# Load Balancing in Large WiFi Networks Using DQL-MultiMDP with Constrained Clustering

Mohamed Bellouch, Lynda Zitoune, Iyad Lahsen-Cherif, Véronique Vèque

# Load Balancing in Large WiFi Networks Using DQL-MultiMDP with Constrained Clustering

Mohamed Bellouch*, Lynda Zitoune*, Iyad Lahsen-Cherif†, Véronique Vèque*
*Université Paris-Saclay, CNRS, CentraleSupélec, Laboratoire des Signaux et Systèmes
3, rue Joliot Curie, 91190 Gif-sur-Yvette, France.
Email: {mohamed.bellouch, lynda.zitoune, veronique.veque}@l2s.centralesupelec.fr
†INPT, CS department, Rabat, Morocco.
Email: lahsencherif@inpt.ac.ma

***Abstract*—Developing efficient load-balancing techniques remains a persistent research challenge as modern WiFi networks evolve into increasingly complex environments, incorporating new enhancements in their standards. For instance, DQL-MultiMDP is a load-balancing algorithm that learns an optimal STA-to-AP association policy to ensure user fairness and optimize network performance in dense and dynamic WiFi networks. The algorithm leverages a Multi-Markov Decision Process (Multi-MDP) strategy to accommodate the fluctuating number of devices caused by their switching on/off. However, scalability challenges arise due to the exponential expansion of the action space. In this paper, we propose a divide-and-conquer approach that extends the algorithm to operate in extremely large deployments: a dynamic partitioning mechanism divides the network into partitions and assigns a sub-controller to manage the STA-to-AP association in each partition independently, and a coordination mechanism enables them to exchange their training updates. Experimental investigations validate the effectiveness of the approach and motivate future work.**

***Index Terms*—WiFi, Load balancing, Deep Q-Learning, Network partitioning**

## I. Introduction

Modern WiFi networks are expanding to accommodate the Internet of Things (IoT) and offer new human-oriented services. Stations (STAs) range from connected sensors to devices running intensive applications like Augmented/Virtual Reality (AR/VR), gaming, or 8K videos. Some devices are attached to users and exhibit complex mobility. Additionally, Access Points (APs) can be managed by energy efficiency mechanisms that turn them on or off depending on traffic and demand dynamics [1], [2], mirroring the unpredictable connection and disconnection behavior of the STAs. Consequently, the deployment of modern WiFi networks is characterized by its dense and dynamic nature, making the management and optimization of Quality of Service (QoS) for diverse applications very challenging. In dense networks, APs are very close to each other, leading to overlapping cells and heightened interference levels. The upcoming IEEE 802.11 standards aim to improve existing WiFi networks to address these circumstances and ensure seamless communication. The IEEE 802.11be (WiFi 7) promises to deliver Extremely High Throughput (EHT) and reduced latency [3]. It introduces several novel enhancements, such as wider bandwidths and multi-link operation [4], to ensure that STAs can seamlessly switch between different APs, thereby providing robust support for load-balancing techniques. However, a primary concern lies in the default association decision policy, where each STA automatically connects to the AP with the highest received signal strength indicator (RSSI). This approach can lead to AP overloading, unfair distribution of resources among users, and a degradation in overall network performance. Although efficient load-balancing has attracted significant attention in this area of research, it remains outside the scope of the IEEE 802.11ax/be standards, with STA-to-AP association decisions being made by vendor-specific algorithms [3], [5].

DQL-MultiMDP [6] is an algorithm for load balancing in dense and dynamic WiFi networks. Leveraging Deep Q-Learning (DQL), it learns through autonomous decision-making an optimal STA-to-AP association policy that ensures long-term fairness among users, balances the load on the APs, and optimizes the overall network performance. Notably, the algorithm leverages a Multi-MDP strategy to overcome the problem of the ever-changing number of devices due to their switching on/off and to provide a fine-grained knowledge of the environment's dynamics. However, the algorithm is typically designed to operate in a fully dense setting where each STA falls in the range of all the available APs. This provokes an exponential growth of the action space and leads to poor learning performance as the number of devices increases, making the algorithm suitable only under a limited *controller's capacity*, i.e., the maximum number of APs and maximum number of STAs that the controller that hosts the algorithm can handle while operating under good conditions.

This paper presents a practical divide-and-conquer approach to extend the algorithm to operate in extremely large WiFi networks. The idea is based on dynamically partitioning the network and assigning a DQL-MultiMDP sub-controller to manage the user association in each partition independently. First, we propose a primary version of a network partitioning mechanism that assumes knowledge of the spatial coordinates of the APs. Then, we design a coordination scheme that allows the sub-controllers to benefit from each other's training updates; we design a strategy that keeps the "best" updates in a central database so that the sub-controllers use them instead of starting training with random parameters.

In the remaining, we review related work on load balancing

and network partitioning in Sec. II. Next, we present our system model in Sec. III, along with a brief background on DQL-MultiMDP and the reason behind its limited *controller's capacity*. We introduce the proposed dynamic partitioning mechanism in Sec. IV. Sec. V presents an abstract architecture of the proposed system alongside the proposed strategy to allow the sub-controllers to exploit each other's training updates. Sec. VI provides simulation results and investigations to validate the effectiveness of our approach. Finally, we conclude the paper in Sec. VII with insights for future work.

## II. Related Work

The authors of [7] provided a comprehensive taxonomy of the "old-fashioned" load-balancing techniques for WLANs. These techniques are considered to be "old-fashioned" because they were developed for the early WiFi standards, and their association decisions are primarily based on simplistic metrics like the RSSI values, users' priorities, and load on the APs, under some strong assumptions on the system.

New solutions were proposed with the emergence of reinforcement learning (RL) and deep RL (DRL) techniques, promising abilities in terms of adaptability and flexibility. For example, [8] proposed an online Q-Learning AP assignment algorithm that aims to ensure user satisfaction and optimize throughput in an SDN-controlled network. Ahmed et al. [9] worked particularly on hybrid WiFi/LiFi networks. However, the models upon which these solutions are built do not often consider the dynamic nature of these networks, such as user mobility and the fluctuating number of devices, and they are validated on simulation settings with fixed devices with few numbers. DQL-MultiMDP [6] addresses these challenges by leveraging the Multi-MDP approach. However, in all these techniques, the scalability limitation remains an ongoing challenge due to the expansion of the action space.

On the other hand, the network partitioning literature is primarily motivated by the Controller Placement Problem [10]. [11] survey of network partitioning techniques used to address this problem in Software Defined Networks (SDNs). The proposed works often preserve the same graph modeling of the network and use algorithms like $k$-center, capacitated $k$-center (that, similar to our work, has the ability to guarantee a size-balanced partitioning, but on graphs), and graph clustering techniques like Affinity Propagation. To the best of our knowledge, this is the first work that addresses the scalability limitation of a DRL-based technique through network partitioning and probably the first one that uses the Constrained $k$-means Clustering algorithm for this purpose.

## III. System model, assumptions, and background

The controlled WiFi network at time $t \in \mathbb{R}$ consists of a finite set of $N_{STA}^{(t)}$ active STAs, and a finite set of $N_{AP}^{(t)}$ active APs, denoted by $\mathcal{STA}^{(t)} = \{\mu_i\}_{i\in\left[N_{STA}^{(t)}\right]}$ and $\mathcal{AP}^{(t)} = \{\alpha_j\}_{i\in\left[N_{AP}^{(t)}\right]}$. The sets are time-dependent as the number of devices changes due to their switching on/off, and we assume that they both remain non-empty at any $t$. For a non-null integer $n$, the notation $[n] = \{1, ..., n\}$ is used throughout the paper. The current STA-to-AP associations at $t$ are modeled by a binary matrix $C^{(t)} = \left(c_{\mu,\alpha}^{(t)}\right)_{(\mu,\alpha)\in\mathcal{STA}^{(t)}\times\mathcal{AP}^{(t)}}$, such that if the STA $\mu \in \mathcal{STA}^{(t)}$ is associated with the AP $\alpha \in \mathcal{AP}^{(t)}$ at $t$, we have $c_{\mu,\alpha}^{(t)} = 1$, otherwise $c_{\mu,\alpha}^{(t)} = 0$. We denote by $RSSI_{\mu,\alpha}^{(t)}$ and $\rho_{\mu,\alpha}^{(t)}$ the RSSI value and the achievable data rate of $\mu$ from $\alpha$ at $t$, respectively. We denote the required data rate by $\mu$ at $t$ as $\sigma_{\mu}^{(t)}$. We assume that the APs' locations are known or can be approximated at a certain degree of precision; we denote the position of an AP $\alpha$ as $p_\alpha = (x_\alpha, y_\alpha, z_\alpha) \in \mathbb{R}^3$. To model the on/off behavior of each device $\beta$ (AP or STA), we assign to it an independent Markov chain process $S_\beta = S_\beta^{(t_0)}, S_\beta^{(t_0+\eta)}, S_\beta^{(t_0+2\eta)}, \ldots$, where the state set is {"ON", "OFF"}, and the index set is $\{t_0 + k\eta' : k \in \mathbb{N}\}$, with $t_0$ being the initial time and $\eta' > 0$ a time step.

### A. Basic DQL-MultiMDP at a glance

The primary version of DQL-MultiMDP operates in a central controller with a global view of the entire network [6]. A pair $(N_{STA}, N_{AP})$ is initially defined and called the *controller's capacity*. If $2 \leq N_{STA}^{(t)} \leq N_{STA}$ and $2 \leq N_{AP}^{(t)} \leq N_{AP}$, the association problem is seen as a MDP $\mathcal{MDP}_{N_{STA}^{(t)},N_{AP}^{(t)}} = (\mathcal{S}_{N_{STA}^{(t)},N_{AP}^{(t)}}, \mathcal{A}_{N_{STA}^{(t)},N_{AP}^{(t)}}, \mathcal{P}_{N_{STA}^{(t)},N_{AP}^{(t)}}, r_t, \gamma)$, where the elements are the state space, the action space, the state transition function, the reward function, and a discount factor $\gamma \in (0, 1)$, respectively; otherwise, the algorithm employs a traditional max-RSSI association strategy.

At $t$, the observed state $s_t$ is a vector of size $N_{STA}^{(t)}N_{AP}^{(t)} + N_{AP}^{(t)} + N_{STA}^{(t)}$ that contains the SINR value at each STA from each AP, the load on the APs, i.e, the number of STAs currently connected to each AP, and the STAs' required data rates. The applied action is a binary matrix $a_t = \left(a_{\mu,\alpha}^{(t)}\right)_{(\mu,\alpha)\in\mathcal{STA}^{(t)}\times\mathcal{AP}^{(t)}}$ that represents the controller's association decision; if the controller associates $\mu$ to $\alpha$, then $a_{\mu,\alpha}^{(t)} = 1$, otherwise $a_{\mu,\alpha}^{(t)} = 0$. As associating a single STA to multiple APs requires further analysis of WiFi's new amendments [3], basic DQL-MultiMDP associates each STA to one and only one AP. Hence, the elements in each row of the matrix $a_t$ are all zeros except one that is 1. Therefore, if the realization of each action is seen as a map form $\mathcal{AP}^{(t)}$ to $\mathcal{STA}^{(t)}$, the action space cardinal is the number of such possible maps. Thus

$$\left|\mathcal{A}_{N_{STA}^{(t)},N_{AP}^{(t)}}\right| = \left|\mathcal{AP}^{(t)}\right|^{\left|\mathcal{STA}^{(t)}\right|} = \left(N_{AP}^{(t)}\right)^{N_{STA}^{(t)}}. \quad (1)$$

By letting $l_\alpha^{(t)} = \sum_{\mu\in\mathcal{STA}^{(t)}} c_{\mu,\alpha}^{(t)}$, the load on the AP $\alpha$, the received reward at $t$ is expressed as

$$r_t = \sum_{\mu\in\mathcal{STA}^{(t)}} \sum_{\alpha\in\mathcal{AP}^{(t)}} \frac{c_{\mu,\alpha}^{(t)}\rho_{\mu,\alpha}^{(t)}}{d_\mu^{(t)} l_\alpha^{(t)}}, \quad (2)$$

so that maximizing the sum of $\gamma$-discounted future rewards implies finding the optimal association strategy that maximizes

the ratio between the Shannon capacities and the required data rates while minimizing the load on the APs. The controller stores a Q-Network for all the possible $(N_{STA}-1)(N_{AP}-1)$ MDPs. For any $2 \leq n \leq N_{STA}$ and $2 \leq m \leq N_{AP}$, the Q-Network that corresponds to $\mathcal{MDP}_{n,m}$ is represented by its parameters vector $\theta_{n,m}$. It outputs $(Q(s_t, a; \theta_{n,m}))_{a \in \mathcal{A}_{n,m}}$ the approximated Q-values of all the available actions in $\mathcal{A}_{n,m}$ given the current state $s_t$ taken as input. Then, the Q-Network's output layer size is $|\mathcal{A}_{n,m}|$.

### B. Restricted controller's capacity

The reason behind imposing the *controller's capacity* $(N_{STA}, N_{AP})$ is attributed to the exponential growth of the action space. When $N_{\text{STA}}^{(t)}$ and $N_{\text{AP}}^{(t)}$ increase, the number of available actions for the agent increases exponentially; thus, the agent needs more interactions to explore the large action space. Additionally, as the corresponding Q-Network's output layer has the same size as the action space, the number of parameters grows with it, increasing the number of required experience samples (interactions) to train it. We refer to this as an *interaction cost*. On the other hand, the search for the greedy action $\arg\max_{a \in \mathcal{A}_{n,m}} Q(s, a; \theta_{n,m})$ is performed in each learning step in the DQL algorithm to compute the Q-Learning target value. This maximization necessitates a sweep over the action space and thus has a complexity of $\mathcal{O}(|\mathcal{A}_{n,m}|) = \mathcal{O}(n^m)$. Moreover, as the number of parameters grows, the time spent in the Q-Network's forward and backward passes also increases. Hence, the action space size also impacts the duration of the learning episodes; we call this a *temporal cost*.

## IV. Network partitioning for scaling up DQL-MultiMDP

Our divide-and-conquer approach to extending DQL-MultiMDP to extremely large WiFi networks consists of partitioning the network and managing the STA-to-AP association within each partition using a DQL-MultiMDP agent that operates within a corresponding sub-controller. We first present our dynamic network partitioning mechanism, and then we describe a coordination mechanism that permits the agents "to benefit from the knowledge of others" in Sec. V.

### A. Partitioning the set $\mathcal{AP}^{(t)}$

A partitioning mechanism divides the set of APs into $k^{(t)} > 0$ disjoint partitions $\varsigma_1^{(t)}, \ldots, \varsigma_{k^{(t)}}^{(t)}$, each of which contains at least $\tau^{(t)} \geq 2$ elements, as formulated in Eqs. 3 & 4.

$$\mathcal{AP}^{(t)} = \bigcup_{i=1}^{k^{(t)}} \varsigma_i^{(t)} \text{ s.t. } \varsigma_i^{(t)} \cap \varsigma_j^{(t)} = \emptyset \; \forall i, j \in \left[k^{(t)}\right], i \neq j. \quad (3)$$

$$\tau^{(t)} \leq \left|\varsigma_i^{(t)}\right| \leq N_{AP}. \quad (4)$$

Nevertheless, this necessitates the condition

$$2 \leq k^{(t)} \tau^{(t)} \leq N_{AP}^{(t)}, \quad (5)$$

that we will consider as an assumption in what follows.

The number of clusters $k^{(t)}$ and the minimal size constraint $\tau^{(t)}$ are time-dependent since the partitioned set $\mathcal{AP}^{(t)}$ also varies due to the APs switching on/off. Eq. 4 ensures two things: all the clusters contain at least 2 APs, and no cluster exceeds the *controller's capacity*. Hence, the partitioning mechanism ensures a first condition of performing load balancing by DQL-MultiMDP in terms of the number of APs in each cluster. Moreover, as a lower bound in Eq. 4, $\tau^{(t)}$ can be used to ensure a certain balance in the clusters' sizes.

As a first step in the design of the partitioning mechanism, for a given $k^{(t)}$ and $\tau^{(t)}$, we first define $\Omega^{(t)}_{k^{(t)},\tau^{(t)}}$, the set of such possible partitions with omitting the partitions' maximum size constraint, since we will next show how it can be ensured through an adequate choice of $k^{(t)}$ and $\tau^{(t)}$. Hence

$$\Omega^{(t)}_{k^{(t)},\tau^{(t)}} = \left\{ \left(\varsigma_i^{(t)}\right)_{i \in \left[k^{(t)}\right]} : \bigcup_{i=1}^{k^{(t)}} \vartheta_i^{(t)} = \mathcal{AP}^{(t)}, \right.$$
$$\left. \forall i, j \in \left[k^{(t)}\right], \tau^{(t)} \leq \left|\varsigma_i^{(t)}\right|, \varsigma_i^{(t)} \cap \varsigma_j^{(t)} = \emptyset \text{ if } i \neq j \right\}. \quad (6)$$

To ensure seamless roaming, the APs in each partition should be optimally close to each other geographically. This yields the following optimization problem:

$$\underset{\left(\varsigma_i^{(t)}\right)_{i \in \left[k^{(t)}\right]} \in \Omega^{(t)}_{k^{(t)},\tau^{(t)}}}{\arg\min} \sum_{i=1}^{k^{(t)}} \sum_{\alpha, \alpha' \in \varsigma_i^{(t)}} \|p_\alpha - p_{\alpha'}\|_2^2. \quad (7)$$

that aims to minimize the pairwise squared deviations of APs' positions in each partition. By letting $\nu_i^{(t)} = \frac{1}{\left|\varsigma_i^{(t)}\right|} \sum_{\alpha \in \varsigma_i^{(t)}} p_\alpha$, i.e. the $i$-th partition's centroid, and based on the identity $\left|\varsigma_i^{(t)}\right| \sum_{\alpha \in \varsigma_i^{(t)}} \left\|p_\alpha - \nu_i^{(t)}\right\|_2^2 = \frac{1}{2} \sum_{\alpha, \alpha' \in \varsigma_i^{(t)}} \|p_\alpha - p_{\alpha'}\|_2^2$, the problem is equivalent to

$$\underset{\left(\varsigma_i^{(t)}\right)_{i \in \left[k^{(t)}\right]} \in \Omega^{(t)}_{k^{(t)},\tau^{(t)}}}{\arg\min} \sum_{i=1}^{k^{(t)}} \sum_{\alpha \in \varsigma_i^{(t)}} \left\|p_\alpha - \nu_i^{(t)}\right\|_2^2. \quad (8)$$

This is typically equivalent to the traditional $k$-means clustering problem if $\tau^{(t)} = 0$. However, $k$-means clustering is sensitive to outliers, i.e., if an AP is far from others, it will form a cluster that contains only that AP. In addition, there are no guarantees that the size of some clusters lies under the *controller's capacity*.

A partitioning solution from $\Omega^{(t)}_{k^{(t)},\tau^{(t)}}$ can be represented by a binary matrix $G = (g_{i,j})_{(i,j) \in \left[N_{AP}^{(t)}\right] \times \left[k^{(t)}\right]}$, such that for all $(i, j) \in \left[N_{AP}^{(t)}\right] \times \left[k^{(t)}\right]$, if the $i$-th AP $\alpha_i$ belongs to the $j$-th partition $\varsigma_j^{(t)}$, we have $g_{i,j} = 1$; otherwise $g_{i,j} = 0$. Thus, to respect the partition's minimum size constraint, the sum of the elements at the $i$-th column of the matrix is less than $\tau^{(t)}$, and the $i$-th raw contains only one element that is equal to 1, which implies that the sum of the elements at the $i$-th raw is equal to 1. We consider the set where the matrices are not necessarily binary

$$\Gamma^{(t)}_{k^{(t)},\tau^{(t)}} = \Bigg\{ G \in \mathbb{R}^{k^{(t)} N^{(t)}_{AP}} : \sum\nolimits_{h=1}^{k^{(t)}} g_{i,h} = 1 \text{ and } \sum\nolimits_{h=1}^{N^{(t)}_{AP}} g_{h,j} \geq \tau^{(t)}, \forall (i,j) \in \left[N^{(t)}_{STA}\right] \times \left[k^{(t)}\right] \Bigg\}. \tag{9}$$

For $\mathscr{C}_j(G) = \sum_{i=1}^{N^{(t)}_{AP}} g_{i,j} p_{\alpha_i} / \sum_{i=1}^{N^{(t)}_{AP}} g_{i,j}$, the authors of [12] proposed an iterative algorithm, called Constrained $k$-means Clustering, that solves the following problem

$$\underset{G \in \Gamma^{(t)}_{k^{(t)},\tau^{(t)}}}{\arg\min} \sum_{i=1}^{N^{(t)}_{AP}} \sum_{j=1}^{k^{(t)}} g_{i,j} \left\| p_{\alpha_i} - \mathscr{C}_j(G) \right\|_2^2. \tag{10}$$

They proved that the algorithm terminates in a finite time and that there exists an optimal solution $G^*$ that satisfies $g^*_{i,j} \in \{0,1\}$. By looking at $g^*_{i,j}$ as an indicator of the membership of the $i$-th AP to the $j$-th partition, i.e., $g^*_{i,j} = 1_{\alpha_i \in \varsigma^{(t)}_j}$, we can observe that $\mathscr{C}_j(G^*) = \nu^{(t)}_j$ and that an optimal solution of the problem in Eq. 8 (so as Eq. 7) can be constructed by forming the partitions as follows $\varsigma^{(t)}_i = \left\{ \alpha_j \in \mathcal{AP}^{(t)} : j \in \left[N^{(t)}_{AP}\right] \text{ s. t. } g^*_{j,i} = 1 \right\}$ for all $i \in \left[k^{(t)}\right]$.

By now, the formed partitions respect only the lower-bound constraint in Eq. 4, i.e., $\tau^{(t)} \leq \left|\varsigma^{(t)}_i\right|$ for all $i \in \left[k^{(t)}\right]$. The upper-bound constraint $\left|\varsigma^{(t)}_i\right| \leq N^{(t)}_{AP}$ will be ensured through an adequate choice of $k^{(t)}$ and $\tau^{(t)}$ as we show in the following.

### B. Bounding the choice of $k^{(t)}$ and $\tau^{(t)}$

For a given network, $k^{(t)}$ and $\tau^{(t)}$ are interdependent, and as we will show experimentally, their choice plays a crucial role in the learning performance. Thus, a strategy for their selection is required. We first begin by extracting their dependency and bounding their choice. From Eqs. 3, 4, and 5, we have

$$N^{(t)}_{AP} = \sum_{i=1}^{k^{(t)}} \left|\varsigma^{(t)}_i\right| \leq k^{(t)} N_{AP} \text{ and } k^{(t)} \leq \frac{N^{(t)}_{AP}}{\tau^{(t)}}. \tag{11}$$

Thus, as $k^{(t)}$ is an integer, we can show that

$$1 + \left\lfloor N^{(t)}_{AP} / N_{AP} \right\rfloor \leq k^{(t)} \leq \left\lfloor N^{(t)}_{AP} / \tau^{(t)} \right\rfloor. \tag{12}$$

Moreover, since $\tau^{(t)} \geq 2$, and based on Eq. 5, we get

$$2 \leq \tau^{(t)} \leq \left\lfloor N^{(t)}_{AP} / k^{(t)} \right\rfloor. \tag{13}$$

### C. Sub-controllers assignment

The clustering algorithm operates within a global controller, which is assumed to have knowledge of the estimated positions and activity of the APs at any $t$. When an AP is turned on or off at $t$, the mechanism will be triggered to select a pair $(k^{(t)}, \tau^{(t)})$ and perform a new partitioning using the Constrained $k$-means Clustering algorithm. Then, the APs in each partition $\varsigma^{(.)}_i$ will be grouped to provide seamless roaming as they are geographically close to each other. A sub-controller will be assigned to $\varsigma^{(.)}_i$ to host a DQL-MultiMDP agent. The sub-controller can be "physical" by selecting an AP in $\varsigma^{(t)}_i$ to act as a master, while the others act as slaves (under the condition that they are within the master's range). Otherwise, the sub-controller can be "virtual" if the network is managed by a central controller (via SDN), to launch a dedicated thread or process that executes the DQL-MultiMDP algorithm. The technical details, however, are omitted in this paper as our focus is on the validation of the partitioning approach. In both cases, we will abstractly look at $\varsigma^{(t)}_i$'s sub-controller as a process referred to as $\mathsf{Process}_i$.

### D. Partitioning the set $\mathcal{STA}^{(t)}$

The set of STAs will similarly be partitioned into $k^{(t)}$ clusters $\vartheta^{(t)}_1, \ldots, \vartheta^{(t)}_{k^{(t)}}$. The $\mathcal{STA}^{(t)}$ partitioning is not constrained and is done by the STAs themselves instead of the controller. When an STA powers on, it scans the available SSIDs, and it connects to the one that contains the closest AP to it. Therefore, $\mathcal{STA}^{(t)} = \bigcup_{i=1}^{k^{(t)}} \vartheta^{(t)}_i$ such that, for all $i \in \left[k^{(t)}\right]$, we have

$$\vartheta^{(t)}_i = \bigcup\nolimits_{\alpha \in \varsigma^{(t)}_i} \Big\{ \mu \in \mathcal{STA}^{(t)} : RSSI^{(t)}_{\mu,\alpha} > RSSI^{(t)}_{\mu,\alpha'}, \forall \alpha' \in \mathcal{AP}^{(t)} \setminus \{\alpha\} \Big\}. \tag{14}$$

The network at $t$ is said to be partitioned into *clusters* $\mathcal{C}^{(t)}_1, \ldots, \mathcal{C}^{(t)}_{k^{(t)}}$, where $\mathcal{C}^{(t)}_i = \left(\vartheta^{(t)}_i, \varsigma^{(t)}_i\right)$ for all $i \in \left[k^{(t)}\right]$. Assuming that the RSSI is a strictly decreasing function of the distance (if any STA $\mu$ is geographically closer to an AP $\alpha$ more than another $\alpha'$, then $RSSI_{\mu,\alpha} > RSSI_{\mu,\alpha'}$), we can formally determine the *spatial range* occupied by $\mathcal{C}^{(t)}_i$:

$$\bigcup\nolimits_{\alpha \in \varsigma^{(t)}_i} \Big\{ x \in \mathbb{R}^3 : \|x - p_\alpha\|_2 < \|x - p_{\alpha'}\|_2, \forall \alpha' \in \mathcal{AP}^{(t)} \setminus \{\alpha\} \Big\}. \tag{15}$$

If a STA falls into this region of the space, it will belong to $\mathcal{C}^{(t)}_i$. This is, by definition, the union of the Voronoi cells that correspond to the positions of the APs in $\varsigma^{(t)}_i$, in the Voronoi partitioning generated by the positions of the APs in $\mathcal{AP}^{(t)}$. The interest of this result will be considered in Sec. VI.

### E. STA-to-AP association

The STA-to-AP associations in the cluster $\mathcal{C}^{(t)}_i$ will be locally managed by the DQL-MultiMDP agent running in $\mathsf{Process}_i$. The condition that the number of APs of $\varsigma^{(t)}_i$ is under the *controller's capacity* is already ensured by the partitioning mechanism. The fluctuating number of STAs in $\vartheta^{(t)}_i$ is caused either by the mobility or by the switching on/off, and this can be effectively addressed by DQL-MultiMDP thanks to the Multi-MDP approach. Hence, if $\left|\vartheta^{(t)}_i\right| = 0$, the sub-controller $\mathsf{Process}_i$ will wait for STAs to join the cluster; if $\left|\vartheta^{(t)}_i\right| = 1$, it will associate the single STA to the AP with the highest RSSI; it may happen that $\left|\vartheta^{(t)}_i\right| > N_{STA}$, in this case, $\mathsf{Process}_i$ will employ the traditional max-RSSI association strategy; otherwise, when $2 \leq \left|\vartheta^{(t)}_i\right| \leq N_{STA}$, $\mathsf{Process}_i$ will handle $\mathcal{MDP}_{\left|\vartheta^{(t)}_i\right|,\left|\varsigma^{(t)}_i\right|}$, and will train the corresponding local Q-Network, whose paramter vector is denoted as $\theta^{(i)}_{\left|\vartheta^{(t)}_i\right|,\left|\varsigma^{(t)}_i\right|}$.

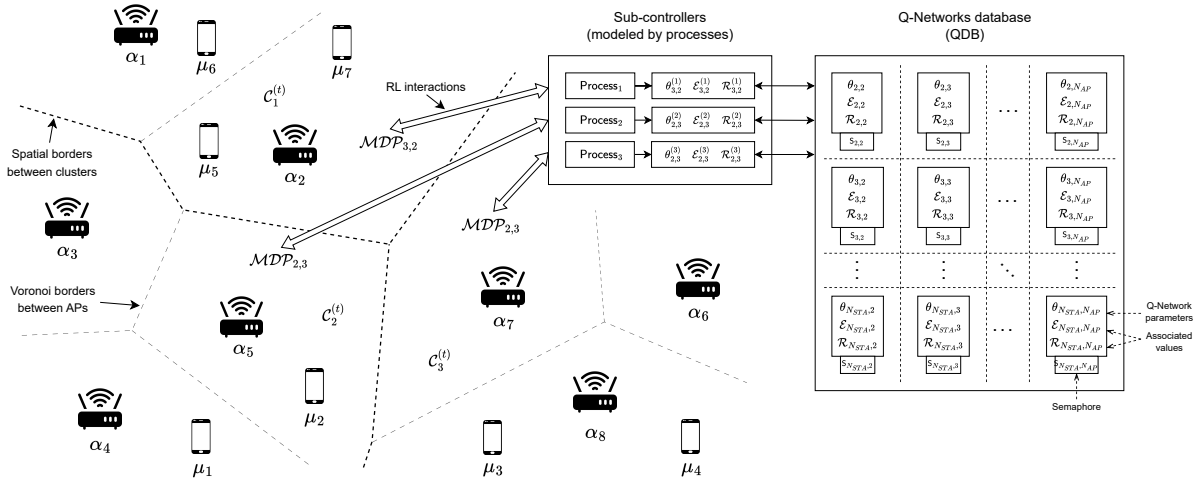



Fig. 1: Proposed abstract architecture

Assume that a STA joins or leaves the cluster at a certain time $t' > t$, so that $2 \leq \left|\vartheta_i^{(t')}\right| \leq N_{STA}$, then $\mathsf{Process}_i$ will transit to a different MDP; it will interrupt the training of $\theta^{(i)}_{\left|\vartheta_i^{(t)}\right|,\left|\varsigma_i^{(t)}\right|}$ to begin the training of $\theta^{(i)}_{\left|\vartheta_i^{(t')}\right|,\left|\varsigma_i^{(t')}\right|}$. It may happen that there was a process that has already visited this MDP and trained the corresponding Q-Network. Hence, it is more efficient to exploit it instead of beginning training with random parameters. Based on this idea, we propose to store the previously trained Q-Networks in a shared database referred to as the Q-Networks database (QDB). This mechanism is developed in Sec. V.

## V. Sharing local updates

The partitioning mechanism forms clusters and assigns a process to independently manage the STA-to-AP association in each one through DQL-MultiMDP. Each process has access to the shared Q-Networks database (QDB). For any pair $(n, m)$ of STAs and APs resp. under the *controller's capacity*, each field in the QBD stores "the best" Q-Network parameters for the corresponding MDP. The Q-Network parameters are the vector $\theta_{n,m}$ associated with two values $\mathcal{E}_{n,m}$ and $\mathcal{R}_{n,m}$. Initially, the Q-Network parameters $\theta_{n,m}$ are initialized randomly, and the associated values are initialized with zeros, as illustrated in Fig. 1 with 3 clusters. Hence, the sub-controllers will submit their local updated Q-Networks after each MDP transition, and the "best" one will be stored in the QDB. The criterion for deciding whether the local update $\theta^{(i)}_{n,m}$ is "better" than the global (stored previously) one $\theta_{n,m}$ is provided by the associated values $\mathcal{E}_{n,m}$ and $\mathcal{R}_{n,m}$. This policy is explained in what follows. Note that since $k^{(t)}$ processes have simultaneous access to the read/write operations in the QDB, each field $(\theta_{n,m}, \mathcal{E}_{n,m}, \mathcal{R}_{n,m})$ is accompanied with a semaphore $\mathsf{S}_{n,m}$.

### A. Local training of a sub-controller $\mathsf{Process}_i$

When $\mathsf{Process}_i$ encounters a transition to $\mathcal{MDP}_{n,m}$, it acquires $\mathsf{S}_{n,m}$, reads the corresponding field, and releases $\mathsf{S}_{n,m}$. We denote the version that it read as $\left(\theta^{\text{old}}_{n,m}, \mathcal{E}^{\text{old}}_{n,m}, \mathcal{R}^{\text{old}}_{n,m}\right)$. It initializes its local Q-Network parameters and associated values as follows: $\theta^{(i)}_{n,m} \leftarrow \theta^{\text{old}}_{n,m}$, $\mathcal{E}^{(i)}_{n,m} \leftarrow \mathcal{E}^{\text{old}}_{n,m}$, and $\mathcal{R}^{(i)}_{n,m} \leftarrow \mathcal{R}^{\text{old}}_{n,m}$; and begins interacting with $\mathcal{MDP}_{n,m}$. After completing a certain number of RL episodes, denoted as $E^{(i)}_{n,m}$, and over which the sequence of the sum of received rewards is denoted as $\left(R^{(i)}_{n,m,h}\right)_{h\in\left[E^{(i)}_{n,m}\right]}$, $\mathsf{Process}_i$ makes a transition to another MDP $\mathcal{MDP}_{n',m'}$. Hence, it interrupts its local training of $\theta^{(i)}_{n,m}$, and computes its local values as follows:

$$\mathcal{E}^{(i)}_{n,m} \leftarrow \mathcal{E}^{\text{old}}_{n,m} + E^{(i)}_{n,m} \tag{16}$$

$$\mathcal{R}^{(i)}_{n,m} \leftarrow \frac{1}{\mathcal{E}^{(i)}_{n,m}} \left[\mathcal{E}^{\text{old}}_{n,m}\mathcal{R}^{\text{old}}_{n,m} + \sum_{h=1}^{E^{(i)}_{n,m}} (1-\omega^{h+\mathcal{E}^{\text{old}}_{n,m}}) R^{(i)}_{n,m,h}\right],$$

for a certain $\omega \in [0, 1)$. Afterward, $\mathsf{Process}_i$ acquires $\mathsf{S}_{n,m}$ again to read the new value $\mathcal{R}_{n,m}$ from the QDB in the corresponding field, that we denote as $\mathcal{R}^{\text{new}}_{n,m}$. Note that this field can be updated by some process $\mathsf{Process}_j$ during $\mathsf{Process}_i$'s interactions, so the extracted value may be different from $\mathcal{R}^{\text{old}}_{n,m}$. $\mathsf{Process}_i$ tests if $\mathcal{R}^{(i)}_{n,m} > \mathcal{R}^{\text{new}}_{n,m}$; if the condition is met, that means that the local Q-Network with parameters $\theta^{(i)}_{n,m}$ is better than the QDB's version $\theta_{n,m}$, then it updates the filed with its local values $(\theta_{n,m}, \mathcal{E}_{n,m}, \mathcal{R}_{n,m}) \leftarrow \left(\theta^{(i)}_{n,m}, \mathcal{E}^{(i)}_{n,m}, \mathcal{R}^{(i)}_{n,m}\right)$. Then, it releases $\mathsf{S}_{n,m}$ and frees its memory from these values, and proceeds to its interactions with $\mathcal{MDP}_{n',m'}$ following a similar procedure.

To explain the idea behind the design of the updating rules in Eq. 16, consider the following scenario: after the initialization, $\mathsf{Process}_i$ is the first sub-controller that observed $\mathcal{MDP}_{n,m}$, after completing $E^{(i)}_{n,m}$ episodes and encountering a local MDP transition, it commits its local updates $\left(\theta^{(i)}_{n,m}, \mathcal{E}^{(i)}_{n,m}, \mathcal{R}^{(i)}_{n,m}\right)$ to the QDB because $\mathcal{R}^{(i)}_{n,m} > 0$. After that, another process $\mathsf{Process}_j$ transited to the same MDP and retrieved $\mathsf{Process}_i$'s update to use it for initialization. After completing $E^{(i)}_{n,m}$ episodes and encountering a transition, we assume that $\mathcal{R}^{(i)}_{n,m} > \mathcal{R}^{(j)}_{n,m}$, then $\mathsf{Process}_j$ committed its local update $\left(\theta^{(j)}_{n,m}, \mathcal{E}^{(j)}_{n,m}, \mathcal{R}^{(j)}_{n,m}\right)$ that will be the current version of that field in the QDB.

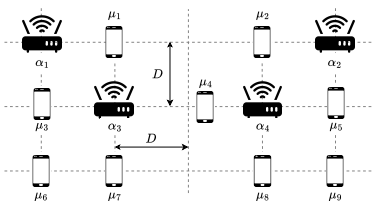



Fig. 2: Network setting

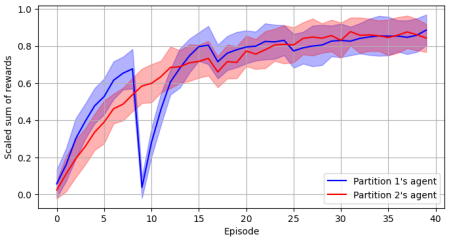



Fig. 3: Learning performance

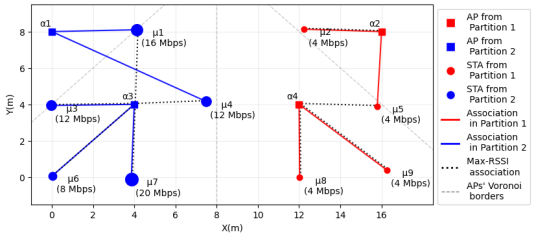



Fig. 4: Association results

We first observe that the current global Q-Network $\theta_{n,m}$ is trained by interacting with two environments: it is first trained in $\mathcal{C}_i^{(.)}$, and completed another training phase in $\mathcal{C}_j^{(.)}$; this will intuitively provide a certain generalization ability to the Q-Networks to operate in unseen environments since they will be trained in different environments following this strategy. Secondly, following Eq. 16, we notice that $\mathcal{E}_{n,m}^{(i)} = E_{n,m}^{(i)}$ and $\mathcal{E}_{n,m}^{(j)} = E_{n,m}^{(i)} + E_{n,m}^{(j)}$, which means that the associated value $\mathcal{E}_{n,m}$ represents the total number of episodes over which $\theta_{n,m}$ is trained. In addition, we have

$$\mathcal{R}_{n,m}^{(i)} = \frac{1}{E_{n,m}^{(i)}} \left[ \sum_{h=1}^{E_{n,m}^{(i)}} (1-\omega^h) R_{n,m,h}^{(i)} \right], \qquad (17)$$

that can be seen as a weighted average of the sums of received rewards during the episodes of $\theta_{n,m}^{(i)}$'s training. Recall that $\mathcal{R}_{n,m}^{(i)}$ is the criterion that represents the "quality" of the local Q-Network. The assigned weight $(1-\omega^h)$ is small for the early episodes, and it exponentially approaches 1 for the later ones. This is because the Q-Network's parameters are initialized randomly, so the early received rewards should not be "considered too important" to represent the quality of the derived policy from the Q-Network. Thus, the sum "penalizes" them and assigns more importance to the sums of received rewards in the later episodes. On the other hand, by letting $\left(\tilde{R}_{n,m,h}\right)_{h \in \left[E_{n,m}^{(i)} + E_{n,m}^{(j)}\right]}$ denote the concatenation of the sums of received rewards during the training of $\theta_{n,m}^{(i)}$ and $\theta_{n,m}^{(j)}$, and by replacing $\mathcal{E}_{n,m}^{\text{old}}$ (resp. $\mathcal{R}_{n,m}^{\text{old}}$) by $\mathcal{E}_{n,m}^{(i)}$ (resp. $\mathcal{R}_{n,m}^{(i)}$) in Eq. 16, we can show that

$$\mathcal{R}_{n,m}^{(j)} = \frac{1}{E_{n,m}^{(i)} + E_{n,m}^{(j)}} \left[ \sum_{h=1}^{E_{n,m}^{(i)} + E_{n,m}^{(j)}} (1-\omega^h) \tilde{R}_{n,m,h} \right]. \qquad (18)$$

Hence, $\mathcal{R}_{n,m}^{(j)}$ is a weighted average of the sums of received rewards during the whole training of $\theta_{n,m}^{(j)}$.

This analysis yields the following generalization: in each field $(\theta_{n,m}, \mathcal{E}_{n,m}, \mathcal{R}_{n,m})$ of the QDB, $\theta_{n,m}$ is by far the "best" Q-Network parameters vector for $\mathcal{MDP}_{n,m}$, $\mathcal{E}_{n,m}$ is the total number of episodes over which it is trained, and $\mathcal{R}_{n,m}$ is a weighted average of the sums of the received rewards during these episodes. This can be proved by considering another sub-controller $\text{Process}_k$ that retrieved the update of $\text{Process}_j$ and committed its local update after following Eq. 16, *ad Infinitum*.

## VI. Experimental results

In the following experiments, we use the `ns3-gym` toolkit to interface the simulated IEEE 802.11ax network configurations under the `ns3` simulator with the Constrained Clustering algorithm that forms the APs partitions and their corresponding DQL-MultiMDP agents running as Python threads.

### A. Effectivness of the partitioning approach

This experiment aims to validate the effectiveness of the partitioning approach. We consider the simplistic network setting depicted in Fig. 2. If the basic DQL-MultiMDP is applied, the network will be managed by a single agent that interacts with $\mathcal{MDP}_{4,9}$ when all the devices are active. Hence, the number of available actions in this MDP is $4^9$. Thus, despite the simplicity of the state representations and the dynamics of the environment, the corresponding agent still needs hundreds of episodes to explore all the actions and reach stability at the optimal policy. By maintaining all the APs active and by specifying a number of clusters $k = 2$ and a size constraint $\tau = 2$, firstly, the Constrained $k$-means Clustering algorithm partitions the network into two clusters. The first one is composed of the APs $\alpha_1$ and $\alpha_2$, and as all the APs are of identical transmission parameters, the STAs in the cluster's spatial range are $\mu_1, \mu_3, \mu_6$ and $\mu_7$. The second cluster contains the APs $\alpha_3$ and $\alpha_4$, and the remaining STAs, including $\mu_4$ since it is slightly closer to $\alpha_4$ than $\alpha_3$ to ensure that it falls within the spatial range of the second cluster. Afterward, a thread is assigned to each cluster to manage the STA-to-AP association through DQL-MultiMDP in parallel. The threads stimulate the clusters' sub-controllers following the abstraction in Sec. V, so we similarly refer to them as $\text{Process}_1$ and $\text{Process}_2$. The first thing that we can notice is that the agent running in $\text{Process}_1$ will handle $\mathcal{MDP}_{4,2}$ if all the devices are active, and the one running in $\text{Process}_2$ will handle $\mathcal{MDP}_{5,2}$. The action space sizes of these MDPs are respectively $2^4$ and $2^5$ and are extremely small compared to the previous $\mathcal{MDP}_{4,9}$'s action space size. In addition, the sub-controllers do not store the Q-Network parameters; instead, they commit their updates to the QDB, eliminating duplicate parameter storage.

Recall that the two main factors causing the MDP transitions are the switching on/off and the mobility, i.e., when a STA migrates from one cluster to another. Hence, to understand the dynamics of the learning process, we simulate the following scenario. The devices remain active during the simulation except $\mu_7$: 4s after launching the simulation, $\mu_7$ will be turned off, and 8 s later, it will be turned on again and remain active till the end. The STAs are moving according to a Markov chain lattice random walk, but they remain in their cluster's spatial range, except $\mu_4$ will migrate to the first cluster at a certain

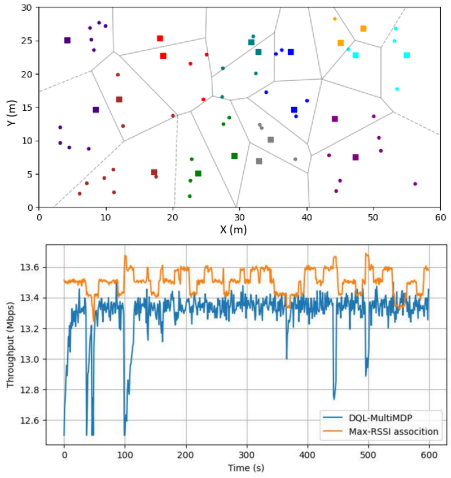



Fig. 5: Many clusters, faster convergence

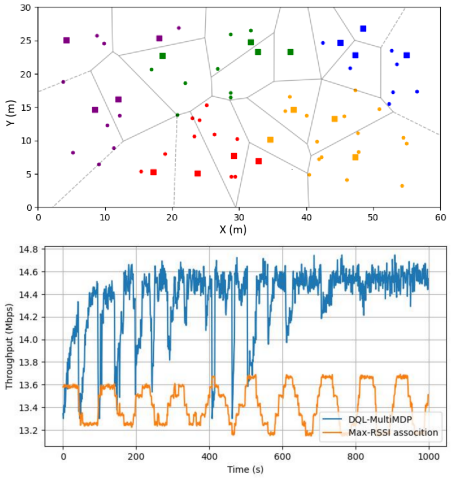



Fig. 6: Few clusters, more quality

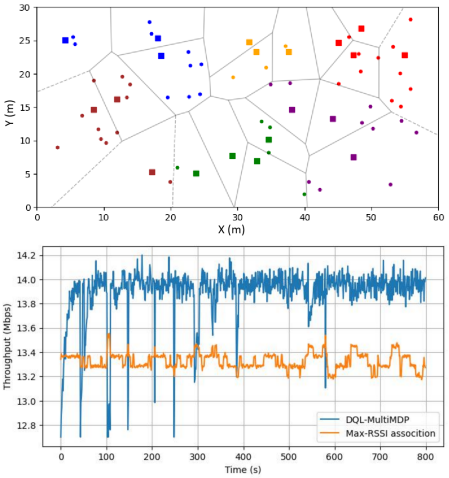



Fig. 7: Moderate number of clusters

moment. After repeating this scenario 30 times and keeping track of the evolution of the sum of received rewards by each agent, we scaled these rewards to $[0, 1]$ to focus solely on the evolution of learned policies' quality.

When $\mu_7$ turns off, during the 6th episode on average, as illustrated in Fig. 3, $\mathsf{Process}_1$ transits from $\mathcal{MDP}_{4,2}$ to $\mathcal{MDP}_{3,2}$, it extracts the parameters of the corresponding Q-Network $\theta_{3,2}$ from the QDB, which are by far initialized randomly, and it commits its locally trained parameters from the previous MDP, i.e. $\theta^{(1)}_{4,2}$, alongside its associated values. Hence, the plot drops since $\mathsf{Process}_1$ starts a new training phase "from scratch," i.e., using a Q-Network with random parameters. Afterward, the migration of $\mu_4$ from the second to the first cluster happens during the 16th episode on average. Here, both $\mathsf{Process}_1$ and $\mathsf{Process}_2$ encounter a transition to $\mathcal{MDP}_{4,2}$. Then, they extract the current version of $\theta_{4,2}$ from the QDB (which is the previous update of $\mathsf{Process}_1$), and they commit their local updates $\theta^{(1)}_{3,2}$ and $\theta^{(2)}_{5,2}$ with their associated values. Consequently, the evolution of the sum of received rewards by both processes encounters a slight drop in the 17th episode, and because $\theta_{4,2}$ is already trained "a bit," the drop is not as strong as the previous one. When $\mu_7$ is turned on after 8 s, $\mathsf{Process}_1$ transits to $\mathcal{MDP}_{5,2}$ and gets $\theta_{5,2}$ from the QDB; which is the previous update committed by $\mathsf{Process}_2$ after the migration event.

Overall, this basic experimental scenario showed the two main strengths of the proposed approach: 1) as a divide-and-conquer approach that overcomes the "curse" of the exponential expansion of the action space when assigning a single controller to manage the whole network, and 2) as an efficient framework for collaborative training of Q-Networks because the sub-controllers exploit each other updates to avoid starting training from scratch. Therefore, in only about 35 episodes, the system explored and reached the optimal policy within all the MDPs of the scenario (i.e., $\mathcal{MDP}_{3,2}$, $\mathcal{MDP}_{4,2}$, and $\mathcal{MDP}_{5,2}$), which would require hundreds of episodes for basic DQL-MultiMDP to do so. The association result at a "snapshot" of the network while the STAs are moving is depicted in Fig. 4; the size of the dots representing the STAs is proportional to their required data rates to facilitate visual analysis. We can notice that the traditional Max-RSSI association strategy causes a high load on $\alpha_3$ and $\alpha_4$ while two other APs are free. The learned policy, in contrast, gives priority to the STAs with high requirements to assign them to close APs while balancing the load on the APs.

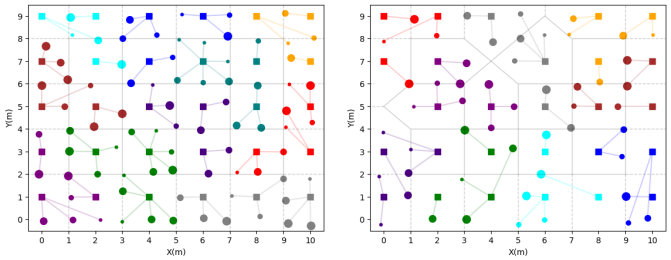
### B. Choice of $(k^{(t)}, \tau^{(t)})$

We investigate in this part the influence of the choice of $\left(k^{(t)}, \tau^{(t)}\right)$ on the learning performance, aiming to design a strategy for their selection. We simulated a network consisting of 20 APs and 50 mobile STAs. Initially, the devices are uniformly distributed in a 60m $\times$ 30m plan. The APs remain active during the simulation, and the clustering algorithm is run only once at the beginning to evaluate the performance regarding a specific choice of $k^{(\cdot)}$ and $\tau^{(\cdot)}$. Meanwhile, the STAs adhere to the switching on/off model described in Sec. III and follow the Markov chain lattice random walk mobility model. Here, we evaluate the network's mean throughput instead of the sum of received rewards across episodes.

*1) Many clusters, faster convergence:* We consider

$$k^{(t)} = \max\left\{1, \left\lfloor N^{(t)}_{AP}/2 \right\rfloor\right\}; \; \tau^{(t)} = \min\left\{2, N^{(t)}_{AP}\right\}. \quad (19)$$

The size constraint is set to avoid resulting in empty or single-element AP clusters. Notice that if the network contains fewer than two APs, the strategy will form only one cluster, and $\tau^{(t)}$ will be equal to the number of APs (Eq. 5 is always satisfied). As illustrated in Fig. 5, the partitioning result formed 10 clusters, each composed of 2 APs. Firstly, each of the 10 sub-controllers observes $\mathcal{MDP}_{N^{(t)}_{STA},2}$ whose convergence to the optimal policy is fast. Secondly, each STA has two candidate APs for association. The performance comparison with the traditional max-RSSI association strategy is given in Fig. 5. Inadequacy with the previous simulation, the drops in the mean throughput plot are caused by sub-controllers transiting to non-visited MDPs. The mean throughput stabilizes after the early few seconds because the observed MDPs are of small action




(a) 80 STAs and 30 APs (b) 51 STAs and 27 APs

Fig. 8: Association results (clustering follows Eq. 21)

spaces, and also thanks to the DQB mechanism, which avoids beginning the training with random parameters.

*2) Few clusters, more quality:* Considering the strategy

$$k^{(t)} = \max\left\{1, \left\lfloor N_{AP}^{(t)}/4 \right\rfloor\right\};\ \tau^{(t)} = \min\left\{3, N_{AP}^{(t)}\right\}, \quad (20)$$

the partitioning mechanism formed a smaller number of clusters (5 clusters) with a size constraint set to 3. The clusters are consequently larger compared to the previous strategy both in terms of the number of APs and the number of STAs because their *spatial range* is increased. Consequently, the observed MDPs by the sub-controllers are of large action spaces. As illustrated in Fig. 6, the stability of the mean throughput after reaching the optimal policy in all the visited MDPs is achieved after about 600s (i.e., 6 times longer than the previous partitioning strategy). However, the difference by which the mean throughput surpasses the previous strategy is larger; this is because each STA has more candidate APs to be associated with compared to the previous strategy.

The learned lesson from the previous two experiments is that choosing a large $k^{(\cdot)}$ and small $\tau^{(\cdot)}$ partitions the network into small clusters, the required time to reach stability is shorter, but the association policy is not much efficient compared to the max-RSSI association strategy. Conversely, choosing a small $k^{(t)}$ and large $\tau^{(t)}$ forms a small number of large clusters; the required time to reach stability is longer, but the association policy is better. Hence, an optimal strategy for such a choice should balance the quality of the learned association policies and the required time to achieve stability. Therefore, we propose the following strategy.

*3) Moderate number of clusters:* After choosing

$$k^{(t)} = \max\left\{1, \left\lfloor N_{AP}^{(t)}/3 \right\rfloor\right\};\ \tau^{(t)} = \min\left\{3, N_{AP}^{(t)}\right\}, \quad (21)$$

the corresponding result is illustrated in Fig. 7; the stability is reached after about 300s (two times faster than the second strategy), and the mean throughput is much more improved compared to the first one.

After achieving throughput stability, we extracted the trained parameters of the Q-Networks from the QDB in this scenario to visualize the association decisions in a network of 90 mobile STAs and 30 APs. Initially, all devices are active and arranged in a grid-like pattern in a $11 \times 10$ m$^2$ surface. The used strategy for selecting the pair $(k^{(\cdot)}, \tau^{(\cdot)})$ follows Eq. 21. Initially, when all the devices are active, the association decisions are shown in Fig. 8a. The different colors represent the devices in each cluster, and the grey lines represent the Voronoi borders between the APs given their position. When the number of STAs in a certain cluster exceeds $N_{STA} = 8$, e.g., the teal cluster, the corresponding sub-controller performs the max-RSSI association strategy. However, load balancing is performed in most clusters using DQL-MultiMDP; we observe that the learned policy follows the same philosophy: the priority to the close AP is given to STAs with high data rates, and the load is balanced on the APs. A few seconds later, 3 APs and 29 STAs are turned off; a new partitioning is performed following the same strategy, and a 'snapshot' of the updated association decisions is illustrated in Fig. 8b.

## VII. Conclusion, Current and Future Work

In this paper, we propose an approach to address the scalability of DQL-MultiMDP and extend its operation to large WiFi networks. This is achieved through a constrained network partitioning mechanism that dynamically divides the network into clusters and assigns a sub-controller to manage the STA-to-AP association independently in each cluster. Additionally, the sub-controllers can exchange their training updates for faster convergence and avoid starting training with random parameters. Our current work focuses on implementation issues, designing more sophisticated partitioning mechanisms that also balance the number of STAs in the clusters, along with efficient Q-Network aggregation strategies, instead of only keeping the best Q-Network in the QDB.

## References

[1] (2022) Cisco: Access Point Power Control. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/wireless/controller/9800/17-9/config-guide/b_wl_17_9_cg/m_access_point_power_control.pdf

[2] M. Heni *et al.*, "Energy consumption model in ad hoc mobile network," *arXiv preprint arXiv:1206.1426*, 2012.

[3] E. Khorov *et al.*, "Current status and directions of IEEE 802.11 be, the future Wi-Fi 7," *IEEE access*, vol. 8, pp. 88 664–88 688, 2020.

[4] S. Verma *et al.*, "A survey on Multi-AP coordination approaches over emerging WLANs: Future directions and open challenges," *IEEE Communications Surveys & Tutorials*, 2023.

[5] E. Khorov *et al.*, "A tutorial on IEEE 802.11 ax high efficiency WLANs," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 197–216, 2018.

[6] M. Bellouch *et al.*, "Fair WiFi STA-to-AP association with DQL-MultiMDP: Investigation and opportunities," in *2024 Global Information Infrastructure and Networking Symposium (GIIS)*. IEEE, 2024, pp. 1–5.

[7] W. K. Soo *et al.*, "Survey on load-balancing methods in 802.11 infrastructure mode wireless networks for improving quality of service," *ACM Computing Surveys (CSUR)*, vol. 51, no. 2, pp. 1–21, 2018.

[8] M. A. Kafi *et al.*, "On-line client association scheme based on reinforcement learning for WLAN networks," in *IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2019, pp. 1–7.

[9] R. Ahmad *et al.*, "Reinforcement learning-based near-optimal load balancing for heterogeneous LiFi WiFi network," *IEEE Systems Journal*, vol. 16, no. 2, pp. 3084–3095, 2021.

[10] B. Heller *et al.*, "The controller placement problem," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 473–478, 2012.

[11] G. Wang *et al.*, "The controller placement problem in software defined networking: A survey," *IEEE network*, vol. 31, no. 5, pp. 21–27, 2017.

[12] P. S. Bradley, K. P. Bennett, and A. Demiriz, "Constrained k-means clustering," *Microsoft Research, Redmond*, vol. 20, no. 0, p. 0, 2000.