}, \forall c \in C. \quad (10)$$

When ρ_c is less than or even equal to 1, the packet loss rate is small and can be ignored according to this model. When ρ_c is more than 1, the packet loss rate is increased obviously.

IV. Problem Formulation

In a hierarchical distributed SD-IoV with multiple edge controllers, the assignment problem is focused on how to assign edge controllers dynamically to cells. In this section, we first model the dynamic edge controller assignment problem. Then, we formulate it as a multi-agent MDP game and try to solve it using MADRL.

We formulate the dynamic controller assignment problem in a way to minimize the sum of maximum cell delay over some time, which can be expressed as

$$minimize \sum_{t \in T} {D_t}^{max}, \quad (11)$$

where ${D_t}^{max}$ is the maximum delay of cells at time t, with definition as $D_t^{max} = \max_{f \in F} D_{f,t}$; $D_{f,t}$ is given in equation (2) for a time t, and T is the time period over which we perform the minimization. The variables of this objective are $\pi_{f,c}$ at each time instant. There are two constraints we should account for. One is $\pi_{f,c,t} \leq \eta_{f,c,t}$, which means that an edge controller can only control cells in its coverage area; and the other one is $\sum_{c \in C} \pi_{f,c,t} = 1$, which means each cell should be under the control of exactly one edge controller at a time.

The dynamic controller assignment problem is a time-series problem. At each time slot, the controller assignment policy, which can be viewed as a decision, should consider the state of SD-IoV to minimize the maximum cell delay shown in Equation (11). The decision making depends on the load of edge controllers, which impacts the queuing time, and on the distance between edge controllers and forwarding devices, which impacts the propagation time. The assignment policy taken in one step influences the next state (e.g., the occupied queue length) of SD-IoV.

In this paper, we formulate this problem as an MDP where a decision is made locally by each controller using a set of local measurements. This is because the communication between controllers is costly not only in time but also in communication resources. Divide and conquer becomes then a good choice, for which edge controllers are designed with intelligence to choose which cells they prefer to manage. Therefore, each edge controller can be viewed as an agent that makes decisions based on partial observation of the environment. This problem can be viewed as a multi-agent extension of MDPs called partially observable Markov games.

The multi-agent MDP is a discrete-time stochastic process, which can be defined by a quad-tuple $< S, A, R, P >$. In this tuple, S is the set of states, A is the set of actions, R is the set of rewards, and P is the probability of transition from state S to state S' based on action A. Next, we define these items.

1) **State space:** The global state is defined as $S = \{s_1, ..., s_N\}$, where s_c denotes the local state of edge controller c. The local states are defined as $s_c = \left\{L_{\widetilde{F}_c}, \mathbf{L}_{F_c}, d_q(c)\right\}$. The first term $L_{\widetilde{F}_c}$ is the total load of edge controller c coming from its NCA cells ($\widetilde{F}_c$). The second item $\mathbf{L}_{F_c}$ is the vector of load in cells of MCA (F_c), denoted as $\mathbf{L}_{F_c} = \left\{l_{f_1}, \cdots, l_{f_{|F_c|}}\right\}$, where $|F_c|$ is the number of cells in F_c. The last item $d_q(c)$ is the current queuing time of the edge controller.

2) **Action space:** The vector of local actions made by edge controllers can be defined as $A = \{a_1, ..., a_N\}$, N being the number of edge controllers. For each edge controller c at time t, the action is defined as a vector of control probabilities on its MCA cells F_c at time t, which is expressed as $a_c = \left\{g_{c,1}, ..., g_{c,|F_c|}\right\}$. The $g_{c,f}$ is the probability of cell f to be under the control of c, which is between 0 and 1, and it can be viewed as a competitive bid.

To get the aggregated action, edge controllers need to exchange messages with their neighbors. As neighbor controllers are located nearby, the propagation delay between them can be negligible (e.g. 0.027 ms for 8 km). The exchanged messages between two neighbors is the vector of the probability of their common cells. The message sent from c_i to c_j can be denoted as $\left\{g_{c_i,f} | f \in \tilde{A}_{c_i,c_j}\right\}$, and the reverse direction message is $\left\{g_{c_j,f} | f \in \tilde{A}_{c_i,c_j}\right\}$. When $\tilde{A}_{c_i,c_j} = \varnothing$, there is no messages sent for cooperative assignment. The aggregated action can be denoted as $M = \left\{m_1, ..., m_{|F_M|}\right\}$, which is the assignment of edge controllers to cells. The m_f is the ID of edge controller who will control the cell f, and $|F_M|$ denotes the number of all cross-coverage cells in one region. Each cell is under control of one edge controller at a time. From the view of each cell, they can greedily choose the edge controller offering the topmost bid, which is denoted as

$$m_f = \arg\max_{c \in C} g_{c,f}\eta_{f,c}, \forall f \in F_M. \quad (12)$$

By introducing $\eta_{f,c}$ into this formula, it can guarantee that only the edge controller, which covers cell f, has the opportunity to manage f.

3) **Transition probability:** Due to the mobility of vehicles and the stochastic nature of the control load, the probability of transition from state S to next state S' is also stochastic. The queuing delay of the next step depends on the control load at the next step and the actions made at the current step.

4) **Reward:** The value of reward depends on the current state and the taken action. Edge controllers try to cooperate to obtain a reward, sharing the same reward at each step. In the controller assignment problem of SD-IoV, the objective is to minimize the maximum delay of cells in the region; thus, the reward is defined as $r_t = -{D_t}^{max}$. Besides, the goal is to maximize the total expected return (long-term cumulative reward), which considers the possible future reward, and is denoted as:

$$R_t = \sum_{k=t}^{T} \gamma^{k-t} r_k, \forall t \in [1, T], \tag{13}$$

where $\gamma \in [0, 1]$ is a discount factor for future rewards to dampen the effect of future rewards on the action.

The above definitions of the environment state, action, and reward give rise to an MDP whose transition probability is unknown. A DRL can be used to solve this kind of problem where the objective is to maximize the cumulative rewards R_t at each of the time steps.

V. Learning Algorithm

In this section, we describe the algorithm based on Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [29] for the dynamic controller assignment problem, which is one kind of DRL algorithms.

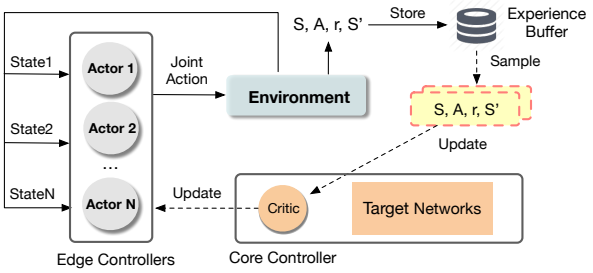


Fig. 3. MADDPG algorithm for dynamic edge controller assignment.

A. MADDPG Components

The MADDPG, as an extension algorithm of actor-critic [31], has both actors and critics. The actor is responsible of making decisions on assignment according to its local state, and the critic give value to tell how good these actions are considering the states. To stabilize the learning, target networks are designed in this algorithm, which offers target policies with delayed parameters. The actors should be located near to vehicles to reduce the delay of making decisions on actions, which can be deployed in edge controllers. To guarantee convergence, the critic and target networks can be designed to have a global view that can be deployed in the core controller. Hence, we adopt the framework with centralized training and decentralized execution, shown in Fig. 3. Besides, as the problem wants to solve in this paper is fully cooperative with common reward r_t between agents, one critic is enough to evaluate a joint action A based on a global state S.

1) Actors: The actors in MADDPG are responsible for making local actions a_i based on local observed states s_i. Considering that there are N edge controllers working as agents, which are neural networks with policies parameterized by $\theta^\mu = \{\theta_1^\mu, ...\theta_N^\mu\}$ called actor networks. Each agent i takes continuous policies μ_i (short for $\mu_{\theta_i^\mu}$) with regard to parameters θ_i^μ. For the deterministic policies, we can have $a_i = \mu_i(s_i|\theta_i^\mu)$ for local actions. Local actions are aggregated into a joint action M, which denotes the control relationship of each cell. Noticed that the real actions (take over or release management of cross-coverage cells) taken by local agents are known to their co-managed neighbors after negotiation.

2) Critic: The critic is fed with extra information about the global state S and policies A of all agents. The critic $Q(S, a_1, ..., a_N|\theta^Q)$ is defined as a centralized action-value function, whose parameters are θ^Q. It takes the global state S, which regroups the local states of all agents, as well as the actions of all agents $a_1, ..., a_N$ as input, which means that the critic know the actions of all agents. Its output is Q value to describe how good the joint action A is on state S.

3) Target Networks: The target policy of agent i is denoted as with μ_i' with parameters $\theta_i^{\mu'}$. The Q' is target critic with parameters $\theta^{Q'}$. These parameters are periodically updated with the most recent θ_i^μ and θ^Q:

$$\theta_i^{\mu'} \leftarrow \tau\theta_i^\mu + (1-\tau)\theta_i^{\mu'}, \theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}, \tag{14}$$

where τ is a coefficient between 0 and 1.

4) Experience Replay Buffer: The experience replay buffer $\mathcal{D}$ is used to store global information represented as tuples (S_t, A_t, r_t, S_t') of states, actions, rewards, and successor states at time slot t. Based on their timestamps, agents' local information is assembled into global information used to train actors and critic.

B. Workflow and Algorithm

After introducing the critical components of the MADDPG algorithm, the overall workflow of the assignment algorithm is presented in Algorithm 1, which consists of two main procedures: the acquisition of empirical data and training. In this algorithm, N_{ep} is the number of episodes, ϖ stands for the number of days used from the data set, and T is the number of steps in one day. The acquisition of empirical data (line 3-11) corresponds to the on-line process. To explore the state space, we add random noise to the output of the actor network (line 7), where $\mathcal{N}_t$ is the exploration noise at time t. According to the actions of agents, the joint action can be calculated with (12). Then, we execute the assignment policy M and obtain the set of rewards r and new state S'. At last, the experience is stored into experience replay buffer. In the training process (line 12-16), off-policy training is chosen, which uses a batch of samplings from the reply buffer. The actor and critic are updated respectively based on randomly selected samples. The parameters θ^Q of the critic network are updated to minimize the loss:

$$\mathcal{L}(\theta^Q) = \mathbb{E}_{S,A,r,S' \sim \mathcal{D}}[Q(S, A) - y]^2, \tag{15}$$

Algorithm 1: Dynamic edge controller assignment with MADDPG.

Initialize critic $Q(s,a|\theta^Q)$ and actors $\mu_i(s|\theta_i^\mu)$ with random weights θ^Q and θ_i^μ
Initialize target networks Q' and μ_i' with random weights $\theta^{Q'}$, $\theta_i^{\mu'}$
for episode = 1: N_{ep} **do**
  Receive initial state S
  for d = 1: ϖ do **do**
    for t = 1: T **do**
      Each agent select action with noise $\mathcal{N}$ $a_i = \mu_i(s_i|\theta_i^\mu) + \mathcal{N}_t$
      Calculate the assignment of cells m_f with (12)
      Execute the assignment with $M = \{m_1, ..., m_{|F|}\}$
      Obtain rewards r and the next state S'
      Store (S, A, r, S') in the experience replay buffer $\mathcal{D}$
  Sample a batch of random samples (S, A, r, S') from $\mathcal{D}$
  Set y with (16) and update the critic network by minimizing the loss (15)
  foreach agent $i = 1 : N$ **do**
    Update its actor network using the sampled deterministic policy gradient with (17)
  Update target network parameters with (14)

where the symbol $\mathbb{E}\{\cdot\}$ denotes the expectation value, and the tuple (S, A, r, S') is the sample from the experience replay buffer $\mathcal{D}$, and the y is defined as:

$$y = r + \gamma Q'(S', A')|_{a'_j=\mu'_j(s'_j)}, \tag{16}$$

in which A' is obtained with function $a'_j = \mu'_j(s'_j)$ of target actors. The parameters θ_i^μ of the actor network i are in turn updated using the sampled deterministic policy gradient:

$$\begin{aligned}\nabla_{\theta_i^\mu} J(\mu_i) &= \mathbb{E}_{S,A_i^- \sim \mathcal{D}}[\nabla_{\theta_i^\mu} Q(S,A)|_{a_i=\mu_i(s_i)}] \\ &= \mathbb{E}_{S,A_i^- \sim \mathcal{D}}[\nabla_{\theta_i^\mu} \mu_i(a_i|s_i) \nabla_{a_i} Q(S,A)|_{a_i=\mu_i(s_i)}],\end{aligned} \tag{17}$$

where A_i^- denotes all actions except the action of agent i. For the actor i, the action of agent i is obtained by the function $a_i = \mu_i(s_i)$ with the sample, and the other actions A_i^- are from the experience buffer. At last, target networks are updated with (14).

C. Computational Complexity

The deep neural network of actors can be represented as a matrix multiplication, and the complexity for each actor is $\mathcal{O}(|F_c|^2 H)$, where $|F_c|+2$ is the dimension of the local state, H is the number of hidden layers, and $|F_c|$ is the dimension of the output. The computational complexity resulting from the collaboration amongst neighboring edge controllers is $\mathcal{O}(|F_c|)$. Thus, the complexity to obtain the final assignment policy at each step is $\mathcal{O}(|F_c|^2 H)$. Note that, due to the distributed nature of the proposed approach, increasing the number of agents would not affect individual agent's computational complexity However, increasing the number of agents N costs more in terms of training time, as the complexity of critic neural networks and training procedure are $\mathcal{O}(N|F_c|H)$ and $\mathcal{O}(N_{ep} K_s N H |F_c|^2)$, respectively. Thus, the scalability of the training algorithms is affected by the number of agents N as well as the number of training episodes N_{ep} and the batch size K_s. In summary, increasing the number of agents would not increase the complexity of real-time controller assignment, but adds to training time.

Compared to single-agent DRL approaches, centralized decision making is more time consuming as it needs to gather and synchronize global data, as well as deploy assignment policies from the agent to the edge. In centralized DRL approaches, the complexity of its actor can be expressed as $\mathcal{O}(|F_M|^2 H)$, where $|F_M|$ denotes the total number of MCA cells. Single agent complexity is significantly higher than local agents' complexity in multi-agent DRL, as $|F_M|$ is much higher than $|F_c|$.

Regarding heuristic algorithms used in previous works, their computational complexity is also higher than our proposed approach. For example, the computational complexity of getting the assignment results in [14] is $\mathcal{O}(N_{ep}|F_M|log_2(|F_M|))$; and $\mathcal{O}(|F_M|^3)$ in [26].

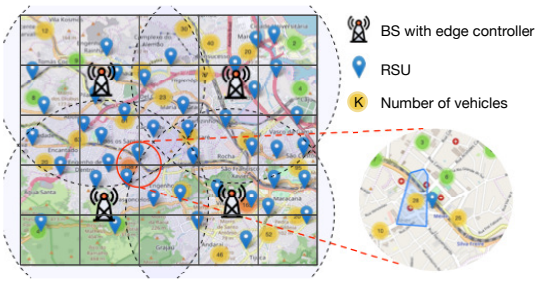


Fig. 4. Map of Rio de Janeiro.

VI. Simulation Experiments

A. Performance Evaluation

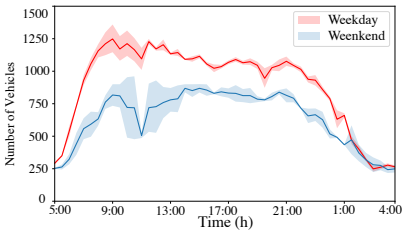
Fig. 4 shows the region used as the geographical footprint of the experiments. This region consists of 10×10 km^2 map of Rio de Janeiro, Brazil, with 42 RSUs nodes, and 4 BSs. Each BS hosts an edge controller, thus the number of edge controllers is $N = 4$. Besides, we assume that a core controller is hosted in the cloud and within the reach of edge controllers. We use a dataset of vehicle mobility traces available in [32]. This dataset offers real-time position data reported by buses from the city of Rio de Janeiro (24h format). In this experiment, we select vehicle data of one week $\varpi = 7$. To avoid frequent controller reassignment, in this paper, the assignment is updated at fixed time slots, e.g., half an hour is viewed as one step, which means that the agents take actions every half an hour. Thus, for one day, there are 48 steps in

total ($T = 48$) and 48 actions by each agent. Traffic, which is a key part of local states, has less use for training if it is with similar characteristics. Due to the characteristics of the dataset we used, and to efficiently obtain traffic with different characteristics, we set 30 minutes as the time interval, and thus, traffic has an obvious difference between the two intervals. Simulation parameters and their values are listed in Table II.

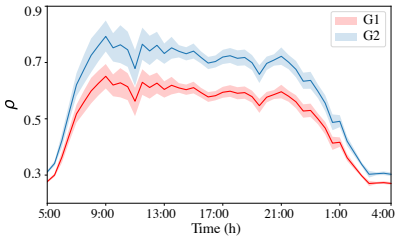
TABLE II
SIMULATION PARAMETERS.

Para.	Meaning	Simulation Value
N	The number of edge controllers	4
E	The number of hidden layers of actors and the critic	3
ϖ	The number of days used for learning	7
T	The number of steps in one day	48
ϑ	The propagation speed	3×10^8m/s
γ_u,γ_d	The wireless transmission speed	10 Mbps
G	The size of Packet-In and Flow-Mod	32 bytes, 56 bytes
ω_c	The coverage radius of controller	4 km
μ_c, μ_c^r	The mean processing rate of edge controller and core controller	10 packets/ms 40 packets/ms
K_c, K_c^r	The queuing length of edge controllers and core controllers	200 packets 800 packets
λ_v	Control traffic arrival rate of vehicles	G1: 10-30 packets/s G2: 10-35 packets/s
λ_r	Traffic arrival rate of RSUs	100-200 packets/s

We assume that each BS has a 4 km coverage radius, which also corresponds to the coverage radius of edge controllers. Latency between BSs to the core controllers depends the backhaul network connecting them. For example, it takes around 100 ms to connect with controllers through the Internet in [10], and it is around 4-10 ms through the OS3E network in [33]. Here, we set it between 2 ms and 8 ms. The controller queue size K is set to 200 packets for edge controllers and 800 packets for the core controller. We chose those values so that the last packet in the queue would be subject to 20 ms waiting time, which is considered the maximum latency for most of IoV applications are willing to tolerate [34]. However, for modern and latency stringent IoV application, e.g. automation and augmented reality applications, their requirements on end-to-end latency is less than 10 ms [3], [4]. Thus, their control latency is expected to be 3 ms, or even less, which is hard to be achieved by controllers deployed in cloud or edge of cloud. For traffic simulation, we choose 2 groups of traffic, namely group G1 and group G2, with different packet arrival rates from vehicles λ_v, as shown in Table II. We assume that new packets arriving from vehicles (λ_v) and RSUs (λ_r) obey uniform distribution, which is added to form the load of the controllers. Fig. 5(a) presents the mean number of vehicles on weekdays and weekends for one week from the data set [32]. Fig. 5(b) presents the ratio of the arrival and service rate of the load ρ with different groups of loads. Group 2 has a relatively higher load compared with Group 1. The number of vehicles and so the load ρ increase after 9:00 and drop after 21:00. According to the research in [35], the mainstream controllers have a limited processing rate, e.g. Ryu is 10 packets/ms, and ODL is around 40 packets/ms. Here, we set 10 packets/ms for the edge controllers, and 40 packets/ms for the core controller



(a) Number of vehicles.



(b) ρ of group G1 and G2.

Fig. 5. Simulation data at different times.

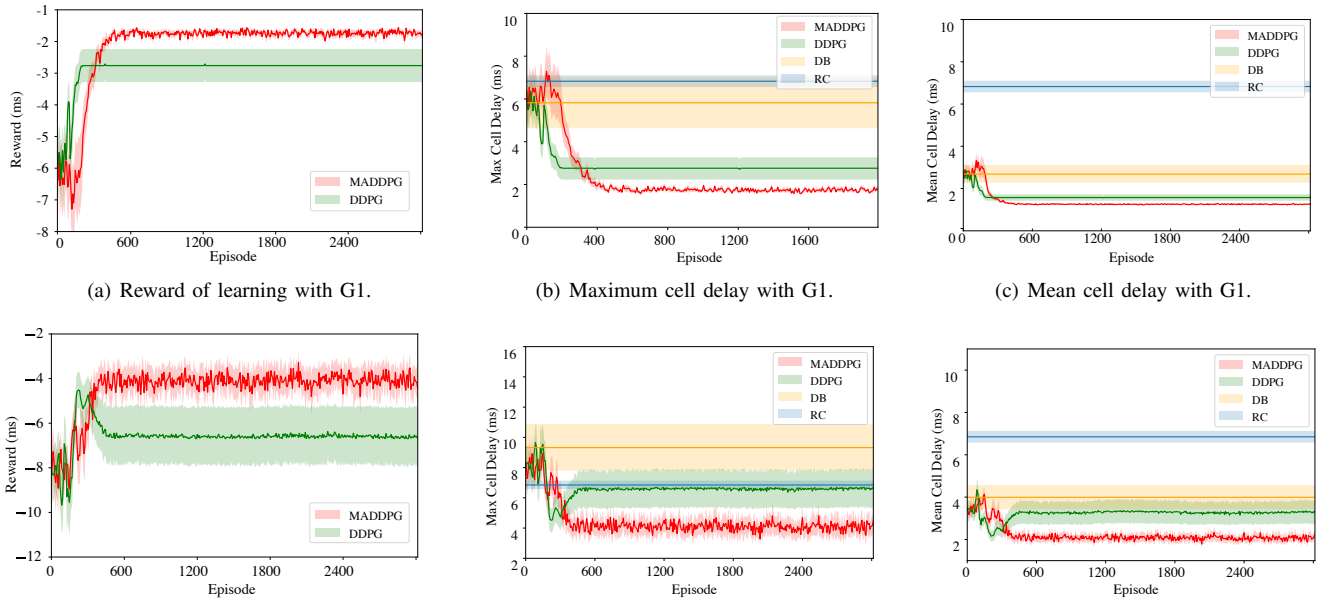
according to the traffic of G1. The reason is that the capacity of Ryu is enough for the traffic load G1 which can assure stringent control delay in 3 ms with good assignments.

B. Results and Performance Evaluation

We show simulation results for our proposed assignment methodology and compare it with three other approaches: (1) non-cooperative deep reinforcement learning (called DDPG for short), in which each edge controller uses deep deterministic policy gradient algorithm respectively and updates its bidding strategy without collaborating with other controllers; (2) distance-based approach (called DB for short), similar to [10], which considers the distance between controllers and forwarding devices in assignments; and (3) centralized approach with one remote controller (called RC for short). This approach is without edge controllers and the load is delivered by BSs directly to the remote controller. The control delay of RC includes 1) round-trip time from vehicles to the nearest BS and from the BS to the remote controller, and 2) the queuing and processing time of the remote controller. The control delay is increased due to the long-distance and routing with more hops compared to the edge controller. However, it can be compensated by the larger queuing size and higher packet processing rate of controllers, which can decease time costs in controllers. We set the capacity of the remote controller as a combination of all edge controllers with $\mu_c^r = 40$ packets/ms and $K_c^r = 800$ packets, which is much higher than edge controllers.

We first show the performance of the learning for different loads, namely G1 and G2, in Fig. 6. The horizontal axes of this figure are the learning episode of Algorithm 1. The subfigures show the performance of rewards, maximum cell delay, and mean cell delay with training, respectively.

Reward of learning with different loads. Figs. 6(a) and 6(d) present the mean rewards of MADDPG and DDPG with



(a) Reward of learning with G1. (b) Maximum cell delay with G1. (c) Mean cell delay with G1.

(d) Reward of learning with G2. (e) Maximum cell delay with G2. (f) Mean cell delay with G2.

Fig. 6. Performance in rewards and cell delays with learning.

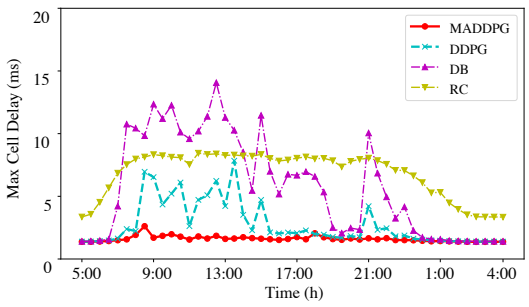
different loads. The Y-axis is the mean reward of one day, which can be defined as $\overline{R}_d = \frac{1}{T}\sum_{t=1}^{T} r_t, \forall d \in [1, \varpi]$. These curves also show the 68% confidence interval of the mean reward in 7 days. We can conclude that MADDPG has better performance of convergence, and it also obtains better rewards compared with DDPG because the DDPG lacks cooperation between edge controllers. Comparing the two sub-figures, we can observe that performance in lower load (G1) has better convergence with less fluctuation and higher rewards. The reason is that a higher load makes it more difficult to perform load balancing between different edge controllers, especially when the load ρ is near to or higher than 1. When ρ is near to 1, the queuing delay is increased obviously in sigmoid-shape with Equation (9), which brings more fluctuations with the same learning rate.

Cell delays with different loads. Figs. 6(b) and 6(e) show the mean of the maximum cell delay with different loads, which can be defined as $\overline{D}_d^{\,max} = \frac{1}{T}\sum_{t=1}^{T} \max_{f\in F} D_{f,t,d}, \forall d \in [1, \varpi]$. The curves also show the 68% confidence interval of the mean of the maximum cell delay for 7 days. Four methods are compared, which include MADDPG, DDPG, DB, and RC. DB and RC can be viewed as baseline methods, providing a performance comparison with the other ones. DB and RC in these figures also have confidence intervals for 7 days, and they are shown as straight lines as their values do not change with the learning process of MADDPG and DDPG. From these two sub-figures, we can see that with learning, both MADDPG and DDPG obtain lower maximum cell delay than DB and RC. Besides, MADDPG has relatively less delay compared with DDPG. Similar results can also be drawn from Figs. 6(c) and 6(f), whose vertical axis is the mean value for each day of the average cell delay calculated over all cells under control, which can be defined as $\overline{D}_d^{\,mean} = \frac{1}{T|F|}\sum_{t=1}^{T}\sum_{f\in F} D_{f,t,d}, \forall d \in [1, \varpi]$.

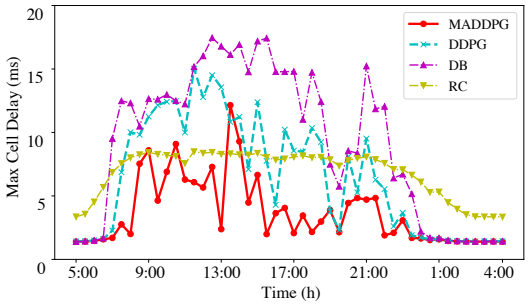
Fig. 7 shows the contrast results of 4 schemes at different times, including DB, RC, trained DDPG and MADDPG. The results include the maximum cell delay, the mean cell delay, the mean packet loss rate of controllers, and the load balance of edge controllers.

Cell delays at different times. Fig. 7(a) and Fig. 7(b) present the maximum delays of cells at different times of one day, averaged over all days at the same time, which can be defined as $\overline{D}_t^{\,max} = \frac{1}{\varpi}\sum_{d=1}^{\varpi} \max_{f\in F} D_{f,t,d}$. Compared to the alternative algorithms, MADDPG exhibits the lowest delay. In Fig. 7(a), the performance of MADDPG is better than others and keeps almost the same value at different time instants of the day. In Fig. 7(b), there are several points where the maximum delay of RC is smaller than MADDPG, for example, at time 13:30. The reason is that in the rush hour, the existing edge controllers can barely schedule the load to ensure that the load of each of them is not beyond the system capacity. Figs. 7(c) and 7(d) show the mean delay of cells at different times of one day, averaged over all days for the same time instant, and defined as $\overline{D}_t^{\,mean} = \frac{1}{\varpi|F|}\sum_{d=1}^{\varpi}\sum_{f\in F} D_{f,t,d}$. In these sub-figures, MADDPG performs better regarding the mean cell delay in both G1 and G2. Firstly, MADDPG has less mean cell delay comparing with DB and DDPG as it is able to balance the load between edge controllers. This reduction can be observed from time 8:00 to 22:00 with higher control loads. Secondly, compared with RC, the other three schemes have less mean cell delay, because their edge controllers are near to vehicles, which can save propagation time.

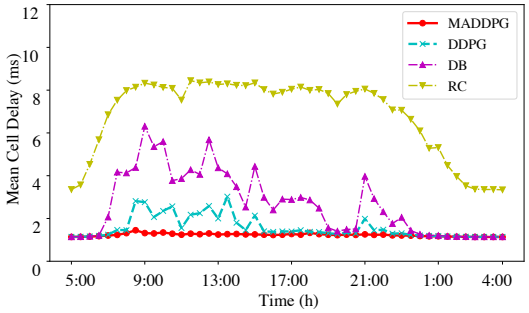
Packet loss rate at different times. Figs. 7(e) and 7(f) present the mean packet loss rate of cells at different times, averaged over days and controllers, and defined as



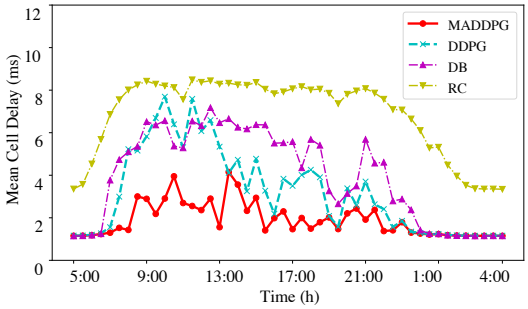
(a) Mean of maximum cell delay with G1.



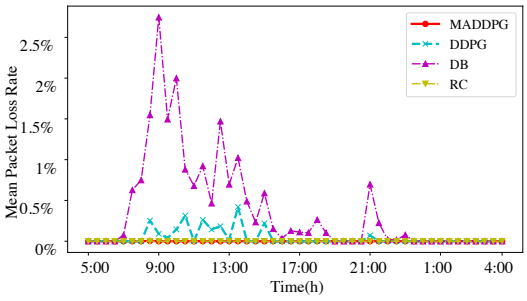
(b) Mean of maximum cell delay with G2.



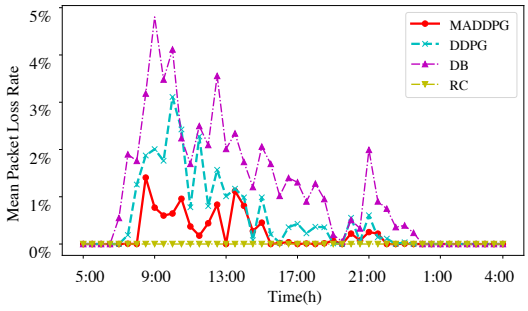
(c) Mean of mean cell delay with G1.



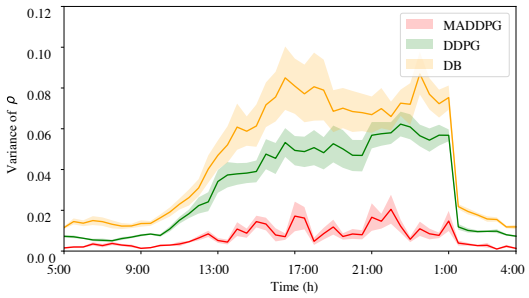
(d) Mean of mean cell delay with G2.



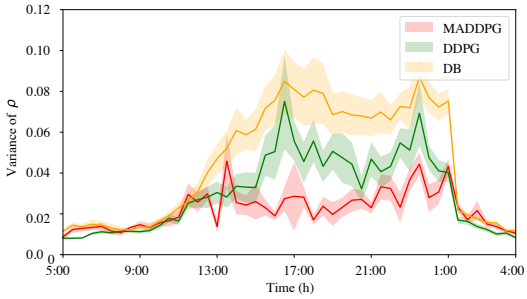
(e) Mean packet loss rate with G1.



(f) Mean packet loss rate with G2.



(g) Variance of ρ with G1.



(h) Variance of ρ with G2.

Fig. 7. Contrast results of cell delays, packet loss rates and controllers' load variances at different times.

$\overline{\varrho}_t = \frac{1}{\varpi N} \sum_{d=1}^{\varpi} \sum_{c \in C} \varrho_{c,t,d}$. Compared with DDPG and DB, MADDPG has a clear gain in reducing the packet loss rate. In Fig. 7(e), the packet loss rate is roughly zero when applying MADDPG. However, in Fig. 7(f), RC obtains the lowest packet loss rate compared with MADDPG because it has a more powerful controller with higher processing ability. Indeed, under this level of load (like G2), edge controllers cannot divide the load more equally to ensure that each of them is not overloaded. Thus, in such a case, we would suggest to upgrade or add new edge controllers into the environment if the packet loss rate is unbearable.

Variances of load. Figs. 7(g) and 7(h) show variances of the ratio of the arrival and service rate (ρ) among controllers as a function of the time of the day, again averaged over all days of the experiment. These variances can reflect the balance of load on different edge controllers. Compared with DDPG and DB, MADDPG has less variance and better load balancing among controllers in most cases. Due to only one remote controller used in RC, there is no load balancing problem for it.

VII. Conclusions and Future Work

In this paper, we proposed a distributed cooperative DRL based approach to solve the dynamic network controller assignment problem targeting connected vehicle services and applications, also known as IoV. Our DRL framework is based on a hierarchically distributed software-defined network control architecture and combines decentralized decision making by controllers located at the edge of the network and thus closer to vehicles and users, with centralized training using global information in order to achieve adequate tradeoff between control latency and global convergence.

Overall, our contributions include: (1) We formulate the dynamic controller assignment problem in SD-IoVs with the goal of minimizing control delay given vehicle location and control traffic load; (2) We propose a real-time distributed cooperative assignment approach, in which controllers make local decisions and coordinate with neighboring controllers; (3) We propose a centralized training approach using global information to attain optimal local assignment yet ensuring global convergence; (4) We evaluate the performance of the proposed dynamic controller assignment approach through simulations. Our results show the potential benefits of the proposed scheme in terms of reduced control delay and packet loss rate using real vehicle trace data from [32]. We make our code, as well as the traces used in our simulation available on GitHub [36].

In this work, we assumed that edge controllers are located in stationary infrastructure, e.g., RSUs and BSs. However, our solution can be extended to support mobile edge controllers (e.g., when edge controllers are deployed in vehicles and/or drones) by adding state information (e.g., mobile edge controller location). Other interesting directions for future work include combining dynamic controller assignment with dynamic placement of edge controllers to further reduce control delay, considering the number of handover times of vehicles, as well as comparing our framework against approaches proposed for other domains, e.g., data centers.

Acknowledgment

This work was partly funded by Inria, supported by the French ANR "Investments for the Future" Program reference #ANR-11-LABX-0031-01, and UNICAMP, through the FAPESP Grant number #2017/50361-0, both in the context of the DrIVE #EQA-041801 associated team.

References

[1] M. Gerla, E.-K. Lee, G. Pau, and U. Lee, "Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds," in 2014 IEEE world forum on internet of things (WF-IoT). IEEE, 2014, pp. 241–246.

[2] 5GAA, "5GAA releases white paper on c-v2x use cases: Methodology, examples and service level requirements," 5GAA white paper, 2019.

[3] Z. H. Mir and F. Filali, "LTE and IEEE 802.11 p for vehicular networking: a performance evaluation," EURASIP Journal on Wireless Communications and Networking, vol. 2014, no. 1, p. 89, 2014.

[4] 3GPP TS 23.203, "Policy and charging control architecture," Available online at http://www.3gpp.org/DynaReport/23203.htm/, accessed 22 Dec. 2019.

[5] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," IEEE Communications Surveys & Tutorials, vol. 16, no. 3, pp. 1617–1634, 2014.

[6] C. Jiacheng, Z. Haibo, Z. Ning, Y. Peng, G. Lin, and S. Xuemin, "Software defined internet of vehicles: Architecture, challenges and solutions," Journal of communications and information networks, vol. 1, no. 1, pp. 14–26, 2016.

[7] D.-J. Deng, S.-Y. Lien, C.-C. Lin, S.-C. Hung, and W.-B. Chen, "Latency control in software-defined mobile-edge vehicular networking," IEEE Communications Magazine, vol. 55, no. 8, pp. 87–93, 2017.

[8] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama et al., "Onix: A distributed control platform for large-scale production networks," in OSDI, vol. 10, 2010, pp. 1–6.

[9] M. A. Togou, D. A. Chekired, L. Khoukhi, and G.-M. Muntean, "A hierarchical distributed control plane for path computation scalability in large scale software-defined networks," IEEE Transactions on Network and Service Management, vol. 16, no. 3, pp. 1019–1031, 2019.

[10] K. S. K. Liyanage, M. Ma, and P. H. J. Chong, "Controller placement optimization in hierarchical distributed software defined vehicular networks," Computer Networks, vol. 135, pp. 226–239, 2018.

[11] A. Kaul, L. Xue, K. Obraczka, M. A. Santos, and T. Turletti, "Handover and load balancing for distributed network control: applications in ITS message dissemination," in 27th International Conference on Computer Communication and Networks (ICCCN). IEEE, 2018, pp. 1–8.

[12] A. Kaul, K. Obraczka, M. A. Santos, C. E. Rothenberg, and T. Turletti, "Dynamically distributed network control for message dissemination in ITS," in 21st International Symposium on Distributed Simulation and Real Time Applications (DS-RT). IEEE, 2017, pp. 1–9.

[13] G. Wang, Y. Zhao, J. Huang, and W. Wang, "The controller placement problem in software defined networking: A survey," IEEE Network, vol. 31, no. 5, pp. 21–27, 2017.

[14] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic SDN controller assignment in data center networks: stable matching with transfers," in Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on. IEEE, 2016, pp. 1–9.

[15] T. Wang, F. Liu, and H. Xu, "An efficient online algorithm for dynamic SDN controller assignment in data center networks," IEEE/ACM Transactions on Networking, vol. 25, no. 5, pp. 2788–2801, 2017.

[16] M. J. Abdel-Rahman, E. A. Mazied, K. Teague, A. B. MacKenzie, and S. F. Midkiff, "Robust controller placement and assignment in software-defined cellular networks," in International Conference on Computer Communication and Networks (ICCCN). IEEE, 2017, pp. 1–9.

[17] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," IEEE Signal Processing Magazine, vol. 34, no. 6, pp. 26–38, Nov 2017.

[18] T. Yuan, W. B. da Rocha Neto, C. Rothenberg, K. Obraczka, C. Barakat, and T. Turletti, "Harnessing machine learning for next-generation intelligent transportation systems: A survey," 2019.

[19] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," IEEE Communications Surveys & Tutorials, vol. 21, no. 4, pp. 3133–3174, 2019.

[20] H. Ye, G. Y. Li, and B.-H. F. Juang, "Deep reinforcement learning based resource allocation for v2v communications," IEEE Transactions on Vehicular Technology, vol. 68, no. 4, pp. 3163–3173, 2019.

[21] M. J. Abdel-Rahman, E. A. Mazied, A. MacKenzie, S. Midkiff, M. R. Rizk, and M. El-Nainay, "On stochastic controller placement in software-defined wireless networks," in IEEE Wireless Communications and Networking Conference (WCNC). IEEE, 2017, pp. 1–6.

[22] G. Wang, Y. Zhao, J. Huang, and Y. Wu, "An effective approach to controller placement in software defined wide area networks," IEEE Transactions on Network and Service Management, vol. 15, no. 1, pp. 344–355, 2017.

[23] P. Sun, Z. Guo, G. Wang, J. Lan, and Y. Hu, "MARVEL: Enabling controller load balancing in software-defined networks with multi-agent reinforcement learning," Computer Networks, p. 107230, 2020.

[24] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in Proceedings of the First Workshop on Hot Topics in Software Defined Networks, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 7–12.

[25] M. Caria, T. Das, A. Jukan, and M. Hoffmann, "Divide and conquer: Partitioning OSPF networks with SDN," in Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on. IEEE, 2015, pp. 467–474.

[26] T. Yuan, X. Huang, M. Ma, and J. Yuan, "Balance-based SDN controller placement and assignment with minimum weight matching," in IEEE International Conference on Communications (ICC). IEEE, 2018, pp. 1–6.

[27] Q. Qin, K. Poularakis, G. Iosifidis, and L. Tassiulas, "SDN controller placement at the edge: Optimizing delay and overheads," in IEEE INFOCOM. IEEE, 2018, pp. 684–692.

[28] X. Huang, T. Yuan, G. Qiao, and Y. Ren, "Deep reinforcement learning for multimedia traffic control in software defined networking," IEEE Network, vol. 32, no. 6, pp. 35–41, 2018.

[29] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in Advances in Neural Information Processing Systems, 2017, pp. 6379–6390.

[30] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "A survey and critique of multiagent deep reinforcement learning," Autonomous Agents and Multi-Agent Systems, vol. 33, no. 6, pp. 750–797, 2019.

[31] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 42, no. 6, pp. 1291–1307, 2012.

[32] D. Dias and L. H. M. K. Costa, "CRAWDAD dataset coppe-ufrj/riobuses (v. 2018-03-19)," Downloaded from https://crawdad.org/coppe-ufrj/RioBuses/20180319, Mar. 2018.

[33] G. Wang, Y. Zhao, J. Huang, and Y. Wu, "An effective approach to controller placement in software defined wide area networks," IEEE Transactions on Network and Service Management, vol. 15, no. 1, pp. 344–355, 2018.

[34] X. Wen, B. Yang, Y. Chen, L. E. Li, K. Bu, P. Zheng, Y. Yang, and C. Hu, "Ruletris: Minimizing rule update latency for TCAM-based SDN switches," in 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2016, pp. 179–188.

[35] L. Zhu, M. M. Karim, K. Sharif, F. Li, X. Du, and M. Guizani, "SDN controllers: Benchmarking & performance evaluation," arXiv preprint arXiv:1902.04491, 2019.

[36] https://github.com/TingtingYuan/maddpg_controller_assignment.git.