Dynamic clustering of software defined network switches and controller placement using deep reinforcement learning

El Hocine Bouzidi, Abdelkader Outtagarts, Rami Langar, Raouf Boutaba

▶ **To cite this version:**

El Hocine Bouzidi, Abdelkader Outtagarts, Rami Langar, Raouf Boutaba. Dynamic clustering of software defined network switches and controller placement using deep reinforcement learning. Computer Networks, 2022, 207, pp.108852. ⟨10.1016/j.comnet.2022.108852⟩. ⟨hal-04512391⟩

HAL Id: hal-04512391

https://hal.science/hal-04512391v1

Submitted on 22 Jul 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons CC BY-NC 4.0 - Attribution - Non-commercial use - International License

Version of Record: https://www.sciencedirect.com/science/article/pii/S1389128622000640
Manuscript_c417d8abd45a8d6d1a8670b3ab621f95

Click here to view linked References

Dynamic Clustering of Software Defined Network Switches and Controller placement using Deep Reinforcement Learning

EL Hocine Bouzidi[a,b], Abdelkader Outtagarts[a], Rami Langar[b,c], Raouf Boutaba[d]

[a]*Nokia Bell Labs, Villarceaux Center - Route de Villejust 91620 Nozay, France*
[b]*LIGM CNRS-UMR 8049, University Gustave Eiffel, 77420 Marne-la-Vallée, France*
[c]*Software and IT Engineering Department, ÉTS, Montréal (QC), Canada*
[d]*D.R. Cheriton School of Computer Science, University of Waterloo, Waterloo (ON), Canada*

Abstract

Software defined networking (SDN) has emerged as a promising alternative to the traditional networks, offering many advantages, including flexibility in network management, network programmability and guaranteeing application Quality-of-Service (QoS) requirements. In SDN, the control plane is separated from the data plane, and deployed as a logically centralized controller. However, due to the large scale of networks as well as latency and reliability requirements, it is necessary to deploy multiple controllers to satisfy these requirements. The distributed deployment of SDN controllers unveiled new challenges in terms of determining the number of controllers needed, their locations and the assignment of switches to controllers that minimizes flow set delay. In this context, we propose, in this paper, a new method that dynamically computes the optimal number of controllers, determines their optimal locations, and at the same time partitions the set of data plane switches into clusters and assigns them to these controllers. First, we mathematically formulate the controller placement as an optimization problem, whose objectives are to minimize the controller response time, that is the delay between the SDN controller and assigned switches, the Control Load (CL), the Intra-Cluster Delay (ICD) and the Intra-Cluster Throughput (ICT). Second, we propose a simple yet computationally efficient heuristic, called Deep Q-Network based Dynamic Clustering and Placement (DDCP), that leverages the potential of reinforcement and deep learning techniques to solve the aforementioned optimization problem. Experimental results using ONOS controller show that the proposed approach can significantly improve the network performances in terms of response time and resource utilization.

Key words: SDN, Controller placement, DQN, Clustering, ONOS

1. Introduction

SDN is an emerging paradigm that allows dynamicity, automation, flexibility and centralized management of the underlying network contrary to traditional networks making it ideal to designing Beyond 5G networks (B5G) that involve essentially higher capacity and lower latency. SDN separates the control plane that is responsible for making routing decisions and the Forwarding Plane, that is only required to perform packet forwarding according to the received routing strategies from the control plane. This communication is enabled via Application Programming Interfaces (APIs) such as the Representational State Transfer (REST) on the northbound interface and OpenFlow on the southbound interface. Even the separation of the control and data planes brings technical benefits, it can cause several drawbacks such as the number of controllers needed, their optimal corresponding placement and which data plane devices to be controlled by which controller.

With the advent of B5G networks, the network size grows exponentially. Therefore, the deployment of a single controller to control the whole network brought greater challenges to the SDN controller's processing capabilities in terms of scalability, performance and reliability. To this end, the use of multiple controllers is primordial, which can be achieved by two main strategies: distributed controllers and replicated controllers. In the replicated controllers strategy, several copies of the SDN controllers have always the same information and keep the full state of the network. Even the fast recovery time when a controller fails, keeping the set of replicated controllers aware of every networking operation, can bring expensive overhead in terms of resource utilisation such as CPU, RAM and storage. On the other hand, the distributed controllers strategy fragments the network into smaller domains, each supervised by a dedicated controller. The set of controllers supervising these domains communicate to each other by their west/eastbound interfaces.

The distributed controllers strategy unveils several challenges that must be considered such as : i) the optimal number of controllers needed for a given network topology, in such a way that, each controller must be not overloaded neither underutilized, ii) the optimization of the controller placement is another concern, as it widely impacts on several fronts such as flow setup latency, resiliency and load balancing and iii) the clustering of the data plane devices in order to improve the intra-cluster performances.

Email addresses: el_hocine.bouzidi@nokia.com (EL Hocine Bouzidi), abdelkader.outtagarts@nokia-bell-labs.com (Abdelkader Outtagarts), rami.langar@univ-eiffel.fr; rami.langar@etsmtl.ca (Rami Langar), rboutaba@uwaterloo.ca (Raouf Boutaba)

© 2022 published by Elsevier. This manuscript is made available under the CC BY NC user license
https://creativecommons.org/licenses/by-nc/4.0/

To this end, we propose, in this paper, a new approach that determines, the optimal number of controllers, their optimal placements as well as the optimal clustering of data plane switches. To do so, we first mathematically formulate the controller placement problem as an optimization one, whose objective is to minimize the controller response time, which corresponds to the delay between the SDN controller and assigned switches, the controller resource utilization, the ICD and the ICT. As the formulated optimization problem is an NP-hard problem [1], we first propose to model the problem as a Markov Decision Process (MDP) and solve it by a Deep Q-Network (DQN) approach. Then, we propose a simple yet efficient heuristic, DDCP that dynamically interacts with the DQN agent to solve the aforementioned optimization problem.

To the best of our knowledge, we are the first to propose and validate such solution using DQN approach. The main contributions of our paper can be summarized as follows:

- First, we mathematically model the controllers clustering and placement problem as an optimization problem whose objective is to minimize the controller response time, the controller resource utilization, the ICD and the ICT.
- Second, we formalize our problem as MDP with appropriate states and actions, then propose using a DQN approach to solve it.
- Third, we propose simple yet efficient heuristic algorithm called DDCP, that takes as inputs the set of switches and controllers, the trained DQN agent and outputs the optimal clustering in both the control and data planes.

The remainder of this paper is organized as follows. Section 2 presents the state-of-the-art in several related research topics and an overview of the DQN method. In Section 3, we discuss the architecture of the proposed framework, describes the problem formulation and presents our DQN-based heuristic. Section 4 evaluates the proposed method. We finally conclude this paper in Section 5.

2. Related Work and Overview

2.1. Related Work

In recent years, with the development of distributed controller architecture such as DevoFlow [2], Kandoo [3], HyperFlow [4], Onix [5], the problem of determining the number of controllers and their placements in SDN has received considerable attention and attracted many researchers.

Authors in [6] presented a comprehensive survey on the controller placement problem (CPP) in SDN. The objective of this survey is to introduce the CPP in SDN and highlight its significance. Then, presenting the classical CPP formulation along with its supporting system model as well as discussing a wide range of the CPP modeling choices and associated metrics.

In [7], authors tackled the multi-controller placement issue in SDN and proposed a new approach with network partition technique. In this approach, the entire network is divided into multiple subnetworks and for each subnetwork, one or more controllers are deployed correspondingly. Specifically, the clustering algorithm is leveraged to partition the network into subnetworks and an optimized K-means algorithm is proposed to shorten the maximum latency between the centroid and associated switches in the subnetwork. The authors optimized the K-means algorithm for clustering to minimize the overall latency of the network.

Authors in [8] considered node-to-controller latency for their controller placement optimization. They presented POCO, a framework for Pareto-based Optimal Controller placement that provides operators with Pareto optimal placements with respect to different performance metrics. This framework does not segment the network into multiple domains by treating the network as a whole and the controllers work collaboratively, which requires frequent exchange of state information between the controllers to achieve an accurate global state.

Authors in [9], defined a capacitated controller placement problem (CCPP), taking into consideration the load of controllers, and they introduced an efficient algorithm to solve the problem. The objective is to reduce the number of controllers and their loads. However, the placement is not based on these criteria. Instead of considering the propagation delay between the switches and the controller and ignoring the critical factor in the actual network which is the switch weights, the authors in [10] defined a new metric : $total\text{-}flow\text{-}request\text{-}cost$. This metric takes into account the switch weights, switch-to-controller routing costs, and inter controller routing costs, where the goal of this metric is to minimize *Packet-in* messages cost from the switch to the controller, and information exchange cost between controllers.

In [11], authors first, considered the controller placement problem from the perspective of energy consumption. Then, they formulated the energy-aware controller placement problem based on a Binary Integer Program (BIP), which has the latency of paths and the load of controllers as constraints for minimizing the energy consumption and proposed a genetic algorithm to solve the formulated problem.

Authors in [12] investigated different approaches to determine the optimal number of controllers for deployment in a given SDN network, by taking into account the latency objective. This study was followed by determining the optimal placements of the SDN controllers.

In [13], [14], authors considered several parameters such as the number of controllers, the location of controllers and the set of switches of the network to achieve a high reliability. In [14], authors introduced QoS-Guaranteed Controller Placement problem, which is to place the minimum number of controllers in the network such that the response time of controllers can meet a given delay bound. Three heuristics was proposed for the proposed approach: incremental greedy algorithm, primal-dual-based algorithm and network partition-based algorithm. Results show superiority of the proposed incremental greedy method on the other two methods on all input topologies.

Recently, Machine Learning (ML) techniques have been used widely to solve complicated decision-making complex

problems arising in Virtual Network Function (VNF) placement and traffic engineering in SDN based networks. Authors in [15] addressed the allocation of Virtual Network Function-Forwarding Graph (VNF-FGs) for realizing network services. First, they modeled the VNF-FG allocation problem as a MDP. Then, they solved it by a DQN approach. Simulation results clearly showed the effectiveness of the deep learning process, where the performance of the proposed approach is improved over time. In [16], authors proposed a simple heuristic algorithm to identify the number of controllers and their locations in SDN networks leveraging a learning automaton (LA) approach, while ensuring that propagation latency from any node to its closest controller does not exceed a threshold.

2.2. *DDCP Key Assumptions and Ideas*

Different from the approaches proposed in related works, we propose in this paper the DDCP approach, by exploring the potential of reinforcement learning techniques, such as DQN, for the clustering and placement of controllers, while taking into account four performance metrics: the Control Delay (CD), the Control Load (CL), the Intra-Cluster Delay (ICD) and the Intra-Cluster Throughput (ICT). The idea behind using a DQN approach to solve the clustering and controllers' placement problem, is to use only one step (after training) to get the optimal controllers placement as well as the corresponding data plane domains of switches, while considering the four performance metrics. The advantage of using DQN is the ability to improve network performances, while maximizing or minimizing a certain reward. This reward can be modeled as an extensible function, taking into account several parameters (that will be listed in the next section) and can be further extended after training. Moreover, to overcome the problem of scalability and extensibility presented in related works, we implement our DQN agent in the knowledge plane, which is not part of the SDN architecture. This makes our DDCP approach independent of the technology implemented in both the control and data planes.

Table 1 positions our proposal vs. the state-of-the-art approaches. Specifically, we highlight the added-values that our DDCP scheme provides based on a set of critical performance metrics, such as the CL, the CD, the ICD, the ICT, the number of controllers (Nbr of CTRLs) and using or not a DQN agent to solve the controller placement problem.

2.3. *Deep Q-Network (DQN)*

The Q-learning technique (QL) is basically based on an autonomous agent that interacts with the environment by sequentially taking actions, while maximizing cumulative rewards [20][21]. As shown in Fig. 1, this can be described as a MDP in which the next state s_{i+1} depends on the current state s_i and the selected action by the agent according to a specific state transition probability distribution $P(s_i, a_i, s_{i+1})$, which represents the probability of switching to state s_{i+1} after action a_i in state s_i. In QL, the agent senses the environment by identifying the current state s_i and then selects the action a_i to execute. The environment subsequently feeds back a reward to the agent, while updating the current state to the new state. Then,

Table 1: Added-value of our contribution

Ref	CL	CD	ICD	ICT	Nbr of CTRLs	DQN
[7]	✗	✓	✓	✗	✗	✗
[8]	✗	✓	✗	✗	✗	✗
[9]	✓	✗	✗	✗	✗	✗
[10]	✓	✗	✓	✗	✗	✗
[11]	✓	✗	✓	✗	✗	✗
[12]	✗	✗	✓	✗	✓	✗
[13]	✗	✓	✓	✗	✓	✗
[14]	✗	✓	✓	✗	✓	✗
[16]	✗	✓	✗	✓	✓	✗
[17]	✓	✓	✗	✗	✗	✗
[18]	✓	✓	✗	✗	✗	✗
[19]	✓	✓	✗	✗	✗	✗
DDCP	✓	✓	✓	✓	✓	✓

the trajectory of states, actions and rewards constitute a MDP: $s_0, a_0, r_0, s_1, a_1, r_1 ...$ The objective is to learn the best policy $\pi(a_i|s_i)$, while maximizing the cumulative rewards of the current and next states which can be written as follows::

$$R = \sum_{i=0}^{n} \varphi^i r_{i+1} \tag{1}$$

Where $\varphi \in [0, 1]$ is a factor to discount future rewards. Given a policy π, the expectation of accumulated rewards from action a_i in a state s_i can be estimated by the Q-value function $Q_\pi(s_i, a_i) = E[R|s_i, a_i, \pi]$. Thereafter, the best policy corresponds to the highest Q-value in each state: $Q^*(s_i, a_i) = max_\pi Q_\pi(s_i, a_i)$. This function can be defined recursively according to the Bellman equation as follows:

$$Q_\pi(s_i, a_i) = r_i + \varphi . Q^*(s_{i+1}, a_{i+1}) \tag{2}$$

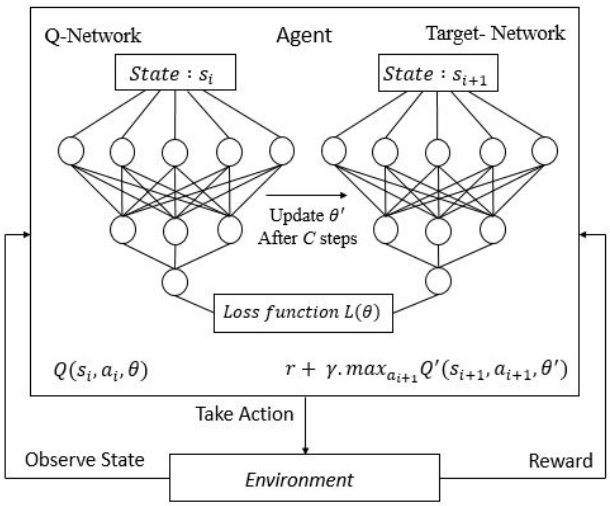

Figure 1: Deep Q-Network (DQN) architecture [20][21]

The policy π can be improved by dynamically updating the $Q_\pi(s_i, a_i)$ as follows:

$$Q_\pi(s_i, a_i) \leftarrow Q_\pi(s_i, a_i) + \eta.\Delta \tag{3}$$

$$\Delta = r_i + \varphi.max_{a_{i+1}} Q_\pi(s_{i+1}, a_{i+1}) - Q_\pi(s_i, a_i) \tag{4}$$

Where $\eta \in [0, 1]$ is the learning rate, and the temporal difference (TD) error Δ corresponds to the correction for the Q-value estimation. The QL technique stores and updates the Q-values in look-up tables, which makes it slow to reach the best policy when exploring the entire table if the number of possible states becomes very large. This affects significantly the performance of Q-learning. To cope with this challenge, the DQN makes use of Neural Networks (NN) to approximate the estimation of the Q-value function. The DQN networks takes as input the state vector. The output is a vector of action Q-values, and its corresponding Loss function is constructed based on the mean square deviation defined as follows :

$$L(\theta_i) = (TargetQ - Q(s_i, a_i, \theta_i))^2 \tag{5}$$

$$TargetQ = r_i + \varphi.max_{a_{i+1}} Q(s_{i+1}, a_{i+1}, \theta_i) \tag{6}$$

Where θ_i is the network parameter at iteration i. It is worth noting that the convergence of the Loss function $L(\theta_i)$ is not stable when using only one Neural Network. To improve the convergence stability, DQN adopts the method called *Experience Replay*, which corresponds to creating two Neural networks, that have the same architecture, with parameters θ and $\theta^{'}$. The first one is used to retrieve Q-values, while the second one includes all updates in the training. After C steps the target network parameters θ' are updated. This mechanism is illustrated in Fig. 1.

3. DDCP Approach

In this section, we present our approach for the clustering and placement of controllers in SDN using DQN. Firstly, we explain the overall framework. Thereafter, the problem formulation and the Deep Q-Network Agent will be described.

3.1. DDCP Architecture

We design our framework according to the Knowledge-Defined Networking (KDN) paradigm [22], by introducing the knowledge plane to the conventional SDN paradigm, in which we exploit the control plane to have a global view of the network (cf. Fig. 2).

Fig. 2 presents our system architecture, which consists of four planes: Data plane, Control Plane, Management Plane and Knowledge Plane.

The data plane that consists of programmable forwarding devices, in charge of data packet processing and forwarding. These devices have no embedded intelligence to take decisions and rely on the control plane to populate their forwarding tables

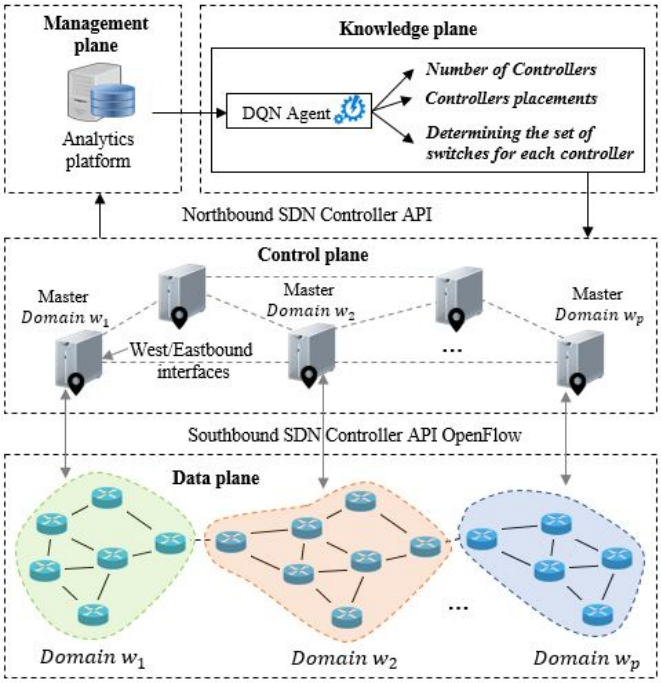

Figure 2: DDCP Architecture

and update their configurations based on the OpenFlow protocol. Moreover, the data plane is divided into multiple domains and each of them is supervised by a dedicated controller.

The control plane is considered as the brain of the SDN network, which incorporates the whole intelligence by abstracting the management and global view of the network in a set of distributed controllers in different locations. Each controller may not have full control or knowledge of the network status and has the responsibility for only a portion of the network (i.e., domain). It communicates with the other controllers through the West/Eastbound interfaces.

The management plane ensures the correct operation and performance of the network by collecting the network measurement from the control plane Network Measurement module, in order to provide network analytic. The collected statistics will be analyzed and sent to the knowledge plane.

In order to not affect the control plane performances, the process of deploying distributed controllers needs to be fully automated and can be ensured by the knowledge plane. This latter exploits the control plane and the management plane by taking the data from the LP as input to be fed to ML algorithms, which will convert them to the form of knowledge. Precisely, it learns the behavior of the network, by processing the collected statistics, then determines the number of controllers, their locations and the set of switches embedded in each controller's domain, by deploying a DQN agent. The output of the DQN agent is transmitted to the control plane through the Northbound SDN controller API.

In what follows, we present our controller placement problem formulation, then we detail the Deep Q-Network Agent.

3.2. Problem Formulation

We model the SDN network as an undirected graph $G = (V, E)$, where $V = \{v_j\}$ is the set of switches and $|V| = k$ is the number of switches and E is the set of edges (i.e., links between switches). The control plane consists of a set of controllers $C = \{c_i\}$, where $|C| = n$ denotes the number of controllers. On the other hand, the data plane is fragmented into a set of domains $W = \{w_l\}$, each domain $w_l = \{v_j\}, v_j \in V$ supervised by a controller, and $|W| = p$ denotes the number of domains. The solution of our Controller Placement Problem (CPP) can be represented as a binary vector $F = (F_1, F_2, ..., F_n) \in R^n$, where F_i is given by:

$$F_i = \begin{cases} 1, & \text{if controller } c_i \text{ is selected} \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

Since our objective is to optimize the number of deployed controllers, while guaranteeing the QoS requirements of all traffic requests in the network, we define here-after four different variables to compute the number of selected controllers: Control Load (CL), Control Delay (CD), Intra-Cluster Delay (ICD), and Intra-Cluster Throughput (ICT).

3.2.1. Number of Selected Controllers

To determine the number of deployed controllers p (i.e., the number of data plane domains), we first, define the variable R^t_{ij} representing the total flow request from switch (j) to controller (i) at time (t), which corresponds to the number of *Packet-In* messages generated by the switch. It is worth noting that, the *Packet-In* messages are generated and sent from switches to the controller when there is no matching flow entries in their flow tables. Secondly, we assume that the portion of resources consumed by one flow request concerns essentially the CPU and RAM, and can be written as follows:

$$\lambda_i = \frac{CPU_{flow_i}}{CPU_i} + \frac{RAM_{flow_i}}{RAM_i} \tag{8}$$

Where CPU_i and RAM_i correspond, respectively, to the maximum capacity of CPU and RAM of the controller c_i. As CPU and RAM use different units, we divide them over their corresponding maximum values to get normalized data. Therefore, the number of selected controllers can be written as follows:

$$p = \sum_{i=1}^{n} \sum_{j=1}^{k} R^t_{i,j}.\lambda_i \tag{9}$$

3.2.2. Control Load (CL)

We define the CL as the load incurred by all the switches belonging to the same domain. To this end, we use a decision variable $H^t_{i,j}$ to determine the relationship between the controller (i) and the switch (j) as follows:

$$H^t_{i,j} = \begin{cases} 1, & \text{if switch } v_j \text{ is controlled by } c_i \text{ at time } t \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

The load of the controller c_i can be thus given by:

$$CL^t_i = \sum_{j=1}^{k} H^t_{i,j}.R^t_{i,j}.\lambda_i \tag{11}$$

To avoid the overloading of some controllers and the underutilization of others, we balance the traffic load between the set of selected p controllers. To this end, we determine the global CL as follows:

$$CL^t = \frac{1}{p} \sum_{i=1}^{p} |CL^t_i - CL^t_{avg}| \tag{12}$$

Where CL^t_{avg} denotes the average of the CL of all selected p controllers.

3.2.3. Control Delay (CD)

The CD corresponds to the average response time of the controller c_i, which is defined as follows:

$$CD^t_i = PD^t_i + 2.CMD^t_i \tag{13}$$

Where PD^t_i and CMD^t_i are, respectively, the processing delay and the communication delay of the controller c_i at time t. We used two times of the communication delay since the *Packet-In* comes from the switch to the controller and returns back to the switch. In this way, the global CD is determined as the average of the CD of the set of selected controllers, which can be written as follows:

$$CD^t = \frac{1}{p} \sum_{i=1}^{p} CD^t_i \tag{14}$$

According to [23], the processing delay is determined as follows:

$$PD^t_i = \frac{1}{\varphi_i - \theta_i} \tag{15}$$

Where φ_i and θ_i are, respectively, the capacity and the workload of the controller c_i. The communication delay is determined as follows:

$$q^t_i = \sum_{j=1}^{k} H^t_{i,j} \tag{16}$$

$$CMD^t_i = \sum_{j=1}^{k} \frac{H^t_{i,j}.d^t_{i,j}}{q^t_i} \tag{17}$$

Where q^t_i denotes the number of switches supervised by the controller c_i and $d^t_{i,j}$ denotes the delay between the controller c_i and the switch v_j. The latter is measured based on our previous work in [24].

3.2.4. Intra-Cluster Delay (ICD)

This metric corresponds to the average value of the propagation delays between all the switches belonging to the same cluster i, which can be written as follows:

$$ICD_i^t = \sum_{v_e \in w_i, v_m \in w_i} \frac{A(v_e, v_m).\psi(v_e, v_m)}{\varsigma_i} \tag{18}$$

Where $A(v_e, v_m)$ denotes the delay between nodes v_e and v_m in the domain w_i, ς_i is the number of links of the cluster i, and (v_e, v_m) is a decision variable representing the relationship between any two switches, which is defined as follows:

$$(v_e, v_m) = \begin{cases} 1, & \text{if } v_e \text{ is connected to } v_m \\ 0, & \text{otherwise} \end{cases} \tag{19}$$

Then, the global ICD of the set of p clusters is determined as follows:

$$ICD^t = \frac{1}{p} \sum_{i=1}^{p} ICD_i^t \tag{20}$$

3.2.5. Intra-Cluster Throughput (ICT)

This metric corresponds to how much data can be transferred by a specific data plane cluster (i) within a given timeframe, which referred to us ICT_i^t. Then, the global ICT of the set of p clusters is determined as follows:

$$ICT^t = \frac{1}{p} \sum_{i=1}^{p} ICT_i^t \tag{21}$$

3.3. Controller Placement and Switches Migration

We consider the controllers as images installed in different servers located in different locations. In this way, the action of controllers placement refers to instantiating a container from the image in the corresponding server located in a specific location. To this end, we consider $c_i, i \in [1, n]$ as the instance of the controller in the server or location i. Also, the deselection follows the same logic by just deleting the instance or container c_i.

It is worth noting that the clustering of control and data planes as well as the placement of the controllers happen when certain controllers are overloaded, while others are underutilized. Hence, we define in equation (22) the load balancing factor between clusters, calculated as follows:

$$CL_{mig} = \frac{CL^t}{max(CL_i)} \times 100 \tag{22}$$

Note that, finding new data plane clusters leads to migrating a set of switches from old clusters controlled by specific controllers to new clusters controlled by new controllers.

3.4. Proposed Optimization Model

The objective of our optimization model is to minimize the aforementioned performance metrics including CL, CD, ICD and ICT. This can be achieved as follows:

$$Min(\alpha \times \frac{CD^t}{CD_{max}} + \beta \times \frac{CL^t}{CL_{max}} + \gamma \times \frac{ICD^t}{ICD_{max}} - \rho \times \frac{ICT^t}{ICT_{max}}) \tag{23}$$

Subject to :

$$\forall\, i \in [1, p] : CL_i^t < M.F_i \tag{24}$$

$$\forall\, (w_u, w_v) \in W^2 : w_u \cap w_v = \oslash \tag{25}$$

$$\text{if controllers } c_i \text{ and } c_j \text{ are selected: } c_i \neq c_j \tag{26}$$

$$|W| = p \tag{27}$$

$$\forall\, (i, l) \in [1, p]^2, \forall\, j \in [1, k] : (d_{i,j} < d_{l,j}) \Rightarrow (H_{i,j}^t > H_{l,j}^t) \tag{28}$$

$$\forall\, j \in [1, k] : H_{i,j}^t \leq F_i \tag{29}$$

$$\forall\, i \in [1, p] : (\sum_{j=1}^{k} H_{i,j}^t = 0) \Rightarrow (F_i = 0) \tag{30}$$

$$\forall\, i \in [1, p] : CD_i^t \leq \sigma_i \tag{31}$$

$$\forall\, (v_e, v_m) \in V^2 : A(v_e, v_m) \leq \delta \tag{32}$$

As the objective function involves different parameters with different measurement units, we have divided each metric over its corresponding maximum value to have a normalized objective function. Note that these maximum values are determined by the network operator and correspond to physical characteristics of involved network devices. Note also that α, β, γ, and ρ are adjustable weighting factors determining the degree of importance of the CD, the CL, the ICD and the ICT metrics, respectively, such that $\alpha + \beta + \gamma + \rho = 1$.

Constraint (24) forces all controllers to not be overloaded. Constraint (25) means that all domains do not overlap and each node belongs to only one domain. Constraint (26) forces the set of controllers to be selected only once. Constraint (27) insures that the number of selected controllers is the same as the number of data plane domains. Constraint (28) is the mapping constraint ensuring that a switch must be mapped to the closest controller in terms of delay. Recall that, we refer to our work in [24] to measure the delay between the switches in the data plane as well as between the switches and their corresponding controllers, where the delay is measured based on the times of sending and receiving a specific packet probe from the controller to the switches in the data plane. Constraint (29) means that a switch is mapped to a controller if the latter exists and is selected. Constraint (30) means that if there is no switch mapped to a controller then the latter will be powered off. Constraint (31) forces the CD metric to not exceed a certain threshold σ_i, and finally constraint (32) forces the links of each cluster to not be delayed.

The formulated optimization problem is an NP-hard problem [1], where the optimal solution is very difficult to obtain in general. To this end, we propose to solve it by modeling and training a Deep Q-Network Agent, as will be detailed in the next section.

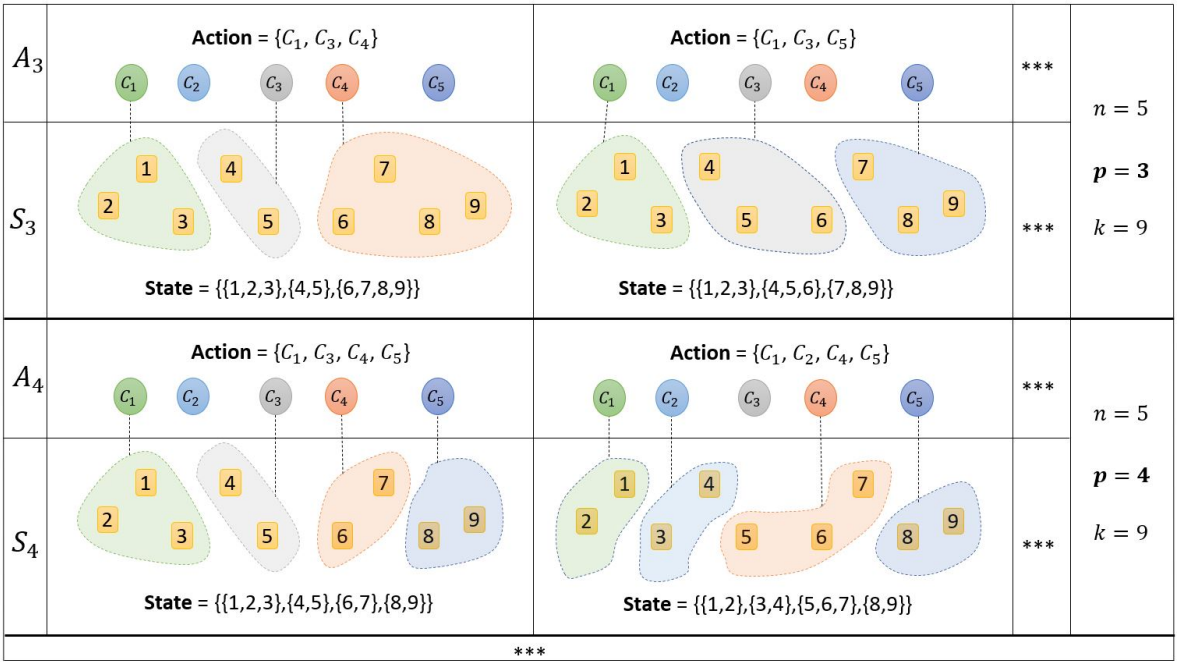

Figure 3: Demonstration of the DQN State and Action spaces using a simple topology

3.5. Deep Q-Network Agent

To dynamically determine the optimal number of controllers and their optimal placements while considering the propagation delay between the switches and the controller, the controller resource utilization, the ICD and the ICT, we propose to model a DQN Agent based on a MDP. In this way, the DQN agent interacts with the environment through three signals: $State$, $Action$ and $Reward$.

3.5.1. State

The State $S_{t,p}$ corresponds to the partitioning of the data plane into p domains. To do so, we define it as a vector of $w_{t,i}$ ($i \in [1, p]$), and can be written as follows:

$$S_{t,p} = [w_{t,1}, w_{t,2}, ..., w_{t,p}]$$

Where $w_{t,i} = [v_e, ..., v_m], \forall (v_e, v_m) \in w_{t,i}^2$ is a vector representing a cluster of switches, such that $\sum_{i=1}^{p} |w_{t,i}| = k$. Recall that, k corresponds to the total number of switches in the data plane.

Let S_p denote the set of all states corresponding to a specific value of p, written as follows:

$$S_p = [S_{1,p}, S_{2,p}, ..., S_{T_p,p}]$$

Where T_p corresponds to the number of states corresponding to p clusters. It is worth noting that, the set of states are constructed by respecting the set of constraints indicated in Section 3.4.

3.5.2. Action

The action taken by the agent $A_{r,p}$ is characterized by a vector representing a selected set of p controllers from the available n controllers, which is defined as follows:

$$A_{r,p} = [c_{r,1}, ..., c_{r,p}], \forall c_{r,i} \in C$$

Where r represents the action number. We denote the set of all actions corresponding to a specific value of p as follows:

$$A_p = [A_{1,p}, A_{2,p}, ..., A_{R_p,p}]$$

Where R_p corresponds to the number of actions corresponding to p clusters. It is worth noting that, the set of actions are constructed by respecting the set of constraints (26) and (27) indicated in Section 3.4. Recall that, constraint (26) avoids selecting one controller for more that one cluster and constraint (27) ensures that the number of controllers is the same as the number of data plane domains.

3.5.3. Reward

The "Reward" function R of the agent consists in minimizing the normalized objective function defined in (23), and can be thus written as follows:

$$R = \alpha \times \frac{CD}{CD_{max}} + \beta \times \frac{CL}{CL_{max}} + \gamma \times \frac{ICD}{ICD_{max}} - \rho \times \frac{ICT}{ICT_{max}} \quad (33)$$

It is worth noting that the proposed DQN agent consists in determining the best mapping between the set of states and the set of actions, while maximizing $1/R$ (i.e., minimizing R).

In order to give more detail on the DQN design, we propose to illustrate it graphically in a small topology with 9 switches in the data plane and 5 controllers in the control plane, as shown in Fig. 3. We can see that, when $p = 3$ the data plane is fragmented into three domains as well as only three controllers are

used to supervise each domain and the rest of controllers are not instantiated. In this case, the state corresponds to the selected three domains and the action corresponds to the selected three controllers. When $p = 4$, we can see that the shapes of the state and action vectors are changed, where four controllers from five are selected for each action and the data plane is fragmented into four domains.

Recall that, the number of controllers to be selected, corresponds to the number of data plane domains or clusters, and can take values from 1 to n, where n corresponds to the total number of controllers in the control plane.

3.6. DDCP Heuristic

Algorithm 1: DDCP algorithm

```
1:  procedure CLUST(S = {S_1, ..., S_n}, A = {A_1, ..., A_n})
2:      Max_Reward ← 0
3:      p ← 1
4:      while p ≤ n do
5:          for each state ∈ S_p do
6:              Reward, Action ← DQN(state, S_p, A_p)
7:              if Reward > Max_Reward then
8:                  Max_Reward ← Reward
9:                  Select_St ← State
10:                 Select_Act ← Action
11:                 Select_p ← p
12:             end if
13:         end for
14:         p ← p + 1
15:     end while
16:     return Select_St, Select_Act, Select_p
17: end procedure
18: procedure MIGRATION(Select_St, Select_Act)
19:     if CL_mig > Threshold then
20:         for each CTRL ∈ Select_Act do
21:             for each Domain ∈ Select_St do
22:                 if Map(CTRL, Domain) then
23:                     for each Switch : s ∈ Domain do
24:                         if CTRL_old(s) ≠ CTRL then
25:                             Remove(s, CTRL_old)
26:                             Assign(s, CTRL)
27:                         end if
28:                     end for
29:                 end if
30:             end for
31:         end for
32:     end if
33: end procedure
```

As the state $S_{t,p}$ and action $A_{r,p}$ take different forms or shapes for each value of p, we propose to split the state space (i.e., training data) based on p values. Then, we train the DQN agent separately for each value of p (i.e., DQN($S_{t,p}$, $A_{r,p}$)). In this way, to determine the optimal number of controllers p and the best mapping between the set of states (clusters of switches) and the set of p controllers, we propose the DDCP heuristic (i.e., Algo. 1):

The DDCP algorithm consists of two main procedures: *Clust* and *Migration*. The *Clust* procedure is called to determine the set of controllers and their corresponding placements as well as the set of data plane clusters. It works as follows: it takes as input the set of splitted states and the set of splitted actions based on p: $S = \{S_1, S_2, ..., S_n\}, A = \{A_1, A_2, ..., A_n\}$. Recall that, S_p corresponds to the space of states where the number of controllers as well as the number of data plane domains is p. Then, by using the trained DQN Agent, it finds the optimal $State$, $Action$ corresponding to the maximum reward (lines 4-15) iteratively for each sub-states $S_p \in S, p \in [1, n]$. The output of the *Clust* procedure is the optimal number of controllers to be deployed ($Select_p$), their corresponding ID ($Select_Act \in A_{Select_p}$), and the corresponding set of switch clusters ($Select_St \in S_{Select_p}$) (line 16).

On the other hand, the *Migration* procedure is called when the control plane load is not well-balanced (lines 19). In this case, it takes as input the output of the *Clust* procedure (i.e., $Select_p$, $Select_Act$, $Select_St$). Then, it migrates the set of switches to the new cluster if the new controller is different from the old one (lines 20-31).

Note that, the *Map* function indicates if a switch in $Select_St$ is mapped to a controller in $Select_Act$. The $CTRL_old$ represents the controller of a specific switch before migration.

4. Performance Evaluation

In this section, we evaluate the efficiency of our proposed approach. We start by presenting our experimental setup. Then, we present the experimental results.

4.1. Experimental Setup

First, the control plane is deployed as a cluster of a set of dockerized OpenFlow ONOS [22] controllers. Then, the DQN agent is implemented based on Python [15] and dockerized on Docker Containers [16]. The latter interacts with the control plane based on the ONOS Northbound API. The control plane consists of 12 controllers. We used the network emulation tool OpenvSwitch [23] to implement the experimental topology, which consists of 32 nodes. To generate traffic among hosts, we used Iperf [24]. The control plane Network Measurement modules collect statistics (latency, throughput, and per-flow size) from the devices and report those time-series statistics to the InfluxDb database [20]. Note that, to show the importance of using DQN on a large space of states and actions, we augmented the collected statistics by using data generated with Python.

Recall that, the DQN model consists of two neural networks, designed to improve the convergence of the Cost Function in (5). One neural network is called *Q-Network* to estimate the *Q*-Values and the second one is called the *Q-target* to estimate the target network, according to the mechanism shown in Fig. 1.

Table 2: DQN parameters

Name	Value
Dense layers	2
Control plane capacity	10
Data plane capacity	32
Minimum number of switches per cluster	1
Maximum number of switches per cluster	13
Q-target network update frequency	200
Learning rate	0.01
Discounted factor	0.6
Mini-batch size	32
Final exploration rate	0.2
Memory size	2000 units
Number of episodes	1000

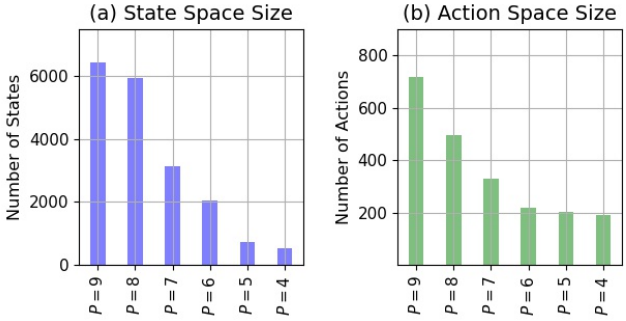

Figure 4: Number of States and Actions under different number of clusters p

We built and trained the DQN model by using the Tensorflow library [21], by deploying separately the two neural networks (i.e., *Q-Network*, *Q-target*), which have the same architecture. The DQN parameters are illustrated in Table 2. In particular, both the *Q-Network* and the *Q-target* consist of 2 dense layers. The number of data plane switches is 32 and the number of controllers in the control plane is 10.

Considering the implemented topology, and in order to not have a huge state space size, the training data are built as follows: i) the minimum and maximum number of switches in each cluster are fixed to 1 and 13, respectively, ii) the clusters where no link between the generated switches of a specific cluster are ignored. On the other hand, the training data (i.e., the set of States) can be classified based on p, as the State corresponds to a vector of p sub-vectors. In this way, both the set of States and the set of Actions are splitted based on p, as illustrated in Fig. 4. Then, we trained separately a set of DQN agents according to p. Recall that, we denoted the number of states corresponding to p by T_p and the number of actions corresponding to p by R_p in Sections 3.5.1 and 3.5.2, respectively. This mechanism of splitting the training data helps our proposed DDCP approach to determine the best clustering of control and data planes, as we need to go through the set of all trained DQN agents.

It is worth noting that, the global number of States and Actions is selected based on a set of parameters: i) the global number of switches in the data plane (i.e., 32 switches in our case) and the number of controllers in the control plane (i.e., 10 controllers in our case), ii) the maximum and minimum capacity of each cluster, and iii) the convergence of the Cost function, while training the set of DQN agents based on p.

During the training phase, we adopt ϵ-greedy method as action selection method. The final exploration rate is fixed at 0.2, while the *Q-target* parameters are copied from the *Q-Network* every 200 steps. The learning rate and discounted factor are fixed to 0.01 and 0.6, respectively, which correspond to η and φ parameters in equations (3) and (4). In addition, each training process corresponds to 1000 episodes. Finally, we fixed the *Threshold*, indicated in Algorithm 1, to 30% to avoid overloading the control plane, as will be justified in the next section.

4.2. Experimental Results

In order to evaluate the performance of our proposed DDCP approach, we first determine the best DQN model based on the reward function weighting factors. Then, we determine the number of controllers (i.e., clusters) p to be deployed. To do so, we compare our approach with the well-known K-means clustering method. Then, we show the benefit of our approach in term of data plane partitioning i.e., which switch assigned to which cluster. Thereafter, we evaluate its performances in terms of CD, CL, ICD and ICT metrics. Finally, we perform a comparative analysis between our proposed DDCP approach and three main schemes proposed in the literature: 1) Optimal and Dynamic Controller Placement (ODCP) [17], 2) Dynamic SDN Controller Placement in Elastic Optical Datacenter Networks (DSCP) [19], and 3) A Hierarchical K-means Algorithm for Controller Placement in SDN-based WAN Architecture (HKCP) [18].

As mentioned in equation (33), the reward function is composed of four performance metrics (i.e., CD (CD_i^t), CL (CL_i^t), ICD (ICD_i^t) and ICT (ICT_i^t)) weighted by four parameters α, β, γ and ρ, respectively. These weighting factors play an important role to determine, in one side, the importance of each performance metric, and on the other side the convergence of the Loss function, shown in equation (5). To this end, we depict, in Fig. 5, the average value of each performance metric (i.e., CD, CL, ICD, ICT) after training the DQN agent under different number of training episodes (1000 episodes in total), while changing the weighting factors according to the following strategies:

- S_1: this strategy considers only the control plane performance metrics (i.e., CD and CL): $\alpha = \frac{1}{2}$, $\beta = \frac{1}{2}$, $\gamma = 0$, $\rho = 0$.
- S_2: this strategy considers only the data plane performance metrics (i.e., ICD and ICT): $\alpha = 0$, $\beta = 0$, $\gamma = \frac{1}{2}$, $\rho = \frac{1}{2}$.
- S_3: this strategy considers delay-throughput performance metrics (i.e., CD, ICD, ICT): $\alpha = \frac{1}{3}$, $\beta = 0$, $\gamma = \frac{1}{3}$, $\rho = \frac{1}{3}$.
- S_4: this strategy gives importance to all performance metrics: $\alpha = \frac{1}{4}$, $\beta = \frac{1}{4}$, $\gamma = \frac{1}{4}$, $\rho = \frac{1}{4}$.

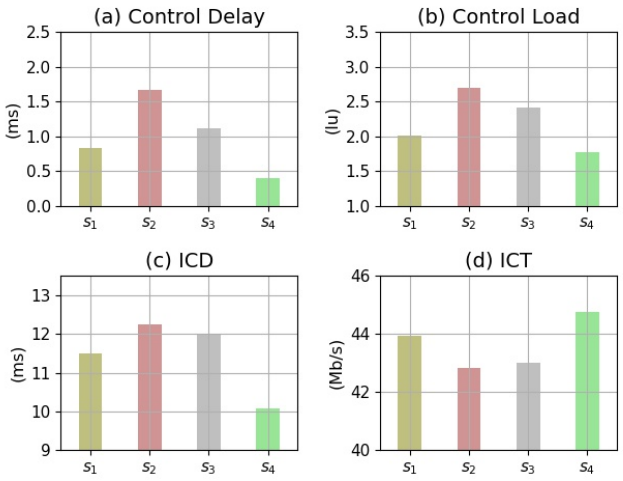

Figure 5: Impact of varying the reward function weighting factors $\alpha, \beta, \gamma, \rho$ on the CD, the CL, the ICD and the ICT metrics

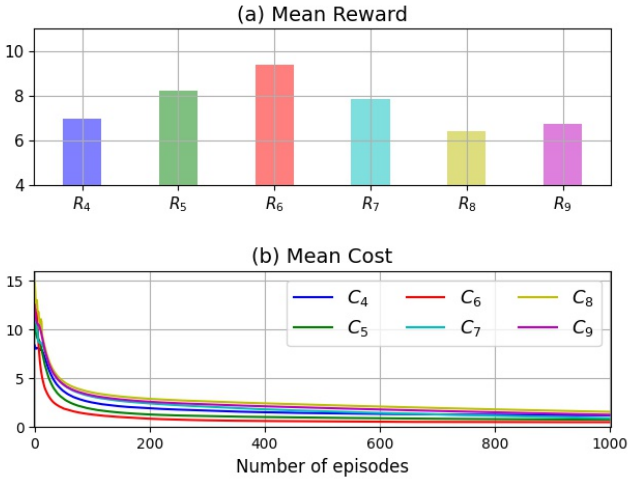

Figure 6: Impact of varying the number of clusters and the controllers placement while training the DQN agent

From Fig. 5, we can see that considering only the data plane performance metrics in strategy S_2 causes obviously high values of CD, CL and ICD, while decreasing the ICT metric. The reason is that some controllers are overloaded and experiencing congestion, while others are under-utilized. On the other hand, strategy S_3 shows better performances compared to strategy S_2, since the CD metric is taken into account. However, the control load (CL) is still high compared to the two remaining strategies (S_1 and S_4) since this metric is not taken into account in the reward function of strategy S_3. Considering the CL metric in strategy S_1 improves the performances compared to the two strategies S_2 and S_3. This shows the importance of balancing the load between the set of controllers in the control plane. Finally, strategy S_4, which takes into account all performance metrics (i.e., CD, CL, ICD and ICT), outperforms all others strategies, showing high throughput, low intra-cluster delay, low control delay, and low control load. Hence, according to these results, we adopt strategy S_4 (i.e., $\alpha = \frac{1}{4}$, $\beta = \frac{1}{4}$, $\gamma = \frac{1}{4}$, $\rho = \frac{1}{4}$) for our subsequent experiments.

Let us now determine the optimal number/range of controllers (i.e., clusters) to be deployed p. To do so, we plot in Fig. 6 the evolution of the reward and loss functions (as defined in equations (33) and (5)) under different number of training episodes and using the aforementioned weighting factors by adopting the strategy S_4. We compare in Fig. 6(a) the following baselines:

- $R_4, R_5, R_6, R_7, R_8, R_9$: where R_p corresponds to the reward under the number of clusters p, $p \in [4..9]$.

Similarly, we compare in Fig. 6(b) the following baselines:

- $C_4, C_5, C_6, C_7, C_8, C_9$: where C_p corresponds to the cost under the number of clusters p, $p \in [4..9]$.

From Fig. 6 (b), we can see that the Loss function for all studied baselines converges. On the other hand, we can observe from Fig. 6 (a) that the mean reward increases while increasing the number of clusters in the fist part of the range where $p \in [4..6]$. However, it starts to decrease in the second part of the range where $p \in [7..9]$. The increase in the first part reflects the existence of new clustering configurations (i.e., based on link latency) and controllers placement that lead to minimize the CD, the CL and the ICD metrics. The decreasing in the second part of the range of p, can be explained by the fact that the increase in the number of controllers in the control plane impacts the performances such as the CL metric. As a result, the number of controllers or clusters to be deployed, according to our DDCP approach, corresponds to that of the maximum reward, which is equal to 6 in our experiments. Next, for the sake of comparison, we take this range [4..6] for the variable p.

Let us now see the returned number of controllers p to be deployed when using the well-known K-means clustering method. Recall that K-means is widely used in network partition problems [25] and includes four main steps: 1) select p random points as cluster centers called *Centroids*, 2) assign switches to the closest cluster based on the latency of links and their locations, 3) recalculate the centroid for each cluster by computing the average of the assigned switches, 4) repeat steps 2 and 3 until none of the cluster assignments change.

To determine the number of controllers to be deployed using the K-means algorithm, we refer to the Within Cluster Sum of Squares (WCSS) method [26], which computes the distance (i.e, delay in our case) between a centroid of a cluster and each observation (i.e., switch in our case) based on which it assigns the observation to the nearest cluster. To do so, we plot, in Fig. 7, the WCSS method after training the K-means algorithm for maximum 300 iterations under different number of clusters of switches, where WCSS is determined as follows:

$$WCSS = \sum_{i=1}^{p} \sum_{j=1}^{|w_i|} (x_j - y_i)^2 \tag{34}$$

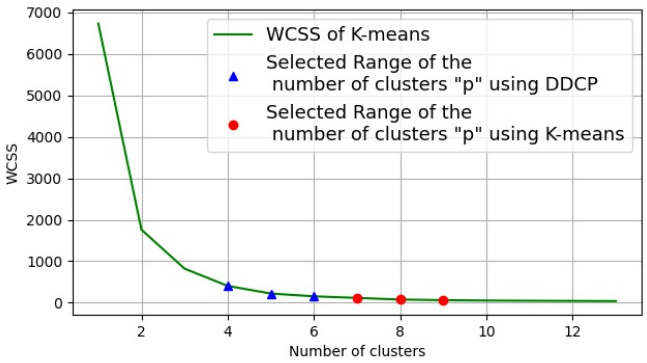

Figure 7: WCSS of K-means models under different number of clusters

$$x_j = (v_{src}, v_{dst}, A(v_{src}, v_{dst})), (v_{src}, v_{dst}) \in w_i^2 \quad (35)$$

Where x_j represents a link in the domain w_i, in which v_{src}, v_{dst} are data plane switches and $A(v_{src}, v_{dst})$ is the delay between them, the y_i denotes the centroid of the domain w_i. In this way, the WCSS method consists in clustering the set of points x_j in order to minimize the latency between switches.

From Fig. 7, we can see that the average controller response time of WCSS continues to decrease, while increasing the number of clusters, since the more we increase the number of clusters, the more we have a small number of switches in each cluster w_i, which leads to minimizing the latency. However, this can lead to overload the control plane due to the increase of the number of controllers. To this end, we select the number of clusters (i.e., controllers) when the WCSS starts to have low values and be stable. Moreover, for the sake of comparison using our approach (i.e., DDCP), we select a range of p values when the WCSS converges. This leads us to choose the range $[7, 8, 9]$ for the number of clusters p based on the K-means algorithm. This results in higher deployed clusters compared to our DDCP approach, where the identified range of p is [4..6].

To further show the benefit of our DDCP approach in term of data plane partitioning (i.e., which switch is assigned to which cluster), we first compare it with the following baseline:

- Reduced DDCP, which corresponds to our proposed approach in which the data plane clustering is determined by the K-means algorithm instead of the DQN agent, and where the number of controllers to be deployed is $p \in [4..6]$ (for the sake of comparison) and identified statically.

Fig. 8 and Fig. 9 depict the data plane devices allocated to each cluster by using the Reduced DDCP and DDCP schemes, respectively, under different values of p. It is noteworthy that, we have used the p values in the range $[4, 5, 6]$. Also, the reward function parameters of the DQN agent correspond to the strategy S_4 (i.e., $\alpha = \frac{1}{4}$, $\beta = \frac{1}{4}$, $\gamma = \frac{1}{4}$, $\rho = \frac{1}{4}$). We can see that, when the number of clusters p is equal to 4, the allocation of switches following the two schemes (i.e., Original DDCP and Reduced DDCP) in Fig. 8(a) and 9(a) is completely different, since the DDCP scheme is based on DQN that uses the previous clustering experiences, while the K-means method used in the Reduced DDCP scheme, is based on recalculating the centroid of each cluster for each step of the model training. When $p = 5$ (cf. Fig. 8(b) and Fig. 9(b)), we can observe more similarity, compared to those when $p = 4$, mostly in the last cluster. When $p = 6$ (cf. Fig. 8(c) and Fig. 9(c)) the two schemes achieve interestingly a close clustering result with a superiority of the DDCP approach in terms of average CD, CL, ICD and ICT metrics, as clearly depicted in Fig. 10. This convergence in clustering can be explained by the fact that, as the K-means considers only the clustering in the data plane, referring to the DDCP approach to determine the number of clusters improves the performances. However, it still causes some degradation comparing to the DDCP approach since the controllers are identified statically in the reduced one, which increase the CD, the CL and the ICD metrics, as clearly depicted in Fig. 10.

Finally, we perform, in the following, a comparative analysis between our proposed DDCP approach and three main approaches proposed in the literature:

- ODCP [17], which consists in using a quadratic program to solve the controller placement problem and determines the number of switches in each switch domain. After solving the controller placement problem, it dynamically migrates switches in case of controller overload.
- DSCP [19], which is based on dynamically matching the set of controllers to the set of data plane switches in order to maximize the resource utilization. Moreover, it dynamically balances the traffic load and deploys the controllers.
- HKCP [18], which is a SDN network partitioning method based on the hierarchical K-means algorithm. It considers the latency between the switches and their controllers as well as the load balancing between the set of controllers.

We used the same experimental topology, described in Section 4.1, for all approaches. In addition, to have a fair comparison, we have compared all approaches before and after switch migration. Recall that the action that triggers the switch migration is the overloading of the control plane. To this end, we used the following scenario:

We consider the set of controllers and data plane clusters obtained by using our DDCP approach from Fig. 9, where the number of clusters is 5 before switch migration. Then, we force the switches to overload the set of controllers by sending a high number of *Packet_In*. This will force the four studied approaches (DDCP, ODCP, DSCP, and HKCP) to perform the migration of some switches to new clusters and change the network topology.

First, it is interesting to see the impact of varying the Threshold defined in our DDCP approach in the switch migration procedure. To do so, we vary this parameter, denoted by Th, between 10% and 40% and depict the number of migrated switches in Fig. 11(a). Recall that CL_{mig}, defined in equation (22) and used in the switch migration procedure, refers to the

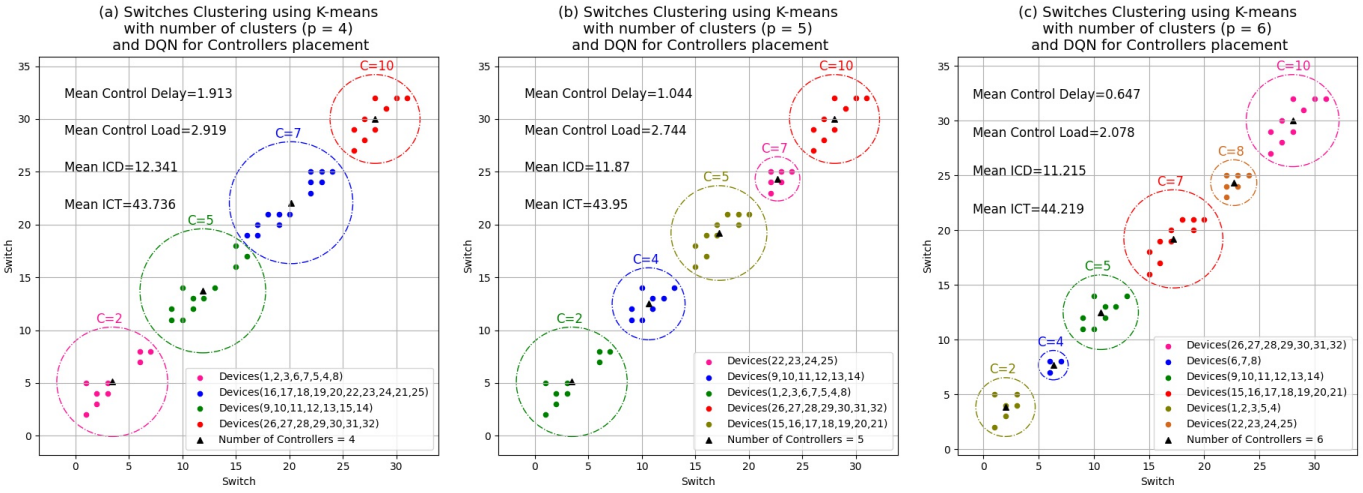

Figure 8: Demonstration of the network clustering for the Reduced DDCP scheme

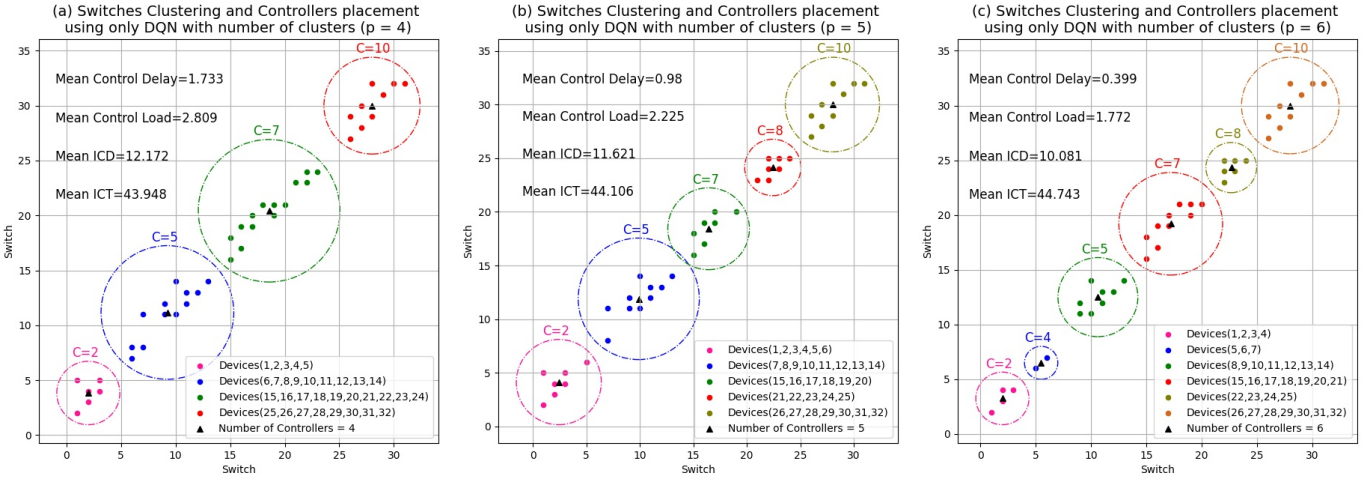

Figure 9: Demonstration of the network clustering for the DDCP scheme

load balancing factor between clusters. The more this factor is high, the more some controllers are overloaded compared to the others. In this way, when CL_{mig} exceeds the Threshold (Th), the migration process is triggered. From Fig. 11(a), we can observe that the number of migrated switches increases while decreasing Th, due to the difference in the load between the set of controllers. However, giving small values to Th may impact network performances since the migration will happen more frequently in this case, which increases the control delay, as shown in Fig. 11(b). On the other hand, when Th is high, the number of switches to migrate is low. However, clusters will be unbalanced in this case since some controllers will be overloaded compared to the others. This results in increasing again but more significantly the control delay, as shown in Fig. 11(b). A trade-off between the clusters' load and the number of switches to migrate is thus necessary. According to Fig. 11, this trade-off is achieved when the Threshold is equal to 30%. Hence, in our subsequent experiments, we fixed Th to this obtained value.

Fig. 12 shows the average resource utilization in terms of CPU and RAM before and after switch migration under all schemes. First, we can observe that, the migration process reduces considerably the average resource utilization for all schemes. The gain is more significant when using our DDCP approach, since it considers both control and data plane performance metrics (i.e., CD, CL, ICD and ICT) in the reward function when deploying a new cluster, as opposed to the other schemes. Indeed, the DDCP approach shows a decrease in CPU usage (respectively, RAM usage) of approximately 24% (respectively, 28%). Compared to ODCP, DSCP, and HKCP, these gains are reduced to 10%, 5%, and 7%, respectively, for the CPU usage. On the other hand, for the RAM usage, these gains are reduced to 10%, 7%, and 9%, respectively. However, we note here that this implies an additional deployment of a cluster (controller) in our DDCP approach since the number of clusters after migration is increased to 6 in our experiment. In contrast, the three other approaches ODCP, DSCP and HKCP keep us-

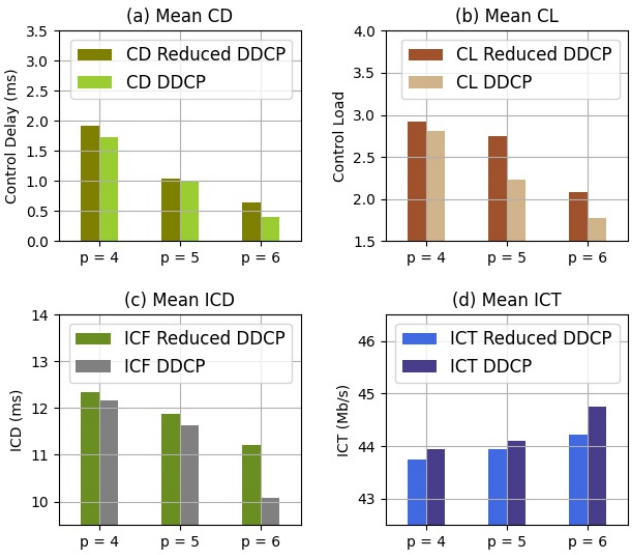

Figure 10: Comparing dynamic clustering and placement based on DQN and K-means methods

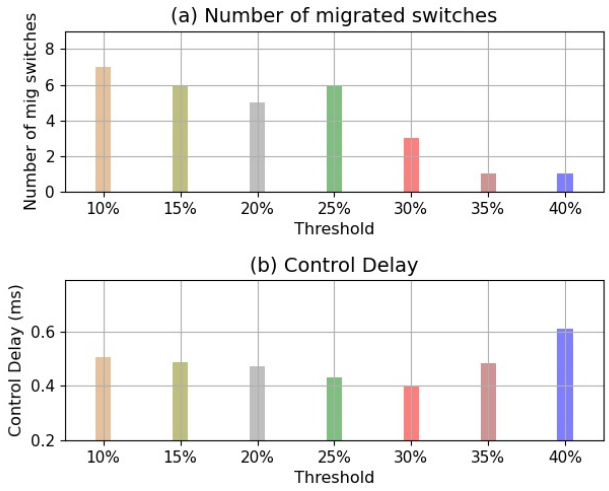

Figure 11: Impact of varying the Threshold (Th) on the number of migrated switches and control delay in the DDCP approach

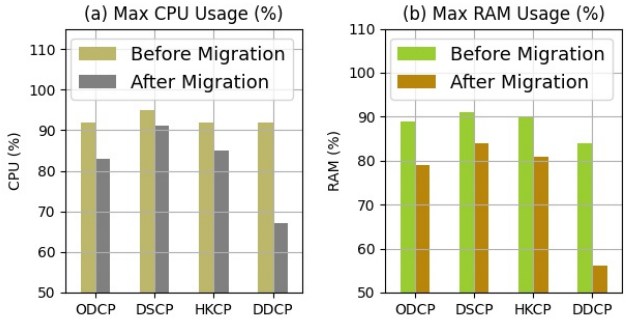

Figure 12: Average resource utilization before and after switch migration

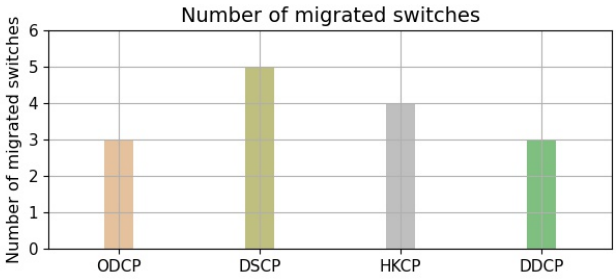

Figure 13: Comparison of number of migrated switches under ODCP, DSCP, HKCP and DDCP schemes

ing the same number of clusters already defined by the network operator.

Fig. 13 depicts the number of migrated switches for all schemes. We can see that both ODCP and DDCP reduce the number of migrated switches. On the other hand, this number is higher in the two remaining approaches (i.e. DSCP and HKCP), impacting thus the robustness of the network. In fact, having a high number of switches to migrate increases the signalling overhead, impacting thus the controllers' load and the network stability.

To further show the benefit of our DDCP approach, we plot in Fig. 14 the delay-throughput performance metrics (i.e., CD, ICD and ICT) for all schemes. We can see that the DSCP scheme increases the intra-cluster delay and decreases the throughput, since those metrics are not considered when clustering the network. On the other hand, both HKCP and ODCP schemes show better performances, as they take into account additional parameters such as the delay between the controllers. Finally, we can see that our DDCP approach outperforms all other schemes, showing higher throughput and lower delay compared to the others, thanks to the use of the DQN agent with a more complete reward function to solve the controllers' placement problem. However, this comes at the expense of an additional deployment of a cluster/controller in the control plane, as stated previously. It is worth noting that, as several controllers need to be deployed in several locations, the network will be more susceptible to different security challenges and threats.

5. Conclusion

How many controllers to use in the control plane, where to place them, which switch in the data plane must be controlled by which controller represent challenging questions in SDN. To address these important questions, we used optimization techniques to determine the optimal number of controllers, their optimal placements and the optimal clustering of data plane switches. Because the formulated optimization problem is NP-hard, a simple yet computationally efficient heuristic algorithm, called DDCP, was proposed and implemented. Our approach solution approach can be used as part of the knowledge plane to optimize control and data plane operations, by deploying a DQN agent that dynamically determines the optimal policy mapping

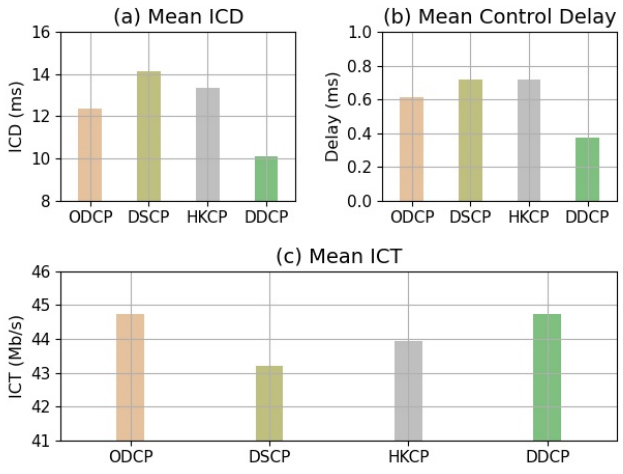

Figure 14: Delay-Throughput metrics evaluation under ODCP, DSCP, HKCP and DDCP schemes

the set of states (clusters of switches) to the set of actions (the set of controllers in specific locations). Experimental results, showed the effectiveness of our approach in identifying the appropriate number of controllers to be deployed and the clustering of data plane switches around these controllers. Moreover, our experiments showed that, the DQN agent outperforms the well known K-means clustering method as well as three main methods proposed in the literature by decreasing the control delay, the control load, and the intra-cluster delay and increasing the intra-cluster throughput. However, this comes at the expense of an additional deployment of a cluster/controller in the control plane.

Acknowledgement

This work was partially supported by the FUI SCORPION project (Grant no. 17/00464).

References

[1] B. Heller, R. Sherwood, N. McKeown, The controller placement problem, SIGCOMM Comput. Commun. Rev. 42 (4) (2012) 473–478. doi:10.1145/2377677.2377767.
URL https://doi.org/10.1145/2377677.2377767

[2] A. Curtis, J. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, Devoflow: Scaling flow management for high-performance networks, Vol. 41, 2011, pp. 254–265. doi:10.1145/2018436.2018466.

[3] Y. Jimenez, C. Cervelló-Pastor, A. Garcia, On the controller placement for designing a distributed sdn control layer, 2014, pp. 1–9. doi:10.1109/IFIPNetworking.2014.6857117.

[4] A. Tootoonchian, Y. Ganjali, Hyperflow: A distributed control plane for openflow, 2010, pp. 3–3.

[5] 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.

[6] T. Das, V. Sridharan, M. Gurusamy, A survey on controller placement in sdn, IEEE Communications Surveys Tutorials 22 (1) (2020) 472–503.

[7] G. Wang, Y. Zhao, J. Huang, Q. Duan, J. Li, A k-means-based network partition algorithm for controller placement in software defined network, in: 2016 IEEE International Conference on Communications (ICC), 2016, pp. 1–6.

[8] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, M. Hoffmann, Heuristic approaches to the controller placement problem in large scale sdn networks, IEEE Transactions on Network and Service Management 12 (1) (2015) 4–17.

[9] G. Yao, J. Bi, Y. Li, L. Guo, On the capacitated controller placement problem in software defined networks, IEEE Communications Letters 18 (8) (2014) 1339–1342.

[10] P. Tao, C. Ying, Z. Sun, S. Tan, P. Wang, Z. Sun, The controller placement of software-defined networks based on minimum delay and load balancing, in: 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 2018, pp. 310–313.

[11] Y. Hu, T. Luo, N. C. Beaulieu, C. Deng, The energy-aware controller placement problem in software defined networks, IEEE Communications Letters 21 (4) (2017) 741–744.

[12] L. Mamushiane, J. Mwangama, A. A. Lysko, Given a sdn topology, how many controllers are needed and where should they go?, in: 2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2018, pp. 1–6.

[13] M. Tanha, D. Sajjadi, J. Pan, Enduring node failures through resilient controller placement for software defined networks, in: 2016 IEEE Global Communications Conference (GLOBECOM), 2016, pp. 1–7.

[14] T. Y. Cheng, M. Wang, X. Jia, Qos-guaranteed controller placement in sdn, in: 2015 IEEE Global Communications Conference (GLOBECOM), IEEE, 2015, pp. 1–6.

[15] P. T. A. Quang, Y. Hadjadj-Aoul, A. Outtagarts, A deep reinforcement learning approach for vnf forwarding graph embedding, IEEE Transactions on Network and Service Management 16 (4) (2019) 1318–1331.

[16] H. Mostafaei, M. Menth, M. S. Obaidat, A learning automaton-based controller placement algorithm for software-defined networks, in: 2018 IEEE Global Communications Conference (GLOBECOM), 2018, pp. 1–6.

[17] N. Mouawad, R. Naja, S. Tohme, Optimal and dynamic sdn controller placement, in: 2018 International Conference on Computer and Applications (ICCA), 2018, pp. 1–9. doi:10.1109/COMAPP.2018.8460361.

[18] H. Kuang, Y. Qiu, R. Li, X. Liu, A hierarchical k-means algorithm for controller placement in sdn-based wan architecture, in: 2018 10th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA), 2018, pp. 263–267. doi:10.1109/ICMTMA.2018.00070.

[19] Y. Liu, H. Gu, X. Yu, J. Zhou, Dynamic sdn controller placement in elastic optical datacenter networks, in: 2018 Asia Communications and Photonics Conference (ACP), 2018, pp. 1–3. doi:10.1109/ACP.2018.8596219.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning (12 2013).

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (2015) 529–33. doi:10.1038/nature14236.

[22] J. Hyun, J. W. Hong, Knowledge-defined networking using in-band network telemetry, in: 2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS), 2017, pp. 54–57.

[23] J. Little, S. Graves, Little's Law, 2008, pp. 81–100. doi:10.1007/978-0-387-73699-0_5.

[24] E. H. Bouzidi, A. Outtagarts, R. Langar, Deep reinforcement learning application for network latency management in software defined networks, in: 2019 IEEE Global Communications Conference (GLOBECOM), 2019, pp. 1–6. doi:10.1109/GLOBECOM38437.2019.9013221.

[25] H. Al-Mohair, J. Mohamad-Saleh, S. A. Suandi, Hybrid human skin detection using neural network and k-means clustering technique, Applied Soft Computing 33 (05 2015). doi:10.1016/j.asoc.2015.04.046.

[26] N. U. Roiha, Y. K. Suprapto, A. D. Wibawa, The optimization of the weblog central cluster using the genetic k-means algorithm, in: 2016 International Seminar on Application for Technology of Information and Communication (ISemantic), 2016, pp. 278–284.

[27] E. H. Bouzidi, D. Luong, A. Outtagarts, A. Hebbar, R. Langar, Online-based learning for predictive network latency in software-defined networks, in: 2018 IEEE Global Communications Conference (GLOBECOM), 2018, pp. 1–6. doi:10.1109/GLOCOM.2018.8648063.

EL Hocine Bouzidi

received the engineering degree in computer science from the National School of Computer Science of Algiers (ESI), in 2010, and the M.Sc. degree in computer science in safe embedded and mobile systems from le CNAM, France, in 2016. He received Ph.D. degree in telecom engineering from Nokia Bell Labs and Gustave Eiffel University, France, in 2021. He joined National School of Computer Science of Algiers (ESI), as a System and Network Engineer, from 2012 to 2015. He joined Orange Lab, France, for his master's degree internship, as a software developer, in 2016. He joined Nokia, France, as cloud packet core engineer, from 2021. His research interests include B5G networks, networks resource management, SDN and NFV.

Abdelkader Outtagarts

IEEE senior member is a senior researcher and technical leader in NFV and SDN orchestration, machine learning and automation in Nokia Bell Labs. Abdelkader received the MSc and Ph.D. degrees from the INSA de Lyon in France, in 1990 and 1994 respectively, on automation of energy systems. In 1999, he receives a MSc on software engineering from Ecole de Technologie Supérieure of Montreal in Canada. His professional experience of over 20 years, on information systems and telecommunications, energy efficiency, software engineering, cloud computing, data mining, NFV micro services and SDN orchestration and automation, is mainly acquired in France and Canada in R&D teams (Alcatel-Lucent, Nextenso, Hydro-Quebec, UTILICASE and SCII Technology), in research laboratories in Lyon (INSA), Montreal (ETS, CRIM) and Nozay (Nokia Bell labs).

Rami Langar

(Member, IEEE) received the M.Sc. degree in network and computer science from University Pierre and Marie Curie (now Sorbonne University) in 2002, and the Ph.D. degree in network and computer science from Telecom ParisTech, Paris, France, in 2006. He was a Post-Doctoral Research Fellow with the School of Computer Science, University of Waterloo, Waterloo, ON, Canada, from 2006 to 2008, and an Associate Professor with LIP6, University Pierre and Marie Curie, from 2008 to 2016. He is currently a Full Professor affiliated with University Gustave Eiffel (France) and ÉTS-Montréal (Canada). He is involved in many European and National French research projects, such as ANR 5G-INSIGHT, ANR ABCD, FUI SCORPION, FUI ELASTIC Networks, FUI PODIUM, MobileCloud (FP7), GOLDFISH (FP7), etc. His research interests include resource management in future wireless systems, cloud-RAN, network slicing in 5G/5G+/6G, software-defined wireless networks, mobile cloud offloading, and green networking. He was a co-recipient of the Best Paper Award from the IEEE/IFIP International Conference on Network and Service Management 2014 (IEEE/IFIP CNSM 2014). He was the Chair of the IEEE ComSoc Technical Committee on Information Infrastructure and Networking (TCIIN) from January 2018 to December 2019.

Raouf Boutaba

(M'93–SM'01–F'12) received the M.Sc. and Ph.D. degrees in computer science from the Pierre and Marie Curie University, Paris, France, in 1990 and 1994, respectively. He is currently a Professor of computer science with the University of Waterloo, Waterloo, ON, Canada. His research interests include resource and service management in networks and distributed systems. Dr. Boutaba is a Fellow of the IEEE, the Engineering Institute of Canada, and the Canadian Academy of Engineering. He served as a Distinguished Speaker for the IEEE Computer and Communications Societies. He is the founding Editor-in-Chief of the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT (2007–2010), and he is on the editorial boards of other journals. He was the recipient of several Best Paper Awards and other recognitions, such as the Premier's Research Excellence Award, the IEEE Hal Sobol Award in 2007, the Fred W. Ellersick Prize in 2008, the Joe LociCero Award and the Dan Stokesbury Award in 2009, the Salah Aidarous Award in 2012, and the McNaughton Gold Medal in 2014.

Elsevier Computer Networks

CRediT author statement for **manuscript COMNET-D-21-00760 entitled:**
"Dynamic Clustering of Software Defined Network Switches and Controller placement using Deep Reinforcement Learning"

EL Hocine Bouzidi: Conceptualization, Methodology, Investigation, Software development and integration, Writing- Original draft preparation.

Abdelkader Outtagarts: Data curation, Software, Supervision, Validation, Reviewing and Editing.

Rami Langar: Supervision, Validation, Writing, Reviewing and Editing.

Raouf Boutaba: Validation, Reviewing and Editing.

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Kind regards,
EL Hocine BOUZIDI, Abdelkader Outtagarts, Rami Langar and Raouf Boutaba