

# Intelligent and Resizable Control Plane for Software Defined Vehicular Network : A Deep Reinforcement Learning Approach

karima Smida (✉ [karima.smida@supcom.tn](mailto:karima.smida@supcom.tn))

SupCom: Universite de Carthage Ecole Superieure des Communications de Tunis

<https://orcid.org/0000-0001-5548-4100>

Hajer Tounsi

SupCom: Universite de Carthage Ecole Superieure des Communications de Tunis

Mounir Frikha

SupCom: Universite de Carthage Ecole Superieure des Communications de Tunis


---

## Research Article

**Keywords:** Software Defined Vehicular Networks, Distributed control plane, Quality of Service, Deep Reinforcement Learning.

**Posted Date:** August 30th, 2021

**DOI:** <https://doi.org/10.21203/rs.3.rs-276280/v1>

**License:**  This work is licensed under a Creative Commons Attribution 4.0 International License.  
[Read Full License](#)

---

**Version of Record:** A version of this preprint was published at Telecommunication Systems on October 23rd, 2021. See the published version at <https://doi.org/10.1007/s11235-021-00838-2>.

# Intelligent and Resizable Control Plane for Software Defined Vehicular Network : A Deep Reinforcement Learning Approach

Karima Smida<sup>1</sup> · Hajer Tounsi<sup>1</sup> · Mounir Frikha<sup>1,2</sup>

Received: date / Accepted: date

---

K. Smida  
✉ karima.smida@supcom.tn

H. Tounsi  
✉ hajer.tounsi@supcom.tn

M. Frikha  
✉ m.frikha@supcom.tn

<sup>1</sup> Carthage University, Sup'com, Tunisia

· <sup>2</sup> King Faisal University (KFU), Saudi Arabia

**Abstract** Software-Defined Networking (SDN) has become one of the most promising paradigms to manage large scale networks. Distributing the SDN Control proved its performance in terms of resiliency and scalability. However, the choice of the number of controllers to use remains problematic. A large number of controllers may be oversized inducing an overhead in the investment cost and the synchronization cost in terms of delay and traffic load. However, a small number of controllers may be insufficient to achieve the objective of the distributed approach. So, the number of used controllers should be tuned in function of the traffic charge and application requirements. In this paper, we present an Intelligent and Resizable Control Plane for Software Defined Vehicular Network architecture (IRCP-SDVN), where SDN capabilities coupled with Deep Reinforcement Learning (DRL) allow achieving better QoS for Vehicular Applications. Interacting with SDVN, DRL agent decides the optimal number of distributed controllers to deploy according to the network environment (number of vehicles, load, speed etc.). To the best of our knowledge, this is the first work that adjusts the number of controllers by learning from the vehicular environment dynamicity. Experimental results proved that our proposed system outperforms static distributed SDVN architecture in terms of end-to-end delay and packet loss.

**Keywords** Software Defined Vehicular Networks, Distributed control plane, Quality of Service, Deep Reinforcement Learning.

## 1 Introduction

Network softwarization is a key trend of current network evolution. The aim of network programmability and virtualization technologies, known as Software Defined Networks (SDN), is to offer more flexibility, scalability, and reliability making in turn the network services deployment faster and cheaper. The main concept behind SDN is to decouple data forwarding and its control contrary to traditional networks. Control decisions are taken by the logically centralized network brain called controller that has the global network view. Passive forwarding devices execute rules received from the controller. For robustness and scalability purposes, the logically centralized control plane is physically distributed to different controllers throughout the network [1]. Vehicular networks technologies and the resulting variety of applications have been a hot topic in recent research. They come with huge promising interests for future Intelligent Transportation System. This paradigm has gone beyond leveraging the rapid development of communication and computation technologies to architectural advancements such as SDN, Network Function Virtualization (NFV), Edge cloud and Fog networking. Recently, we talk about Software Defined Vehicular Networks (SDVN) [2]. Solutions for scalability challenges in centralized and distributed SDN are extended to SDVN [3]. Despite the performance improvement offered by the distributed approach in SDVN [4], the choice of the optimal number of controllers that ensures better quality of service while minimizing the cost of inter-controllers synchronization stills to be addressed [5]. Indeed, even if the control plane is physically distributed, its operation remains logically centralized since the controllers exchange reports on their corresponding domains to have a global state of the whole network and act as a single controller. This issue can have a negative impact on delay-sensitive vehicular applications especially road-safety services. We have shown in [5] that deploying a wrong number of controllers increases the packet end-to-end delay and the

packet loss ratio. In fact the performance degradation is due either to the overload of these controllers (in case of an under sizing of the number of controllers) or because of the increased synchronization delay between them (in case of an over sizing of a number of controllers). The criticality of such applications with the increase in the complexity of vehicular networks due to their dynamic and large scale nature have fueled the need of an adaptive control plane. However, adjusting the optimal number of controllers, in real time and in an ever-changing environment, is difficult and requires intelligence, agility, automation, and the ability to handle a large number of constantly changing states. Few works have focused on estimating the optimal number of controllers in a distributed SDN architecture. Most of proposals have instead compared the performance of different implementations of centralized or distributed controllers [1], otherwise they have addressed the controller placement problem [6]. Indeed, authors in [7] proposed an adaptively adjusting and mapping controller (AAM-con) in data center networks. Their proposal consists into two parts: one algorithm, which consists in reducing the number of controllers according to network demand and increasing it to the initial number when controller overload is noted. The second part consists in adopting network community theory to select a key switch to place the controller that is closer to switches it controls in a subnet. In [8], they have proposed an elastic distributed SDN controller. It consists in resizing the number of controllers based on the sum of the mean and standard deviation of CPU usage over the stored values for each controller, compared to two thresholds (low, high). Once the number of controllers is modified (decreases or increases) then a load rebalancing will be performed between the controllers of the new pool.

On another side, [Machine Learning](#) (ML) has seen great success in solving various problems and enabling automation in network operation and management including routing, traffic prediction and congestion control, resource

allocation, QoS and QoE management, and network security [9]. Deep reinforcement learning (DRL) is a category of ML. It brought big improvements compared to traditional algorithms of ML in several areas such as problems that were previously insoluble by reinforcement learning (RL) due to their infinite state space [10]. Combining supervised and reinforcement learning, DRL performs simultaneously feature learning (from examples) and policy learning (from experience). A software agent aims to discover the most adequate actions to get closer to a target policy by maximizing the received rewards for every action choice. In the long run, the agent will learn the entire configuration updates (actions) that result in such a target strategy. DRL power has been exploited in networking domains too. For instance, authors in [11] used DRL to optimize resource allocation in vehicle-to-vehicle communications. In [12], authors were interested in leveraging DRL in routing protocols to optimize inter-controller synchronization in SDN networks.

Our approach to improve the Distributed SDN control is to use DRL to achieve an intelligent and resizable (elastic) control plane for distributed SDVN architecture. We propose an Intelligent and Resizable Control Plane for Software Defined Vehicular Network (IRCP-SDVN) which consists in adjusting the number of the distributed controllers according to the real traffic load of the vehicular network. Indeed, the proposed DRL agent learns vehicles density and speed from the vehicular network and predicts the efficient number of controllers based on a reward function which is a compromise between responsiveness (end-to-end delay) and reliability (packet loss) and taking into account the vehicular applications stringency in terms of QoS.

The rest of this paper is organized as follows. In [Section 2](#), we give some background on SDVN and controllers placement and synchronization problems in distributed SDN architecture. In [Section 3](#), we review DRL basics and its application in SDN networks. [Section 4](#) presents our proposed architecture (IRCP-SDVN). Our experimental evaluation results are shown in [Section 5](#). Finally, we conclude our work in [Section 6](#).

## 2 Distributed Software Defined Networks

### 2.1 Overview

SDN is a recent paradigm based on centralized Management, programmability and network abstraction and automation. It is one of the most important architectures of next generation network. It composes of three planes [13]. The control plane, defined as a SDN controller, is the abstraction layer that provides a logically centralized control to monitor and manage a geographically distributed network. The SDN controllers role is to translate networking applications policies from the upper layer into

rules and push them to forwarding devices such as OpenFlow switches, via southbound interface. OpenFlow [14] is widely used for control and data plane communication. Interaction with the upper layer or application plane is performed through the northbound interfaces such as [REST API \(Representational State Transfer Application Programming Interface\)](#) [15]. To solve scalability issues, the logically centralized control is distributed on multiple controllers that communicate via east-west bound APIs [1].

The logically centralized control is shared among physically distributed controllers, each of which manages a subnet referred to as a domain. These domain controllers have to synchronize with each other to maintain a logically centralized consistent global view. There are two state-of-the-art shared data structure models: strong consistency model or Single Data Ownership (SDO) and eventual consistency model or Multiple Data Ownership (MDO). In SDO model, a single controller referred to as data owner is the only responsible for the update of the shared data structure. Any update operation on any local data structure performed by any domain controller must be forwarded to the data owner. In other words, each domain controller has a local copy of the main data structures (global view) but any read/write operation is obligatory forwarded to the corresponding leader. This latter ensures the update replication to its followers and awaits the acknowledgment of the majority of them before the update validation. This centralized approach strengthens data consistency and facilitates its management exploiting the distributed nature of the data structures only for failover [16]. In MDO model, every controller has a local copy of the network view and can run locally updates then advertises them in an asynchronous way to all other controllers. Hence, inconsistent network views are noted until each controller updates the others. Thus, higher network availability is achieved at the cost of temporary inconsistency according to the CAP theorem [17]. As famous examples, the two state-of-the-art distributed controllers Open Daylight (ODL) [18] and Open Network Operating System (ONOS) [19] adopt the Raft consensus algorithm [20] to achieve strong consistency for the shared data structures. In fact, most of ODL versions are endowed with clustering service to allow the deployment of multiple instances of the controller following Raft algorithm [21]. ONOS provides eventual consistency model too, through the so called anti-entropy algorithm based on a gossip protocol [22].

Many challenges of traditional vehicular communication networks are resolved thanks to SDN giving birth to SDVN. SDVN has achieved big improvements in terms of quality of service (QoS), resource allocation and network optimization in such dynamic (variable speed and density) and stateless environment [23]. Scalability in vehicular network is a hot topic for researchers. For that end,

several distributed SDN architectures were proposed and have proved better performances [3]. A general distributed SDVN architecture is presented in figure 1. Vehicles are in the lowest level of data plane, they are the sensing data source. The top of data plane includes wireless forwarding devices like Road Side Units (RSU), Wi-Fi access points (AP), 3G/LTE base stations (BS), etc.

## 2.2 Distributed Control Plane Related Work

In order to enhance the control plane, some works have focused on scaling up centralized controllers. Beacon [24] and McNettle [25] are famous examples of high performance centralized controllers. Indeed, Beacon can run a network of 80 servers, 320 virtual machines and 20 physical switches wired as 4level-fat-tree. A single McNettle controller has 46 cores achieving 14 million flows per second and can serve up to 5000 switches. However, scaling up the single controller does not resolve the performance drop caused by distance between switches and single controller in large scale networks neither the problem of single point of failure [3]. Thus distributing the control plane among multiple controllers became a necessity. As a consequence, many efforts are directed to distributed SDN architectures as efficient solution to maintain and improve QoS in large scale networks especially SDVNs [3].

The main interest is given to the distributed control plane optimization through investigating the controller synchronization and placement problems. Authors in [26] intend to optimize controller placement in hierarchical distributed SDVN in order to minimize controller-node latency. They privilege a static placement of controllers at the RSU level. This choice aims to have the vehicles, particularly in dense regions like road intersections, closest to the controllers. Results proved that this approach reduces delays compared to controller placement in the cloud as well as a random placement. However, this static strategy in controller placement may be not suitable with road traffic variation in time-space dimension. For example, a sudden traffic jam may occur far from the chosen places for controllers because of an accident. In this case, emergency messages may face high delays. Also, experiments are done on simplified small network which may question the effectiveness of the solution for large scale SDVN. The controller placement problem is also investigated in [27] where a dynamic strategy is promoted taking into account the dynamicity of vehicular network topology. Authors of this work carried out experiments with a realistic traffic scenario and proved experimentally the limitations of static controller placement in SDVN. Yet in our opinion, deploying controllers on RSUs may generate additional expenses because RSUs are originally dedicated hardware to ITS (Intelligent Transportation System) which functionality is usually limited to data exchange with OBUs (On

Board Units) in their communication ranges [28]. Thus, their capacities are not adapted to deploy controllers and they require enormous investments to equip them with such capacities. Besides, knowing the critical role of a controller as a brain of the network, we still support the principle of deploying controllers in data centers for security and management reasons. Authors in [16] investigate the role of inter-controller traffic in distributed SDN controllers placement. Analytical model is developed to assess response time perceived by switches taking into consideration the inter-controller and the switch to controller OpenFlow traffic. In both existing models SDO and MDO, the dominance of the synchronization latency between controllers is highlighted and an optimal placement for the leader controller in SDO is defended to find the best delay tradeoff in a strong consistency model.

The model is evaluated for Internet Service Provider (ISP) topologies where they evaluate switch-to-controller delay and controller-to-controller delay for multiple controller locations. Thus, they proved the importance of the interaction between the controllers in the placement problem without focusing on the number of controllers. Indeed, their object is to find the optimal placement for a given controller number and not the optimal number of controllers for variable network load like the case of dynamic vehicular networks.

Authors in [29] attempt also to achieve efficient placement of controllers in a SDN hierarchical architecture based on minimizing control plane delay. They formulated optimization problem as controller-switch association and controller capacity sub-problems. They proved the effectiveness of the proposed algorithm in an SDN scenario consisting of randomly deployed SDN switches in 2000 km 2000 km square region. Their algorithm achieved lower values of control plane delay compared to proposed algorithm in [30]. The latter work also investigates multi-controller placement problem from the perspective of latency minimization based on network partition technique. The problem is addressed by an optimized K-means algorithm. Results are 2.437 times better than the average latency achieved by the standard K-means clustering method. Another work [31] investigates controller placement to address network reliability and reduce delay for critical applications. The proposed heuristic algorithm uses robustness factors namely algebraic connectivity, network criticality, load centrality, and communicability. The proposed approach achieved high network resilience against targeted attacks and low controller to controller delays outperforming k-median and k-center baseline methods. Authors of [32] consider the communication delay, the utilization of control plane, and the distribution of controller workload to tackle the control plane placement problem. They develop a new algorithm that integrates the genetic algorithm (GA) and the gradient descent (GD) optimization method. They proved the efficiency of their approach on small-scale and large-scale networks by finding a trade-



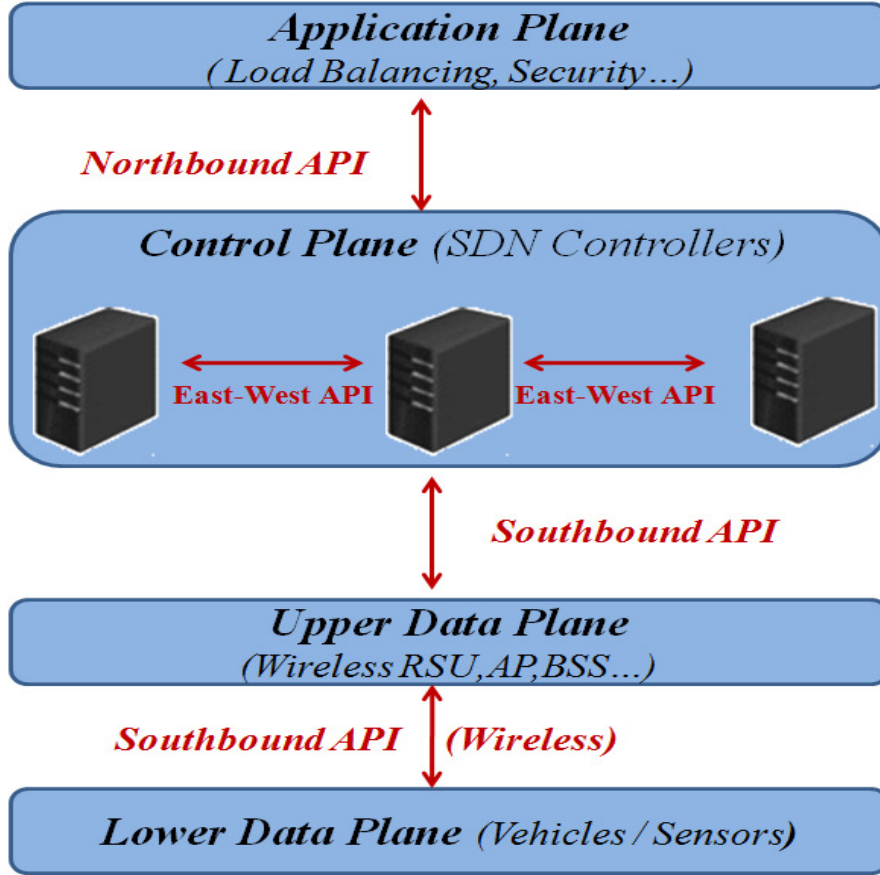


Fig. 1 Distributed SDVN Architecture

off between the network response time and the control plane utilization. However, these works aim to determine optimal locations for a given number of controllers to optimize the overall network performance without looking for the optimal number of deployed controllers. Focusing on minimizing inter-controller delay, authors in [33] use the concept of Minimum Connected Dominating Set (MCDS) to minimize delay time between controllers and the delay time between controller and switch then assign the number of controllers that satisfy the calculated minimum delay. Efficiency of their approach is proved on small topologies like India35 and ATT North America and medium topologies like Bell Canada and Germany50.

Synchronization between distributed controllers in large scale and, especially, highly dynamic networks incurs high costs in terms of extra-delay which is not suitable with delay-sensitive applications. It is therefore crucial to adjust the number of distributed controllers to the actual load of the network. An adaptively distributed SDN controller in data center networks (AAMcon) is proposed in [7]. The number of distributed controllers is increased and decreased based on controller overload level using NFV technology. Besides, a switch-to-controller mapping scheme based on complex network community theory is adopted. Good achieved results proved the importance of tuning controller number and placement according to

network demands. However, AAMcon is tested only in the context of datacenter networks. Experiments in dynamic environments such as vehicular networks are missing to prove its efficiency in such challenging conditions.

Authors in [12] were interested in distributed SDN and dealt with the synchronization problem between controllers based on a **Deep Reinforcement Learning (DRL)** approach. They propose a routing-focused DQ (Deep-Q) Scheduler to get the best policy that optimizes controller synchronization scheme over a time period. They prove that it outperforms anti-entropy algorithm. They extended their work by proposing a policy design for Controller Synchronization based on DRL too [34]. They proved the efficiency of their DRL policy in maximizing the performance enhancements brought by controller synchronizations in terms of delay over a period of time. However, they only consider eventual consistency model.

In a previous work [5], we investigated the performance of distributed SDVN in variable density vehicular scenarios. Experiments have shown the benefit of using multiple controllers in terms of delay but also they confirmed that the use of a non-optimal number of controllers incurs high costs in terms of extra-delay resulting in QoS drop. In fact, inter-controller synchronization increases the End-to-End

delay that is not tolerated for critical applications.

In this paper, we propose to optimize the number of controllers used in distributed SDN according to the density and the load of the vehicular network. We propose a Deep Reinforcement Learning (DRL) approach that learns from the vehicular network environment and decides the optimal number of controllers. The main contributions of our work are as follows:

- First, we make deployment of controllers in distributed control plane tunable and adjustable in function of data plane requirements in terms of QoS.
- Second, this adjustment is made automatically and in real time thanks to AI by using Deep Reinforcement Learning. Hence, our system learns in real time from its experience in the vehicular environment to make adequate decision (the effective number of controllers to deploy for better QoS) and do not need prior knowledge about the network. This fits vehicular networks nature (high dynamicity).
- Finally, our system is open and can be deployed with any kind of distributed controller independently of its capacity thanks to the DRL agent that decides the effective number of used controllers based on the achieved QoS.

Table 1 presents the positioning of our work in relation to the cited related ones.

### 3 Deep Reinforcement Learning and SDN

SDN brings new opportunities to apply Machine Learning (ML) techniques inside networks through its capabilities such as centralized control, global network view etc. Among the common ML algorithms applied to SDN we cite supervised learning like Neural Networks, non supervised learning such as K-means algorithm and Reinforcement Learning (RL). DRL inherits the main advantage of RL which is the ability to work without prior knowledge of a mathematical model of the environment. This makes it very suitable for dynamic networks especially with its capability of solving problems with high-dimensional state space [35].

#### 3.1 DRL application in SDN

DRL has attracted much research interest. A comprehensive survey [10] introduces this field and covers its main algorithms such as the deep Q-network and asynchronous advantage actor-critic. The advantages of DRL and the current areas of research within it are also described. Many works are focusing on leveraging DRL in making SDN networks more intelligent and agile by adding

automatic and autonomous reactivity. The idea was first proposed in Knowledge Defined Networks (KDN) [35] where a knowledge plane is added on top of the control plane in SDN architecture. This paradigm mixes neural networks with reinforcement learning to provide automation and optimization. In [36] a DRL mechanism for SDN, called DROM, is proposed for routing optimization. Performance metrics like delay and throughput are improved thanks to the provided black-box optimization in continuous times. Applying DRL keeps showing better results in other SDN-based fields like IoT and vehicular networks resolving scalability challenges. Authors in [37] propose a DRL based QoS-aware Secure routing Protocol (DQSP) to address security and latency issues in large-scale SDN-IoT networks. A dueling DRL-based trust management scheme for SDVN is proposed in [38] to face performance degradation caused by malicious nodes. An agent is deployed on a centralized SDN controller to learn the most trusted routing path by deep neural network. The designed model is used to evaluate neighbors behavior when forwarding routing information.

#### 3.2 Deep Reinforcement Learning Fundamentals

##### 3.2.1 Reinforcement Learning

Reinforcement learning (RL) is one of Machine Learning (ML) approaches that is different from supervised, unsupervised and Semi-supervised learning techniques. It is based on an agents interactions with uncertain external environment which it explores and exploits the gathered knowledge to make suitable decisions for every state. The taken decision, also called action, is qualified by a reward or a penalty that the agent receives as a feedback from the environment. Therefore, instead of being taught by exemplars in datasets, the training data in RL is an evolving set of state-action pairs and their corresponding rewards (or penalties). The best sequences are used by the agent to have an optimal strategy or policy optimizing a cumulative reward. Thanks to its gradually refinement and dynamic adaptation, RL is suitable to problems that have no analytic formulation [40].

##### 3.2.2 Q-Learning

Q-learning algorithm is a form of model-free RL. It is a value-based method used to solve learning problems without knowledge on the environment dynamics. It is a mapping mechanism between a called state space  $S$  and action space  $A$  according to a policy. It aims at supplying information to an Agent about the most appropriate action to take in a given state in order to maximize a long term reward. At an instant  $t$ , a state-action value function  $Q(s_t, a_t)$  is used to measure the return of choosing the action  $a$  in the state  $s$ . The agent starts by a

**Table 1** Distributed SDN optimization approaches

Ref	SDN context	Approach	DRL based	Object
[7]	Datacenter Networks	Optimal Controller Number with dynamic Switch-to-controller mapping	No	Latency & Failure tolerance
[12]	WAN/Dynamic Topology	Scheduled inter-controller synchronization	Yes	Overall routing quality
[16]	WAN/ Fixed Topology	Dynamic leader-controller Placement	No	Latency
[26]	Vehicular/ Dynamic Topology	Static Controller Placement	No	Latency & Capacity
[27]	Vehicular/ Dynamic Topology	Dynamic Controller Placement	No	Latency & Capacity
[29]	WAN/Fixed Topology	Optimal Capacitated Controller Placement	No	Latency
[30]	WAN/ Fixed Topology	Optimal Controller Placement using K-means-based network partition	No	Latency
[31]	Backbone Networks	Optimal Controller Placement based on robustness factors	No	Latency & Reliability
[32]	Small/Large scale fixed topologies	Controller Placement Optimization using genetic algorithm (GA) and gradient descent (GD)	No	Latency
[33]	Small/medium scale fixed topologies	Controller Placement Optimization based on inter-controller delay minimization using MCDS	No	Latency
[34]	WAN/Dynamic Topology	Controller Synchronization Policy Design	Yes	Latency
IRCP-SDVN	Vehicular/ Dynamic Topology	Efficient Distributed Controller Number with Dynamic domain allocation	Yes	Latency & Reliability

random Q-function then it continuously updates its Q-Values through Equation (1) where  $\alpha \in [0, 1]$  is the learning rate and  $\gamma \in [0, 1]$  is the discount rate [41]. The final goal of Q-learning is to find the best policy  $\Pi$  making the agent select the suitable action that maximizes the reward  $r_t$  over time. This policy is improved every time a new Q-value is given. For every state  $s_t \in S$ , the selection of the best action  $a_t \in A$  is described by Equation (2). The mathematical function  $\operatorname{argmax}$  returns the action  $a_t \in A$  that achieves the highest value of  $Q(s_t, a_t)$  function. The policy becomes optimal once the maximum Q-value is determined through training.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot [r_{t+1} + \gamma \cdot \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (1)$$

$$a_t = \arg \max_{a \in A} Q(s_t, a) \quad (2)$$

### 3.2.3 Deep Reinforcement Learning

Based on table method to record state-action rewards, RL is limited to low-dimensional problems. It is not designed to large-scale and highly dynamic networks, such as vehicular networks, that have high dimensional state and action spaces. In fact, it is impractical to retain all the Q-values for all the state-action pairs. Deep learning (DL), gives RL the ability to scale up to such complex decision-making problems. Indeed, deep neural networks (DNN) can automatically extract low-dimensional features from high-dimensional data thanks to function approxima-



tion and representation learning capabilities. Therefore, the function  $Q(s, a)$  is approximated by a parameterized function  $Q(s, a, \theta)$  where  $\{\theta\}$  are weights of the Deep Q-network or DQN [42]. Figure 2 recapitalizes Deep Reinforcement Learning process.

### 3.2.4 The epsilon-greedy policy

In order to avoid the risk that the agent falls into sub-optimal patterns without exploring the whole action/reward space, the two concepts called exploration and exploitation should be used especially in the beginning of training to allow some randomness in the selection of the best action. Otherwise, the agent may not find the best policy. The general approach used to solve that problem is the  $\epsilon$ -greedy method. Every iteration, the agent selects a random action with probability  $\epsilon$  or an optimal action (the output of the DNN equation 2) with a probability of  $(1-\epsilon)$ . The epsilon value starts close to 1 and slowly decays to 0 during training. That is, after a large exploration of the state/action space, the agent ends with taking actions only from the deep neural network [42].

## 4 IRCP-SDVN Framework

### 4.1 Problem Description and Motivation

Network reactivity in distributed SDVN is affected by three classes of delay namely vehicle-RSU, RSU-Controller and Controller-Controller delays. Vehicle-RSU delay is mainly affected by environmental factors such as interference, signal strength and handover frequency depending of vehicle speed. RSU-Controller delay is inherent to the known Controller Placement problem. Controller-Controller latency includes inter-controller propagation and processing delay that depends on the processing capability and load of the controller. It has been optimized in several works treating the so called Controller Synchronization problem [34]. However, in large scale networks, inter-controllers latency dominates the other classes of latency [16]. The increase in the number of controllers can generate more cost and energy consumption. Despite reliability and processing latency improvement thanks to load balancing between multiple distributed controllers, extra delays are produced due to the rise of inter-controller synchronization transactions with every added controller. Using ODL distributed controller [43], we showed in [5] that it is inefficient to increase the number of controllers above a certain value ( 2 controllers in our scenario ), because of remarkable increase in end-to-end delay which is not suitable for safety application. In this work, we will focus on the estimation of the optimal number of controllers in SDVN case. Our optimization objective is to find a tradeoff between minimizing latency to enhance SDVN reactivity and maintaining acceptable reliability level to meet the variety of vehicular applications requirements.

### 4.2 Solution Context

To analyze the inter-controller synchronization delay in both SDO and MDO models previously described in section 2.1, we consider a distributed SDVN divided into  $N$  domains controlled by  $N$  controllers that are deployed in the cloud. We suppose that a vehicle triggers a communication in the range of one RSU that misses entry in its flow table matching received packets. The RSU sends a packet-in message to its master controller (step1 in figure 3 (a) and (b)). The rest of steps depends on the consistency model.

In SDO model, the master controller sends a Raft request to the leader controller (step 2 in figure 3 (a)). The latter floods replicas of the last updated view on all the domains controllers (step 3 in figure 3 (a)) and waits for acknowledgment from the majority of them (step 4 in figure 3 (a)). Once it receives at least  $(N/2 + 1)$  controllers' acknowledgement (4 in figure 3 (a)), the leader controller sends the most recent global view to master controller (step 5 in figure 3 (a)) thus a packet-out is sent to requesting RSU (step 6 in figure 3 (a)). In MDO model, the master controller processes the request locally and sends back a packet-out message to the RSU (step 2 in figure 3 (b)). In the mean time, it advertises the updates to all the other domain controllers in an asynchronous way (step 3 in figure 3 (b)). Thus, to have consistent network view on all the network controllers, a number of  $(N-1)$  transactions is needed for each event. In both cases, these numbers are multiplied by propagation and processing delays for each controller. Consequently, in both synchronization models, the number of deployed controllers has a clear impact on increasing the total synchronization delay raising by that the entire network latency.

In our study, we will consider ODL distributed controller. In fact, we have shown in [44] that ODL outperforms other studied controllers (ONOS, POX, Floodlight) in terms of end-to-end delay for safety vehicular applications. Moreover, as we mentioned earlier, ODL allows the deployment of multiple instances of controller thanks to clustering technique according to strong consistency model. Thus, we adopt the above SDO model which requires fewer transactions so less synchronization delay.

### 4.3 System Design

Our architecture is mainly divided into three layers: the data plane, the control plane and the Intelligence plane as it is described in figure 4. The intelligent DRL agent (upper layer) retrieve data (traffic state, rewards) from the data plane and estimates the optimal number of controllers, then communicates it to the control plane for controllers adjustment.

#### 1. Data plane

The data plane mainly consists of forwarding devices

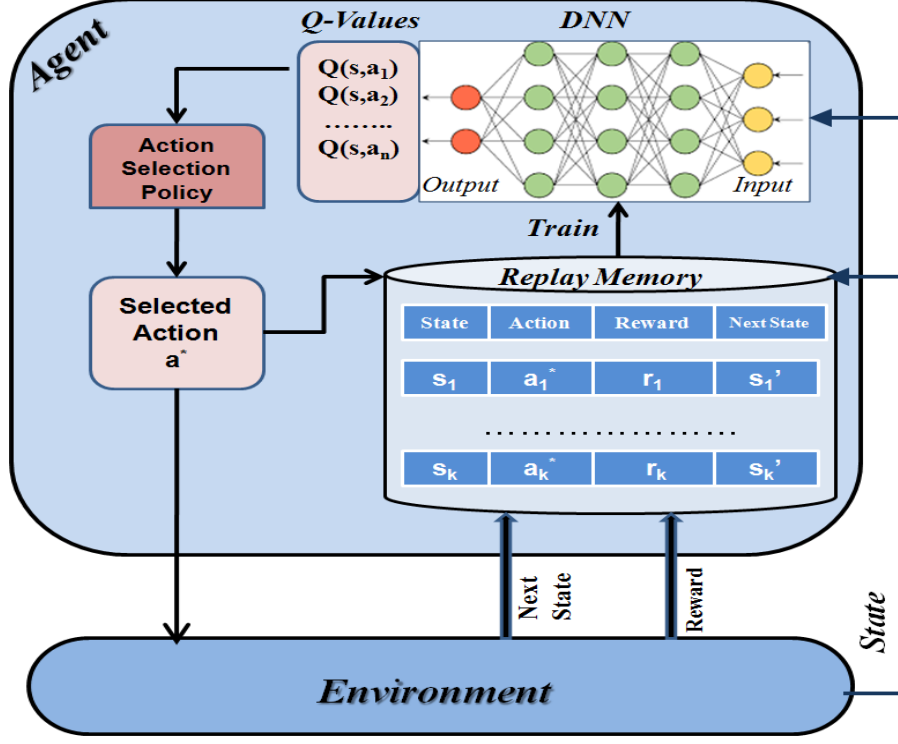


Fig. 2 Deep Reinforcement Learning process

such as OpenFlow-enabled RSUs (Road Side Units). These RSUs are grouped into different control domains in a dynamic way according to the distributed controllers allocation. It is also responsible for reporting the vehicles information and status such as velocity and density to DRL agent. Besides, feedback on executed actions is reported to the agent as a linear reward function composed of end-to-end (e2e) delay and packet loss level indication during vehicles transmissions.

2. Control plane: Adjustable number of distributed SDN controllers  
The role of this layer is to adjust the number of distributed controllers according to the DRL agent decision. It consists of two modules. The integrated **Control Resizer (CR)** which is responsible for adding or removing controllers to deploy the effective number (ENC) dictated by the DRL agent, and the **Load Balancer (LB)** which is charged to reassign the different RSUs to the deployed controllers to balance the traffic load.
3. Intelligence Plane: Intelligent Regulator  
The intelligence core is the DRL Agent that learns the policy through interaction with the vehicular environment. In other words, it collects and processes the state reported by the data plane, namely vehicles density and speed. Then, taking into account the feedback on achieved delay and packet loss, it transforms information to a policy that it improves during training

to make best decision about the optimal number of distributed controllers. In the following, we will detail the different modules of our architecture.

#### 4.3.1 Deep Reinforcement Learning Agent

In order to enable the DRL agent for control regulation, we have to define the states, actions, and reward function in the IRCP. All used notations are summarized in table 2.

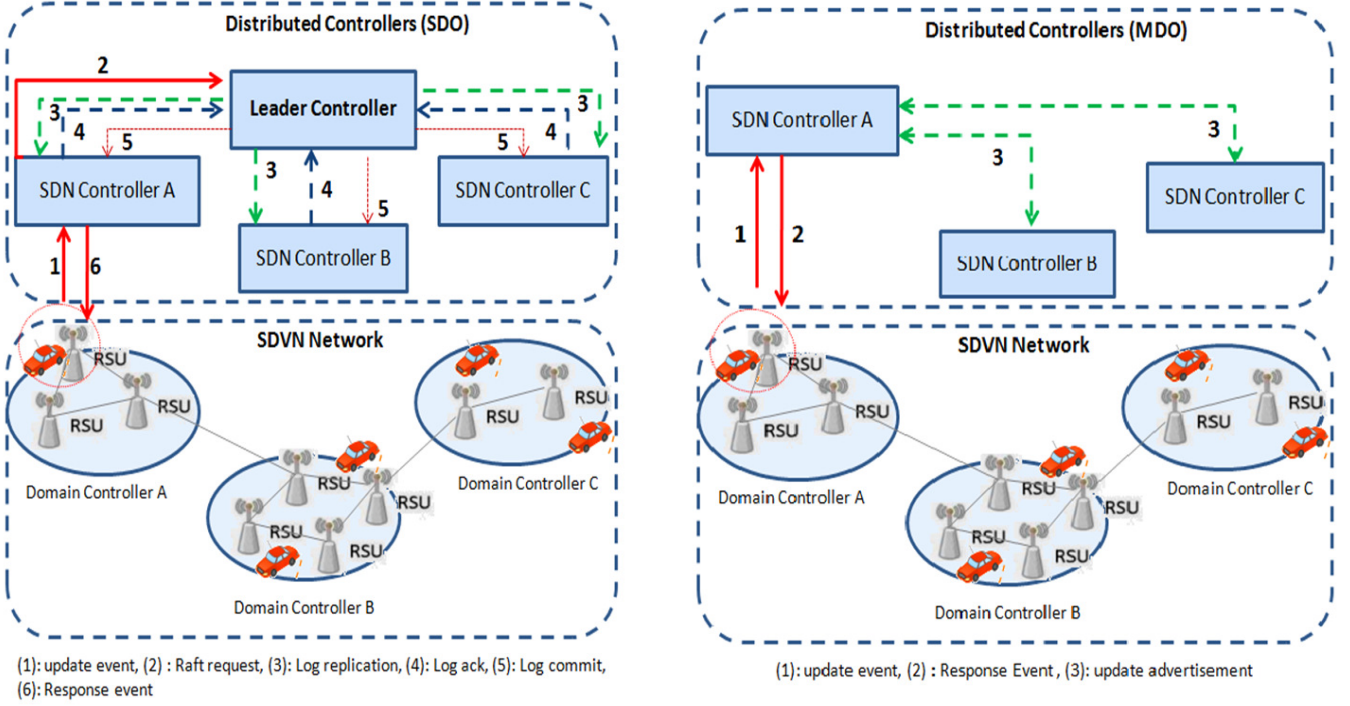
##### 1. IRCP State Space

We denote the state space by  $S$ . We opted that a state  $s_t \in S$  is represented by the tuple  $\{t, \rho, v\}$ . The variables  $t, \rho$  and  $v$  refer respectively to the slot time, the density of the vehicular network and the vehicles' velocity at the instant  $t$ . We are considering only communicating vehicles that are generating traffic in our SDVN network.

##### 2. IRCP Action space

In our system, at each time  $t$  and based on a decision policy, the agent takes an action  $a_t \in A$ , which consists of deciding the effective number of distributed controllers denoted by ENC, according to a current state  $s_t \in S$ . The action space is  $A = \{1C, 2C, 3C, 4C \dots nC\}$  where the number  $n$  depends on the network scale and the allocated controller performances.

##### 3. IRCP Reward function



**Fig. 3** Consistency models in distributed SDVN. (Left) Fig. 3(a) SDO model (Raft algorithm); (Right) Fig. 3(b) MDO model (anti-entropy algorithm)

The agent relies on the feedback received on the reward  $r_t$  to evaluate the effectiveness of the action in order to improve the strategy. We define the reward function using two most important QoS indicators in vehicular networks: average end-to-end delay (d) and packet loss (p) that reflect respectively the responsiveness and reliability of the vehicular network under the control of the decided ENC controllers. End-to-end delay is measured in milliseconds and packet loss is the percentage of sent but not acknowledged packets. Reward  $r_t$  is denoted by Equation 3. In each state the received values of instant end-to-end delay and packet loss are compared to specified thresholds respectively  $D_{th}$  and  $P_{th}$ . Thus, penalties are incurred if latency and loss values exceed the defined thresholds. Otherwise, reward is incremented. These thresholds are defined depending on the vehicular application class. For example, safety applications are strictly delay-sensitive while infotainment applications do not tolerate packet loss.

$$r_t = r_d + r_p \quad (3)$$

Where  $r_d$  is delay reward and  $r_p$  is packet loss reward. They are calculated according to following process:

- $r_t$ ,  $r_d$  and  $r_p$  are initialized to zero
- Get the instant end-to-end delay (d) and packet loss (p) values in instant (t)
- If ( $d < D_{th}$ ) then  $r_d ++$ ; else  $r_d --$ ;
- If ( $p < P_{th}$ ) then  $r_p ++$ ; else  $r_p --$ ;

**Table 2** Summary of key notations

Notation	Description
ENC	Effective Number of Controllers
CNC	Current Number of Controllers
$\rho$	Vehicular density
$v$	Velocity
d	End to end delay
p	Packet loss
$D_{th}$	End to end delay threshold
$P_{th}$	Packet loss threshold
$r_t / r_d / r_p$	Rewards

#### 4.3.2 Deep Q-learning Based IRCP

We draw up in figure 5 a flow chart describing the steps followed by our system to obtain the optimal policy. Three main modules compose our system: the DRL module, the vehicular network (vehicles and RSUs), and the distributed controllers. The policy is initialized randomly as well as the DNN parameters and the CNC. Once simulation environment and CNC distributed controllers are started, vehicles and RSUs are generated and wireless V2I (Vehicle to Infrastructure) links are established to allow traffic exchange. Necessary data for agent learning is therefore available.

As the agent has no sufficient experience in the beginning to select the best action, the  $\epsilon$ -greedy policy is used to equilibrate the exploration and exploitation [45]. The  $\epsilon$  value decay is repeated at every step until the

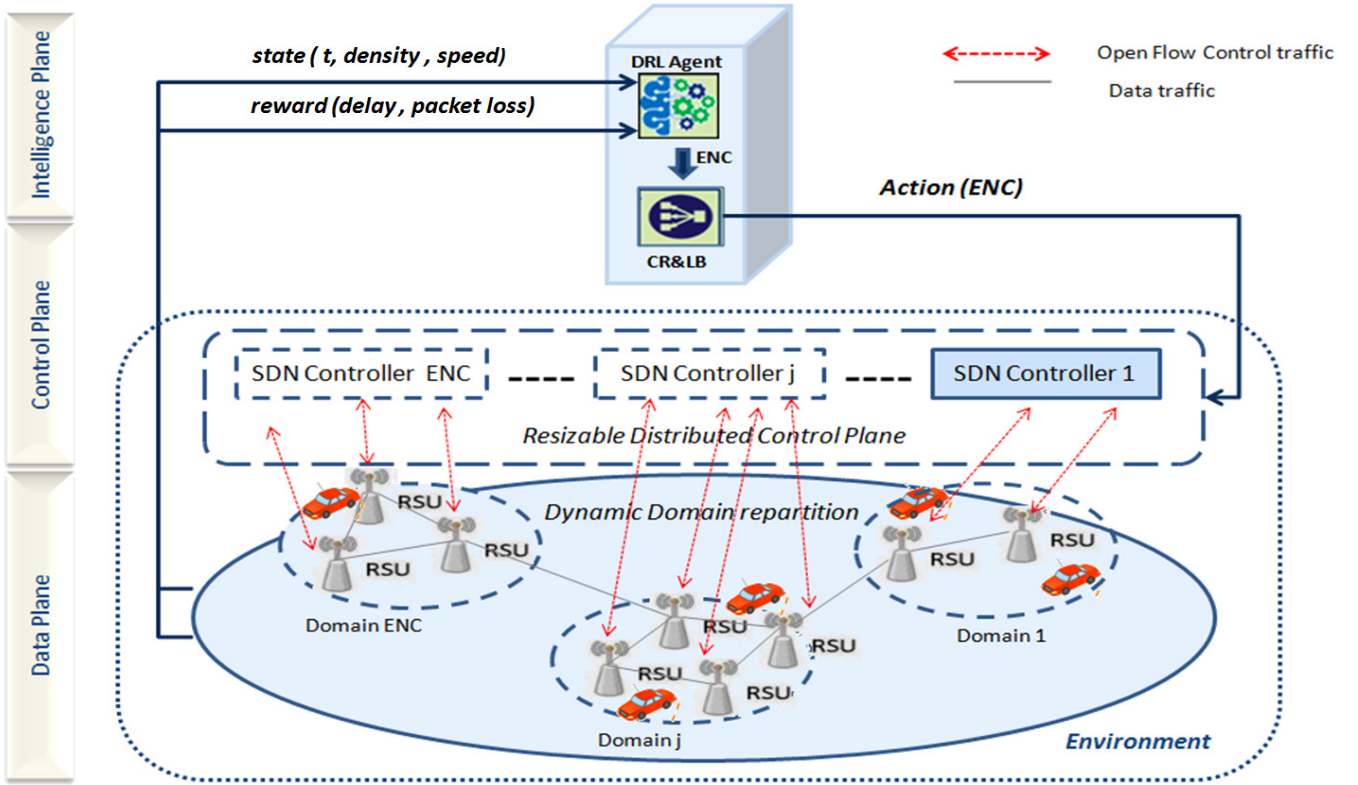


Fig. 4 IRCP-SDVN Architecture

choice of action becomes completely made by the DNN as explained previously in section 3.2.4. Thereafter, batching and training take place. As explained in paragraph 3.2.3 the parameterized Q-function is represented by a Deep Neural Network (DNN) that needs to be trained on randomly sampled batches from the agents replay memory [45] for stability reasons. Training data is generated from the interactions between the environment simulator, controlled by the distributed controllers, and the agent that decides the efficient number of those controllers. The former returns rewards for the selected actions by the latter. The memory is filled by samples including agents data namely its original state  $s_t$ , the chosen action  $a_t$ , the received reward  $r_t$  for that action and its next state  $s_{t+1}$ . Initially, the policy used to select the Efficient Number of Controllers (ENC) is random. It is gradually improved by updating the Q-network (adjusting weights and biases). At the end of the training process the returned Deep Q-Network will be ready to estimate the best action for each state.

#### 4.3.3 IRCP Components Interaction

The execution of the DRL agent's orders is the responsibility of the two modules CR and LB. The CR is responsible of adding (start) new controllers if ENC is greater than CNC and removing (shut down) extra controllers if ENC is smaller than CNC. CR stills idle when the two numbers

are equal. The LB module is in charge of RSUs reassignment to deployed controllers (ENC) in an almost equal manner. This simple method leads to a rapid mapping between RSUs and controllers reducing time of migration and re-association procedures. Also, it is an event-based method as LB is triggered only when ENC is different of CNC. Such a suitable method of load balancing improves the scalability, responsiveness and reliability. It helps minimizing resource consumption and averts the overload of any of the distributed controllers [46]. Figure 6 resumes the interactions between our architecture components.

## 5 Evaluation and results

The purpose of this section is to prove the feasibility and efficiency of our proposed IRCP-SDVN. We opted to Road Safety Application use case in a realistic urban mobility trace. It is the most delay-sensitive class of vehicular applications. We recall that our aim is to improve QoS in distributed SDVN taking into consideration the impact of vehicles mobility and traffic density. Indeed, when density of communicating vehicles increases, it may lead to controller overload. Also, high mobility results in increasing handover from one RSU to another and if these RSUs are belonging to different control domains, more inter-controller synchronizations are triggered.

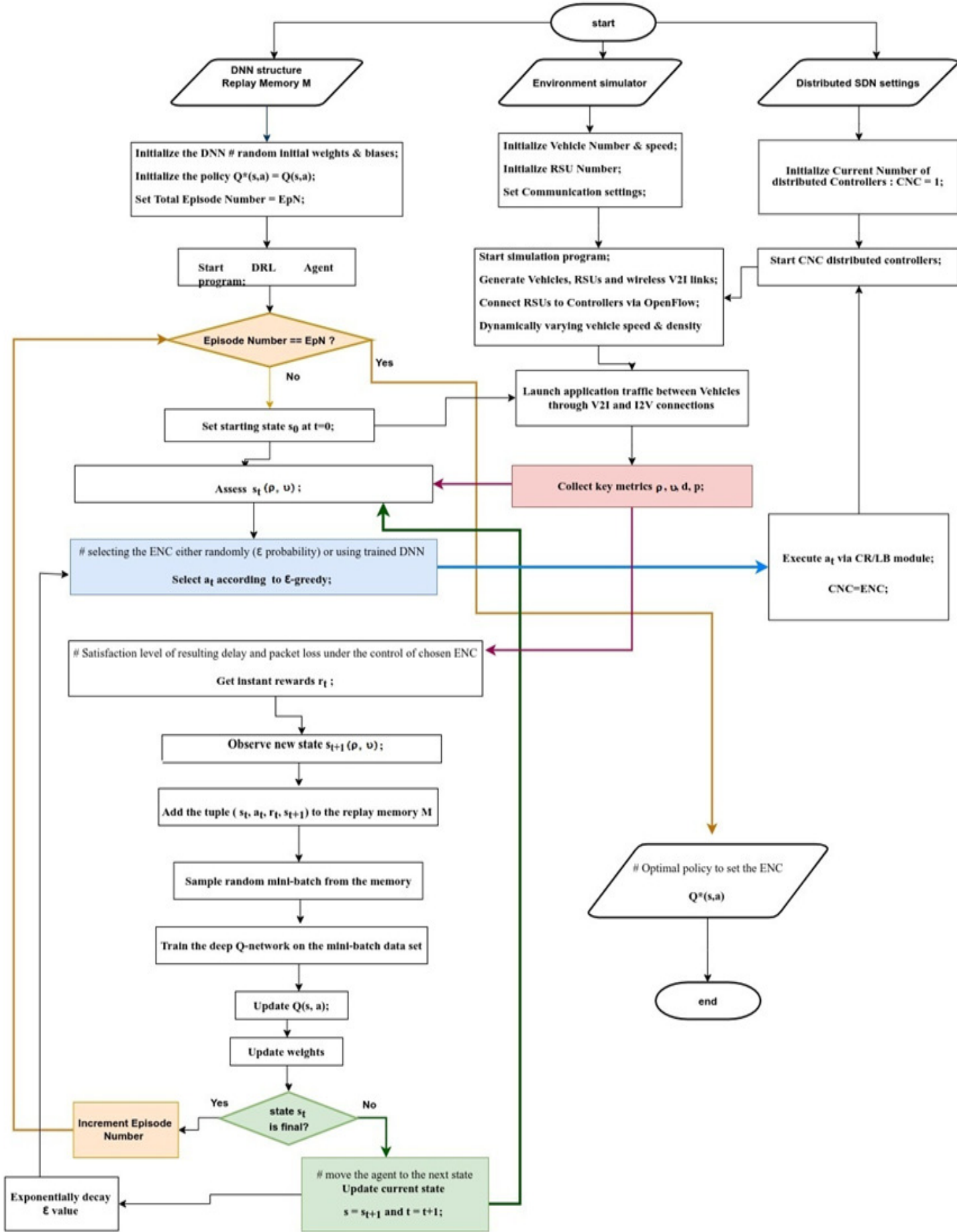
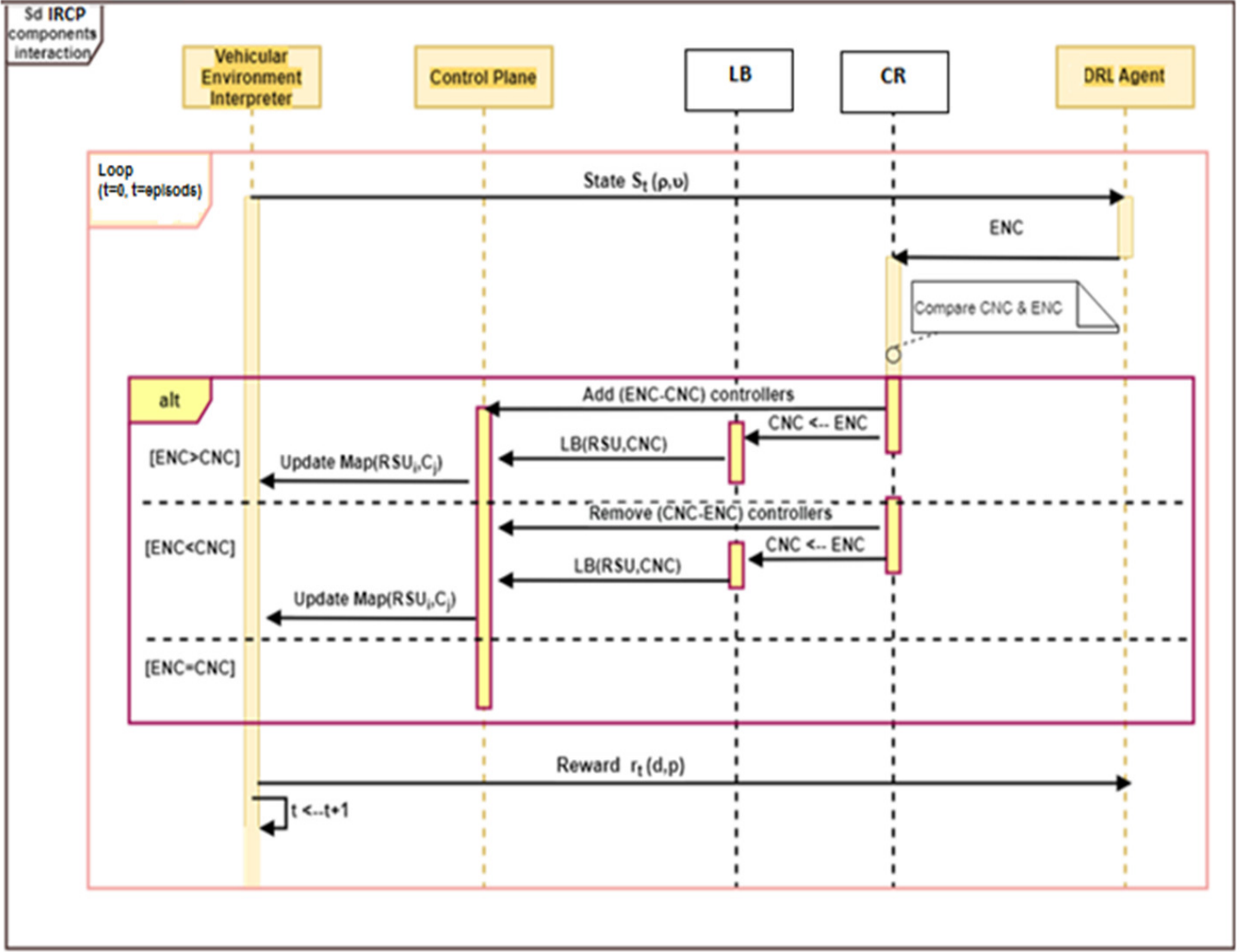


Fig. 5 IRCP operation scheme





**Fig. 6** Sequence diagram of IRCP components interactions

### 5.1 Experimental settings

Our simulation is implemented using Ubuntu 18.0.4 operating system running on an Intel Core i7 CPU with 32GB of RAM. We emulate SDN based vehicular network with Mininet-wifi [47]. We downscaled a 2000m X 2000m area of San Francisco from the Open Street Map (OSM) for our road topology simulated with SUMO simulator [48]. Regarding emergency traffic generation and measurements of end-to-end delay and packet loss, we use iperf 2.0.13 [49]. We opt for ODL as SDN controller as we previously proved its efficiency in such environment and scenario [44]. We run ODL controllers on Docker containers [50]. For packet routing, we use the layer-2 forwarding application of ODL called l2-switch and deploy it on our OpenFlow enabled RSUs. This application affords the default reactive learning/forwarding mechanism. Inter-Controller synchronization is ensured by the clustering technique of ODL [43]. For IRCP agent training we use NVIDIA Tesla k80 GPU with 12 GB of RAM offered by the free GPU provider Google Colab, and the following

software environments: TensorFlow 1.15.0 and keras 2.3.1 with python 2.7.17. Regarding our deep Q-network, training convergence leads to a stable five-layer fully connected neural network with validation mean square error (MSE) equal to 0.02. The 3 hidden layers consist respectively of 100, 80 and 50 neurons. We opt for Relu as activation function and adaptive moment estimation method (Adam) for training with a learning rate of 0.01. Our replay memory has a maximum size of 10.000 samples with batch size of only 20 samples as the smaller batch size is, the better generalization will be. To balance exploration and exploitation we utilized  $\epsilon$ -greedy policy. The two thresholds  $D_{th}$  and  $P_{th}$  are respectively fixed to 100ms and 15% according to ETSI specifications for vehicular services requirements [51]. Figure 7 sketches our Test-bed.

### 5.2 Experimented Scenario

We consider an accident scenario where a car crash occurs in one RSUs coverage, and a neighboring ambulance or an informed authority car will disseminate periodically



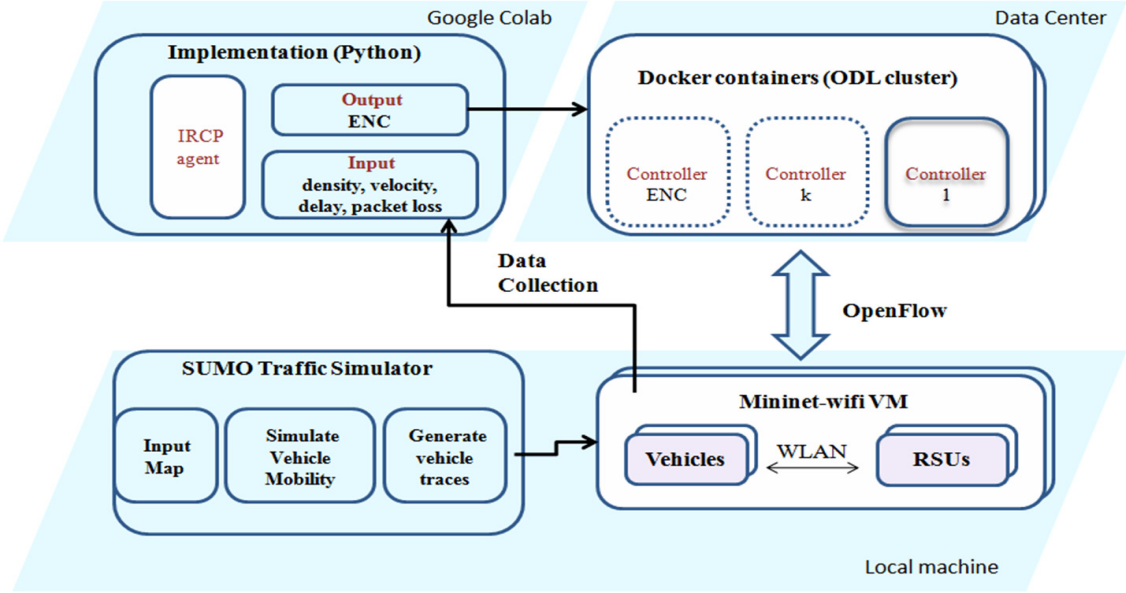


Fig. 7 Test-bed

Table 3 Simulation Setup

Parameters	Value
Vehicles density	10 to 90 by step 10
Vehicles speed (kmph)	10 to 90 by range 10
RSU range	600m
RSU number	8
Bandwidth	54Mb/s (802.11g)
OpenFlow version	1.3
Simulation area	2000m x 2000m
Simulation time	20 min
Propagation Model	Log-Distance-Loss Model (exp=2.5)
Traffic type	User Datagram Protocol (UDP)
Packet Size	1470 bytes
Periodicity	3s

the warning (Decentralized Environmental Notification Message or DENM) to the rest of vehicles in the relevance area as sketched in figure 8. We mention that for readability sake, we have represented the control and intelligence plane with a single IRCP entity. Such safety related messages are strictly delay sensitive and require high reliability [52]. We consider two cases. In the first one, we fix the speed and gradually vary the communicating vehicles number. In the second we do the contrary as we fix communicating vehicles number while varying gradually the velocity (random values) by range of 10 km between minimum and maximum speeds. Simulation parameters are summarized in table 3.

To prove the performance of our proposed IRCP-SDVN architecture, we will compare it to a centralized or single controller SDVN architecture as well to a distributed SDVN architecture with fixed number of controllers for both described scenarios. The first reference architecture is used to show the importance of distributing the control plane in avoiding single controller overload and consequent

degradation of QoS. The second reference architecture represents the oversized distributed control plane for the same density of our vehicular network. The goal of IRCP is to resize the distributed control plane according to the traffic load.

Our tests were limited to four controllers and a maximum number of communicating vehicles to 90 given the constraints of resources available to us. In both scenarios, we used the same settings and traffic scheme where vehicles establish connections only with RSUs for exchanging data packets (V2I communications). Thus, a vehicle belongs to a controller domain only when it is in the range of one or more RSUs of its domain. We precise that in the following measurements, V2V (Vehicle to Vehicle) links are disabled on purpose.

### 5.3 Results and analysis

We compare the performance of the 3 architectures in terms of end-to-end delay and packet loss. In the case of IRCP-SDVN, we consider first one controller then we increase this number if needed according to the network load. Our experimental results are sketched in figures 9 to 14. Each represented value is averaged over 5 runs. Picked end-to-end delay and packet loss values are the average of all achieved values among the communicating vehicles during the simulation during each episode.

The first part of our experiments is to study the performance of all architectures in terms of end to end delay and packet loss by varying the density of vehicles. In figure 9, we can perceive that end to end delay increases when vehicular density rises. In fact, the SDN controller has to serve more and more communicating vehicles and to forward further control messages. We note that the case

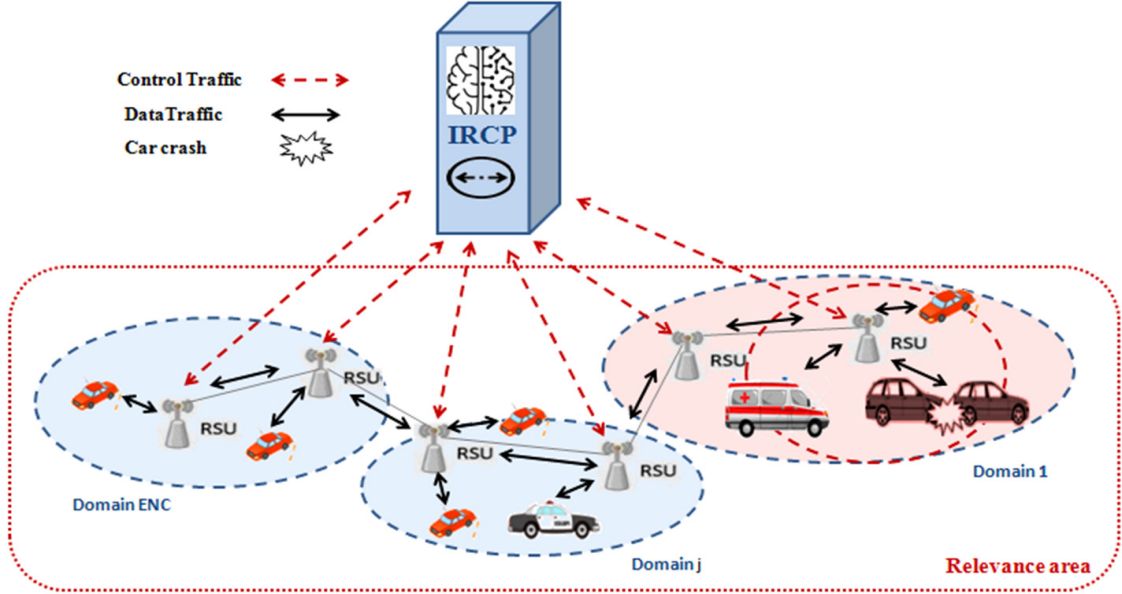


Fig. 8 Emergency scenario

of single controller presents the most important delays due to its rapid overload when density reaches its peak. Despite the better delays presented by the distributed controller (4 controllers) compared to a single controller, it notes lower performance than IRCP when density is high. This difference is due to the synchronization overload of the four controllers compared to IRCP which uses only two controllers when the vehicles density increases (figure10). In fact, each update needs to be synchronized among different domain controllers which results in additional delays. This is clearly shown in figure9 where IRCP performs significantly lower delay where density exceeds 60 vehicles. Indeed, the sudden fall in delay is the result of IRCP agent intervention when it decides to add a second controller around the sixth minute as described in figure10.

As a consequence, the vehicles requests are balanced between two controllers and the delay is remarkably reduced. From the load of seventy vehicles, end-to-end delay returns to rise as the density increases. Around the ninth minute, density reaches its peak but IRCP performs well as end-to-end delay stills under the threshold (100 ms) and better than the case of 4 controllers (figure9) . Furthermore, by decreasing the vehicles density, the end-to-end delay decreases and IRCP removes the extra controller and returns to work with only one controller like in the beginning of the simulation period (figure 10). Thus, IRCP adapts dynamically to the load of the vehicular network by adopting an elastic control.

Figure11 presents the packet loss in function of Vehicular density. For the same aforementioned reasons, density increase also leads at certain level to controller overload which results in packet loss rise. This is well shown with the centralized architecture that has the highest loss rate. Using four controllers, packets have more

chance to be treated and packet loss is less. However, we remark that IRCP performs worse than the four-controller distributed architecture even while adding the second controller (around the 6<sup>th</sup> minute) which is expected since two controllers are more charged than four. But IRCP performs better than the case of single controller. In the following experiments, we investigate the performance of the three architectures by varying the speed of vehicles. Figure12 illustrates the end-to-end delay with different vehicle velocities. As we notice, delay escalates with the speed. First, this is due to the rapid change of vehicular network topology so controller has to update its network view frequently. Centralized architecture performs the worst delays in this case as it has to manage more topology changes so more delays to respond to vehicles requests or exactly more treatments to install new flows on RSUs. In fact, as it groups together all the RSUs in a single domain, each vehicle in the network is liable to modify the overall view. In distributed architectures, controllers have to synchronize their views with each other after every update.

The high speed of the vehicles can also increase the delay due to broken links and their recovery time. The link break may occur when control domain is changed, which triggers global view update via inter-controller synchronization. That explains the high values of end-to-end delay produced by the static four-controller distributed architecture when velocity exceeds 40 km/h (figure12). However, our proposed IRCP performs clearly better as it intervenes in these conditions to add the second controller, as shown in figure13, thus decreasing the delay by processing the vehicles requests more quickly while limiting the controllers synchronization. Moreover, we remark that despite the increase of the end-to-end delay for a velocity greater than 60 km/s in the case of IRCP (figure12), it

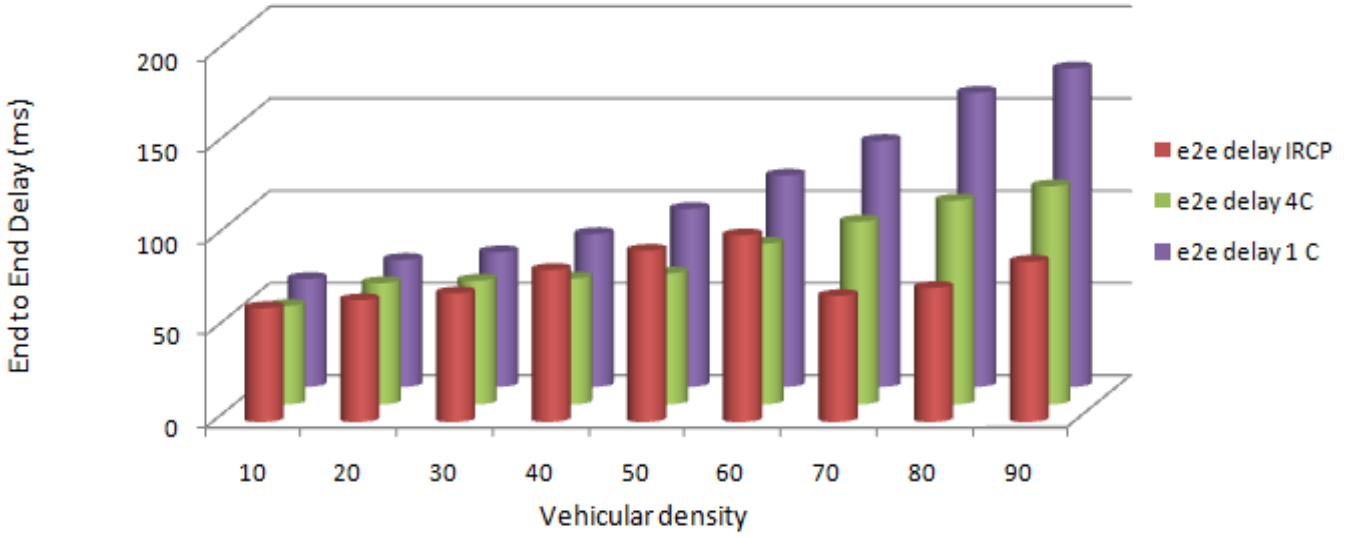


Fig. 9 E2E Delay with variable density

stills better than the case of 4 controllers (figure12). In addition, we notice that the delay decreases with the decrease in speed in the case of IRCP while adjusting the number of controllers to one (figure13).

These good results of IRCP in terms of responsiveness are supported by better reliability shown in figure 14. High vehicle speed means frequent migration of vehicles from one RSUs range or control domain to another which leads to high number of broken links that increases packet loss. By adding the second controller at speed exceeding 50 km/h, IRCP lightens the packet loss rate by balancing the load while keeping a reasonable number of distributed controllers (two) leading to less connection break than in case of four control domains. However the packet loss still

increasing with the increase of the velocity but still better than the case of one controller and 4 controllers. With a single controller, increasing speed results in frequent topology changes due to frequent handoffs from one RSUs range to another, hence the growth in packet loss (figure 14).

Overall, the results of our experiments have shown the interest of distributing the control plane compared to centralized architectures but with a reasonable deployment in terms of the number of distributed controllers. Our proposed IRCP-SDVN outperforms static four-controller distributed SDVN architecture especially in terms of delay, the key QoS metric for road safety applications. Even though results in terms of packet loss were higher than

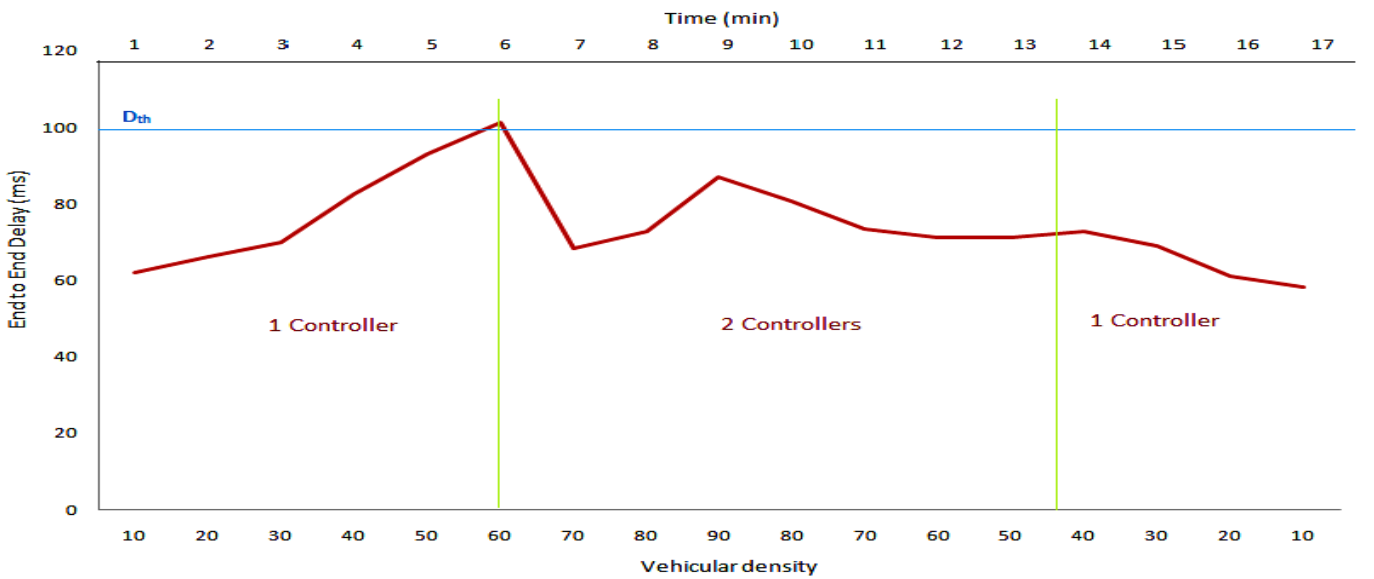


Fig. 10 IRCP behavior (End to End delay in function of variable density)

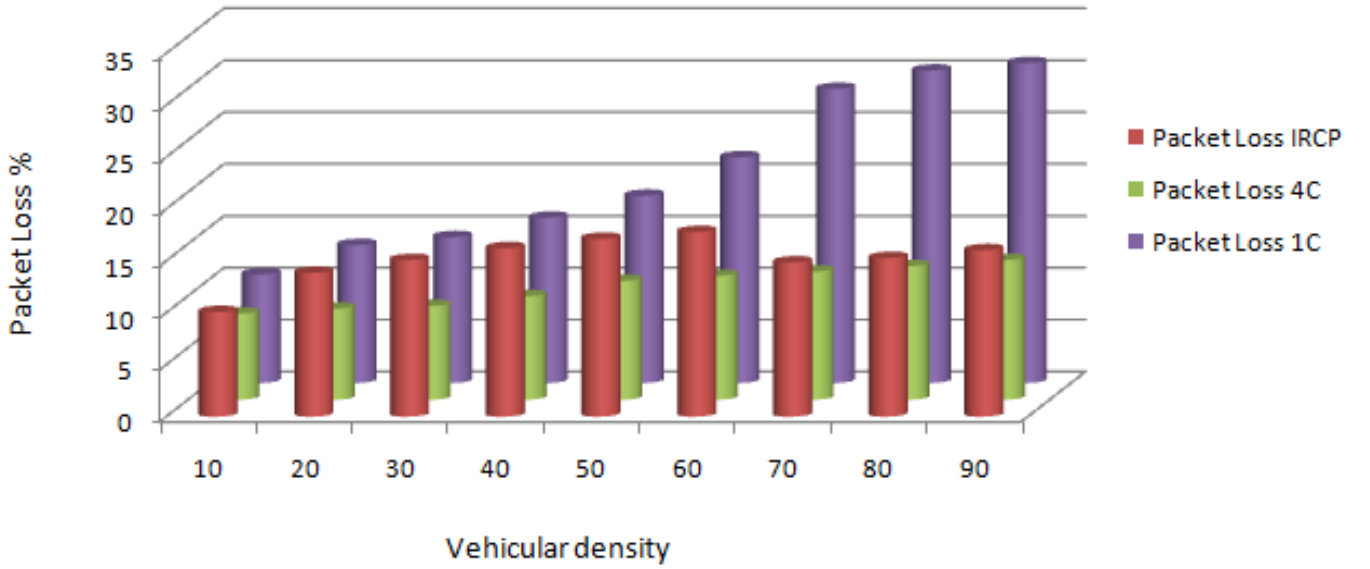


Fig. 11 Packet Loss with variable density

that achieved by the static four-controller distributed architecture during variable density experiments. These results can be improved by adjusting the reward function taking into account the end to end delay and packet loss parameters. We recall that the goal of IRCP is to find a compromise between responsiveness and reliability according to the requirements of the specified vehicular application.

## 6 Conclusion

It is clear that SDN's distributed control plane has shown its efficiency compared to the centralized controller, how-

ever over-dimensioning the number of controllers can cause additional delays for synchronization and can lose efficiency due to unnecessary overheads. Optimal choice of the number of controllers is necessary. In this paper, a deep reinforcement learning based approach called IRCP-SDVN is proposed to resize and adjust the distributed control plane in SDN-based vehicular networks according to the traffic load and network density with respect to vehicular applications requirements in terms of responsiveness and reliability. Experimental results prove that our proposed system can achieve better results in terms of end-to-end delay and packet loss for critical vehicular application such as road safety services. Emergency messages are delivered with significantly lower end-to-end delay and

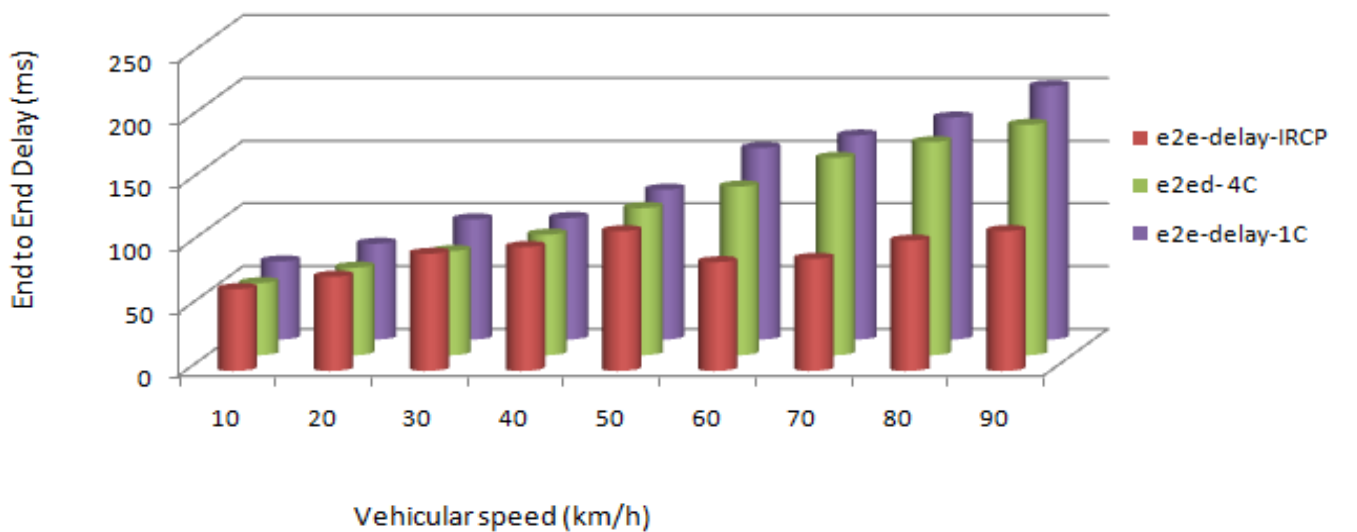


Fig. 12 E2E Delay with variable vehicle speed

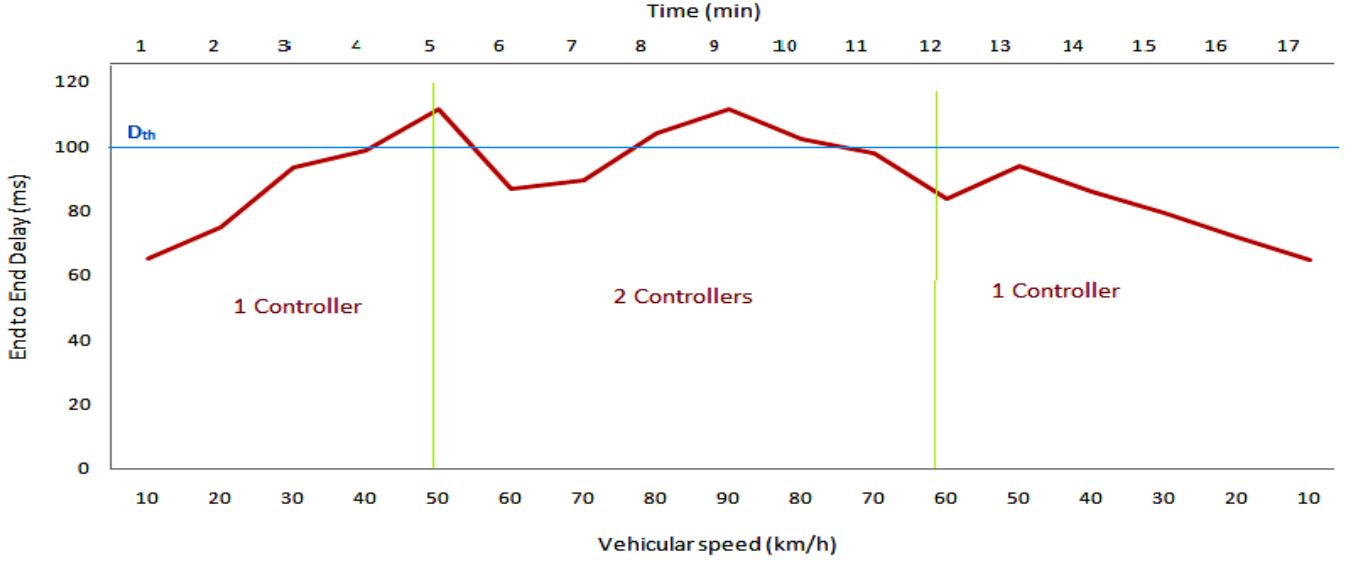


Fig. 13 IRCP behavior (End to End delay in function of variable speed)

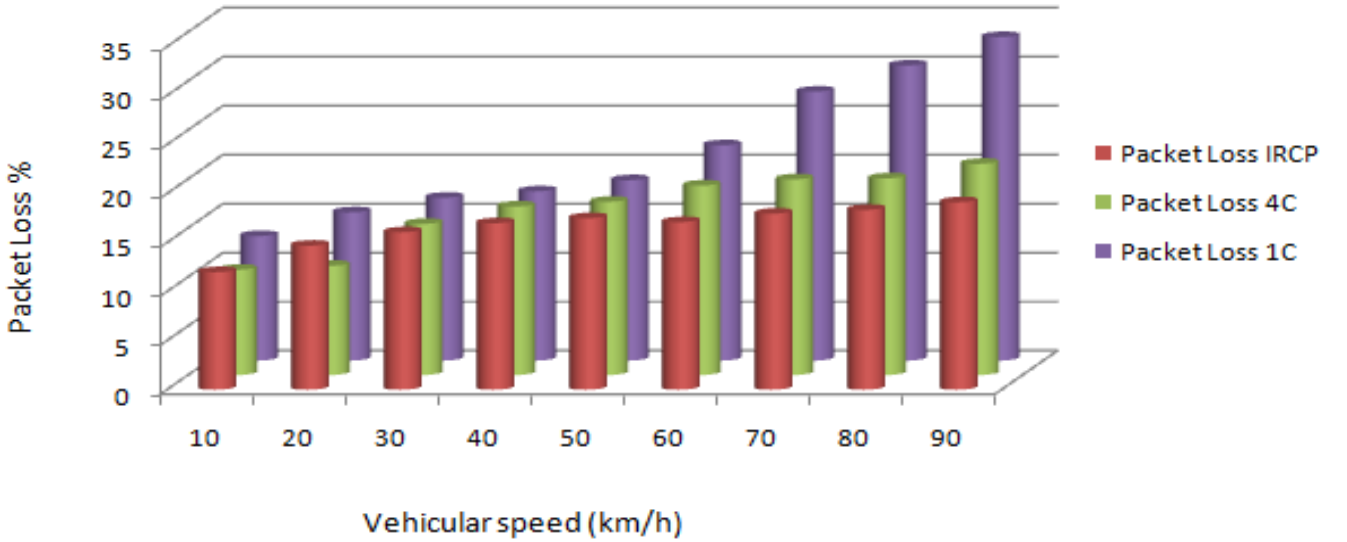


Fig. 14 Packet Loss with variable vehicle speed

alleviated packet loss comparing to statically distributed SDVN architectures under various dynamicity degrees. Indeed, thanks to DRL based IRCP, efficient number of controllers is deployed avoiding generated extra-delays of inter-controller synchronization and extra consumed resources for extra controllers. More experiments should be performed in order to study more precisely the contribution of optimal controller number choice on the QoS requirements of other vehicular applications such as infotainment. In addition, it is important to assess the DRL agent overload in the case of a denser and larger network even though we believe that the DRL is executed by a much more powerful controller than the others domains controllers. So, the cost of processing will be much lower

than that of the frequent processing of the various transactions executed by the domain controllers. Encouraged by the obtained results, we intend as a perspective to challenge the proposed framework in a more intensive scenario and compare it to other numerical or ML methods to estimate the optimal number of active controllers. In addition, we plan to compare our proposal of the elastic control plan to existing works. It would also be interesting to investigate the possible combination between our approach and the capacity scaling of the controllers as proposed by Beacon and McNettle projects.

## Conflicts of interest/Competing interests

The authors declare that there are no conflicts of interest and competing interests

## References

1. Bannour, F., Souihi, S., & Mellouk, A. (2018). Distributed SDN Control: Survey, Taxonomy, and Challenges. *IEEE Commun. Surv. Tutor*, 20, 1, 333354. doi: 10.1109/COMST.2017.2782482.
2. Fan, Y., & Zhang, N. (2017). A Survey on Software-defined Vehicular Networks. *Journal of Computers*, 28, 4, 236-244. doi:10.3966/199115592017062803025.
3. Smida, K., Tounsi, H., Frikha, M., & Song, Y. (2019). Software Defined Internet of Vehicles: a survey from QoS and scalability perspectives. In 15th International Wireless Communications Mobile Computing Conference (IWCMC) (pp. 13491354). doi: 10.1109/IWCMC.2019.8766647.
4. Kaul, A., Obraczka, K., Santos, M. A. S., Rothenberg, C. E., & Turletti, T. (2017). Dynamically distributed network control for message dissemination in ITS. In *IEEE/ACM 21st International Symposium on Distributed Simulation and Real Time Applications (DS-RT)* (pp. 19). doi:10.1109/DISTRA.2017.8167677.
5. Smida, K., Tounsi, H., Frikha, M., & Song, Y. (2019). Delay Study in Multi-controller Software Defined Vehicular Network Using OpenDaylight for Emergency Applications. In 15th International Wireless Communications Mobile Computing Conference (IWCMC) (pp. 615620). doi: 10.1109/IWCMC.2019.8766633.
6. Kumari, A., & Sairam, A. S. (2021). Controller placement problem in software-defined networking: A survey. *Networks*. doi: 10.1002/net.22016.
7. Liu, W., Wang, Y., Zhang, J., Liao, H., Liang, Z., & Liu, X. (2020). AAMcon: an adaptively distributed SDN controller in data center networks. *Front. Comput. Sci.*, 14, 1, 146161. doi: 10.1007/s11704-019-7266-6.
8. Dixit, A. A., Hao, F., Mukherjee, S., Lakshman, T. V., & Kompella, R. (2014). *ElastiCon: an elastic distributed SDN controller*. Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems - ANCS 14, Los Angeles, California, USA (pp. 1728). doi: 10.1145/2658260.2658261.
9. Boutaba, R., & al. (2018). A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *J. Internet Serv. Appl.*, 9, 1, 16. doi: 10.1186/s13174-018-0087-2.
10. Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Process. Mag.*, 34, 6, 2638. doi: 10.1109/MSP.2017.2743240.
11. Ye, H., & Li, G. Y. (2018). Deep Reinforcement Learning for Resource Allocation in V2V Communications. In *IEEE International Conference on Communications (ICC)* (pp. 16). doi: 10.1109/ICC.2018.8422586.
12. Zhang, Z., Ma, L., Poularakis, K., Leung, K., & Wu, L. (2019). DQ Scheduler: Deep Reinforcement Learning Based Controller Synchronization in Distributed SDN. *ArXiv: 1812.00852*, 1-7. doi: 10.1109/ICC.2019.8761183.
13. Open Networking Foundation. Software-Defined Networking (SDN) Definition. Retrieved mai 10, 2020, from <https://www.opennetworking.org/sdn-definition/>
14. Open Networking Foundation. OpenFlow protocol Archives. Retrieved may 09, 2020, from <https://www.opennetworking.org/tag/openflow-protocol/>
15. IBM Cloud Education. (2021). REST APIs, Retrieved august 02, 2021, from <https://www.ibm.com/cloud/learn/rest-apis>
16. Zhang, T., Bianco, A., & Giaccone, P. (2016). The role of inter-controller traffic in SDN controllers placement, In *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)* (pp. 8792). doi: 10.1109/NFV-SDN.2016.7919481.
17. Brewer, E. (2012). Pushing the CAP: Strategies for Consistency and Availability. *Computers*, 2, 45, 2329. doi: 10.1109/MC.2012.37.
18. OpenDaylight. Retrieved jul 26, 2020, from <https://www.opendaylight.org/>
19. Open Network Operating System (ONOS) SDN Controller for SDN/NFV Solutions, Open Networking Foundation. Retrieved jul 26, 2020, from <https://www.opennetworking.org/onos/>
20. Ongaro, D., & Ousterhout, J. (2014). In Search of an Understandable Consensus Algorithm. In *Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference* (pp. 305-320).
21. Sakic, E., & Kellerer, W. (2018). Response Time and Availability Study of RAFT Consensus in Distributed SDN Control Plane. *IEEE Trans. Netw. Serv. Manag.*, 15, 1, 304318. doi: 10.1109/TNSM.2017.2775061.
22. Network Topology State - ONOS - Wiki. Retrieved jul 29, 2020, from <https://wiki.onosproject.org/display/ONOS/>
23. Jiacheng, C., Haibo, Z., Ning, Z., Peng, Y., Lin, G., & Xuemin, S. (2016). Software defined Internet of vehicles: architecture, challenges and solutions, *J. Commun. Inf. Netw.*, 1, 1426. doi: 10.1007/BF03391543.
24. Erickson, D. (2013). The Beacon OpenFlow Controller. (p. 18). doi: 10.1145/2491185.2491189.
25. Voellmy, A., & Wang, J. (2012). Scalable Software Defined Network Controllers, *ACM SIGCOMM Comput. Commun. Rev.*, 42, 289290. doi: 10.1145/2377677.2377735.
26. Kalupahana Liyanage, K. S., Ma, M., & Joo Chong, P. H. (2018). Controller placement optimization in hierarchical distributed software defined vehicular networks. *Comput. Netw.*, 135, 226239. doi: 10.1016/j.comnet.2018.02.022.
27. Toufega, S., Abdellatif, S., Assouane, H. T., Owezarski, P., & Villemur, T. (2020). Towards Dynamic Controller Placement in Software Defined Vehicular Networks. *Sensors*, 20, 6, 1701. doi: 10.3390/s20061701.
28. An Overview of USDOT Connected Vehicle Roadside Unit Research Activities. (2017). Retrieved jul 28, 2020, from <https://connectedautomateddriving.eu/wp-content/uploads/2017/08/USDOT.pdf>
29. Chai, R., Yuan, Q., Zhu, L., & Chen, Q. (2019). Control plane delay minimization-based capacitated controller placement algorithm for SDN, *EURASIP. J. Wirel. Commun. Netw.*, 1, 282. doi: 10.1186/s13638-019-1607-x.
30. Wang, G., Zhao, Y., Huang, J., Duan, Q., & Li, J. (2016). A K-means-based network partition algorithm for controller placement in software defined network. In *IEEE International Conference on Communications (ICC)*, (pp. 16). doi: 10.1109/ICC.2016.7511441.
31. Alenazi, M. (2019). Distributed SDN Deployment in Backbone Networks for Low-Delay and High-Reliability Applications. *Int. J. Adv. Comput. Sci. Appl.*, 10. doi:10.14569/IJACSA.2019.0101274.
32. Huang, V., Chen, G., Fu, Q., & Wen, E. (2019). Optimizing Controller Placement for Software-Defined Networks. In *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, (pp. 224232).
33. Alowa, A., & Fevens, T. (2020). Towards Minimum Inter-Controller Delay Time in Software Defined Networking. *Procedia Comput. Sci.*, 175, 395402. doi: 10.1016/j.procs.2020.07.056.
34. Zhang, Z., Ma, L., Poularakis, K., Leung, K. K., Tucker, J., & Swami, A. (2019). MACS: Deep Reinforcement Learning based SDN Controller Synchronization Policy Design. *ArXiv190909063 Cs*. Retrieved jul 17, 2020, from <http://arxiv.org/abs/1909.09063>.



35. Xie, J., & al. (2019). A Survey of Machine Learning Techniques Applied to Software Defined Networking (SDN): Research Issues and Challenges. *IEEE Commun. Surv. Tutor.*, 21, 1, 393430. doi: 10.1109/COMST.2018.2866942.
36. Mestres, A., & al. (2017). Knowledge-Defined Networking. *ACM SIGCOMM Comput. Commun. Rev.*, 47, 3, 210. doi: 10.1145/3138808.3138810.
37. Yu, C., Lan, J., Guo, Z., & Hu, Y. (2018). DROM: Optimizing the Routing in Software-Defined Networks with Deep Reinforcement Learning. *IEEE Access*, PP, 11. doi: 10.1109/ACCESS.2018.2877686.
38. Guo, X., Lin, H., Li, Z., & Peng, M. (2019). Deep Reinforcement Learning based QoS-aware Secure Routing for SDN-IoT. *IEEE Internet Things J.*, 11. doi: 10.1109/JIOT.2019.2960033.
39. Zhang, D., Yu, F. R., Yang, R., & Tang, H. (2018). A Deep Reinforcement Learning-based Trust Management Scheme for Software-defined Vehicular Networks. In *Proceedings of the 8th ACM Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications*, Montreal, QC, Canada, (pp. 17). doi: 10.1145/3272036.3272037.
40. Latah, M., & Toker, L. (2019). Artificial Intelligence Enabled Software Defined Networking: A Comprehensive Overview. *IET Netw.*, 8, 2, 7999. doi: 10.1049/iet-net.2018.5082.
41. Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Mach. Learn.*, 8, 3, 279292. doi: 10.1007/BF00992698.
42. Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). A Brief Survey of Deep Reinforcement Learning. *ArXiv*. doi: 10.1109/MSP.2017.2743240.
43. Suh, D., Jang, S., Han, S., Pack, S., Kim, T., & Kwak, J. (2016). On performance of OpenDaylight clustering, In *IEEE NetSoft Conference and Workshops (NetSoft)* (pp. 407410). doi: 10.1109/NETSOFT.2016.7502476.
44. Smida, K., Tounsi, H., Frikha, M., & Song, Y.Q. (2020). Efficient SDN Controller for Safety Applications in SDN-Based Vehicular Networks: POX, Floodlight, ONOS or OpenDaylight?. In *IEEE Eighth International Conference on Communications and Networking (ComNet)* (pp.16). doi:10.1109/ComNet47917.2020.9306095.
45. Mnih, V., & al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518, 7540. doi: 10.1038/nature14236.
46. Neghabi, A. A., Jafari Navimipour, N., Hosseinzadeh, M., & Rezaee, A. (2018). Load Balancing Mechanisms in the Software Defined Networks: A Systematic and Comprehensive Review of the Literature. *IEEE Access*, 6, 1415914178. doi: 10.1109/ACCESS.2018.2805842.
47. Mininet-WiFi: Emulating software-defined wireless networks. Retrieved feb 23, 2019, from [https://www.researchgate.net/publication/295861311\\_Mininet-WiFi\\_Emulating\\_software\\_defined\\_wireless\\_networks](https://www.researchgate.net/publication/295861311_Mininet-WiFi_Emulating_software_defined_wireless_networks).
48. Behrisch, M., Bieker, L., Erdmann, J., & Krajzewicz, D. (2011). SUMO - Simulation of Urban MObility An Overview. In *The Third International Conference on Advances in System Simulation*.
49. Iperf2. SourceForge. (2019). Retrieved feb 23, 2019, from <https://sourceforge.net/projects/iperf2/>
50. Empowering App Development for Developers — Docker. Retrieved jul 29, 2020, from <https://www.docker.com/>
51. Service requirements for V2X services. (2017). Retrieved jan 22, 2021, from [https://www.etsi.org/deliver/etsi\\_ts/122100.122199/122185/14.03.00.60/ts\\_122185v140300p.pdf](https://www.etsi.org/deliver/etsi_ts/122100.122199/122185/14.03.00.60/ts_122185v140300p.pdf)
52. Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; (2013). Retrieved Aug 11, 2020, from [https://www.etsi.org/deliver/etsi\\_en/302600\\_302699/30263703/01.02.00.20/en\\_30263703v010200a.pdf](https://www.etsi.org/deliver/etsi_en/302600_302699/30263703/01.02.00.20/en_30263703v010200a.pdf).