Reinforcement Learning-based Wi-Fi Contention Window Optimization

Sheila C. da S. J. Cruz, Messaoud Ahmed Ouameur, and Felipe A. P. de Figueiredo

***Abstract*—The collision avoidance mechanism adopted by the IEEE 802.11 standard is not optimal. The mechanism employs a binary exponential backoff (BEB) algorithm in the medium access control (MAC) layer. Such an algorithm increases the backoff interval whenever a collision is detected to minimize the probability of subsequent collisions. However, the increase of the backoff interval causes degradation of the radio spectrum utilization (i.e., bandwidth wastage). That problem worsens when the network has to manage the channel access to a dense number of stations, leading to a dramatic decrease in network performance. Furthermore, a wrong backoff setting increases the probability of collisions such that the stations experience numerous collisions before achieving the optimal backoff value. Therefore, to mitigate bandwidth wastage and, consequently, maximize the network performance, this work proposes using reinforcement learning (RL) algorithms, namely Deep Q Learning (DQN) and Deep Deterministic Policy Gradient (DDPG), to tackle such an optimization problem. In our proposed approach, we assess two different observation metrics, the average of the normalized level of the transmission queue of all associated stations and the probability of collisions. The overall network's throughput is defined as the reward. The action is the contention window (CW) value that maximizes throughput while minimizing the number of collisions. As for the simulations, the NS-3 network simulator is used along with a toolkit known as NS3-gym, which integrates a reinforcement-learning (RL) framework into NS-3. The results demonstrate that DQN and DDPG have much better performance than BEB for both static and dynamic scenarios, regardless of the number of stations. Additionally, our results show that observations based on the average of the normalized level of the transmission queues have a slightly better performance than observations based on the collision probability. Moreover, the performance difference with BEB is amplified as the number of stations increases, with DQN and DDPG showing a 45.52% increase in throughput with 50 stations. Furthermore, DQN and DDPG presented similar performances, meaning that either one could be employed.**

***Index Terms*—Wi-Fi, contention-based access scheme, channel utilization optimization, machine learning, reinforcement learning, NS-3.**

I. Introduction

THE IEEE 802.11, or simply Wi-Fi, is a set of wireless network standards designed and maintained by the Institute of Electrical and Electronics Engineers (IEEE) that defines MAC and physical layer (PHY) protocols for deploying wireless local area networks (WLANs). Their MAC layer implements a contention-based protocol, known as carrier-sensing multiple access with collision avoidance (CSMA/CA), for the nodes to access the wireless medium (i.e., the channel) efficiently [1], [2]. With CSMA/CA, the nodes compete to access the radio resources and consequently, the channel [3], [4].

One of the most critical parameters of the CSMA/CA mechanism is the contention window (CW) value, also known as back-off time, which is a random delay used for reducing the risk of collisions. If the medium is busy, an about-to-transmit Wi-Fi device selects a random number uniformly distributed within the interval [0, CW] as its back-off value, which defers its transmission to a later time. CW has its value doubled every time a collision occurs (e.g., when an ACK is not received), reducing the likelihood of multiple stations selecting the same back-off value. CW values range from the minimum contention window (CWMin) value, generally equal to 15 or 31 depending on the Wi-Fi standard, to the established maximum contention window (CWMax) value, which is equal to 1023. CW is reset to CWMin when an ACK is received, or the maximum number of re-transmissions has been reached [5]. This deferring mechanism is also known as binary exponential back-off (BEB) [6] and is shown in Fig 1.

The BEB algorithm has several limitations and may not always be the best solution for collision avoidance, often providing suboptimal solutions [7], [8]. These limitations include inefficiency under high loads, lack of fairness among competing nodes, inability to adapt to changing network conditions, vulnerability to the hidden node problem, and no global optimization. While the BEB algorithm is widely used in Wi-Fi networks, it may not always be the best solution for collision avoidance. Other approaches, such as machine learning-based ones, may be better able to address the limitations of the BEB algorithm and provide more effective collision avoidance strategies.

Sheila C. da S. J. Cruz and Felipe A. P. de Figueiredo are with the National Institute of Telecommunications (INATEL), Minas Gerais, Brazil (e-mail: sheila.cassia@mtel.inatel.br, felipe.figueiredo@inatel.br); Messaoud Ahmed Ouameur is with Université du Québec à Trois-Rivières (UQTR), Quebec, Canada (e-mail: messaoud.ahmed.ouameur@uqtr.ca).

This work was partially funded by Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) - Grant No. 2070.01.0004709/2021-28, by FADCOM - Fundo de Apoio ao Desenvolvimento das Comunicações, presidential decree no 264/10, November 26, 2010, Republic of Angola, by Huawei, under the project Advanced Academic Education in Telecommunications Networks and Systems, Grant No. PPA6001BRA23032110257684, by the Brazilian National Council for Research and Development (CNPq) under Grant Nos. 313036/2020-9 and 403827/2021-3, by São Paulo Research Foundation (FAPESP) under Grant No. 2021/06946-0, by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) and RNP, with resources from MCTIC, under Grant Nos. 01250.075413/2018-04, 01245.010604/2020-14, and 01245.020548/2021-07 under the Brazil 6G project of the Radiocommunication Reference Center (Centro de Referência em Radiocomunicações - CRR) of the National Institute of Telecommunications (Instituto Nacional de Telecomunicações - Inatel), Brazil; and by FCT/MCTES through national funds and when applicable co-funded EU funds under the Project UIDB/EEA/50008/2020.

Digital Object Identifier: 10.14209/jcis.2023.15

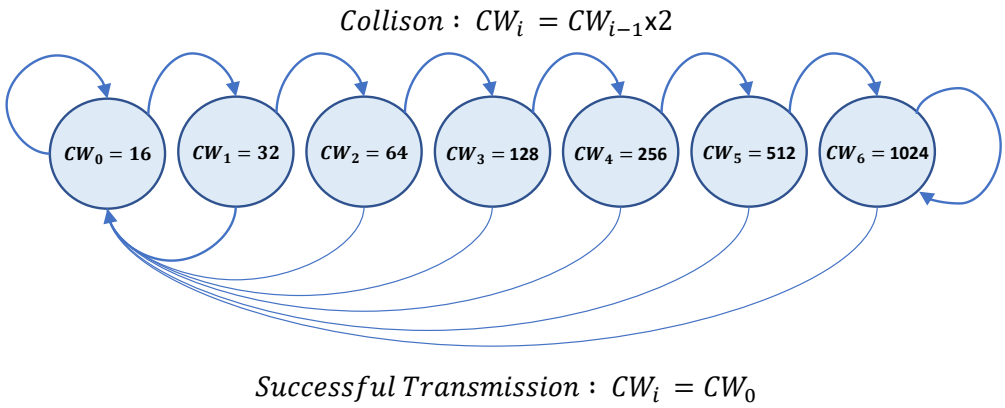

Fig. 1. Binary Exponential Back-off.

In scenarios with few nodes, collisions will be less frequent and impactful, especially in static scenarios, where the number of nodes remains the same. On the other hand, in dynamic scenarios, where the number of nodes increases throughout time, collisions will be commonplace. Furthermore, the high number of collisions reduces the network throughput drastically since CW has its value doubled when collisions are detected, leading to an inefficient network operation [9].

As can be seen, optimizing the CW value could be beneficial for Wi-Fi networks since the traditional BEB algorithm does not scale well when many nodes compete for the medium [10]. Once network devices with high computational capabilities become increasingly common, CW can be optimized through machine learning (ML) algorithms. In ML, the most common learning paradigms are supervised, unsupervised, and reinforcement learning. Supervised algorithms require a labeled dataset where the outcomes for the respective inputs are known. Still, creating such a dataset requires a model and a solution to the problem. Developing accurate models is, in several cases, a challenging and troublesome task. Besides that, many solutions are suboptimal, and supervised ML algorithms learning from datasets created with those solutions will never surpass its performance since the algorithm tries to replicate the input to label mapping present in the dataset [11].

Unsupervised learning occurs when the ML algorithm is trained using unlabeled data. The idea behind this paradigm is to find hidden (i.e., latent) patterns in the data. This paradigm could be used to identify hidden patterns in Wi-Fi traffic that could be contributing to collisions. By clustering similar traffic patterns, it may be possible to identify common sources of interference or other issues that are leading to collisions and take actions to mitigate them. Nonetheless, unsupervised learning can be limited in its applicability to collision avoidance in Wi-Fi networks due to the lack of labeled data, limited interpretability of results, limited control over the learning process, and lack of a feedback loop for ongoing adjustment and optimization. While unsupervised learning can be useful for analyzing Wi-Fi traffic data and identifying patterns, other machine learning paradigms, such as reinforcement learning, may be better suited to the problem of collision avoidance.

Finally, reinforcement learning occurs when the ML algorithm (a.k.a. agent in this context) interacts with the environment through trial and error action attempts without requiring labels, only reward information about the taken actions. This paradigm allows the agents (e.g., Wi-Fi nodes) to explore the environment by taking random actions and finding optimal solutions. This paradigm can be employed to optimize the actions of Wi-Fi nodes in a network to minimize collisions and maximize throughput. The nodes can learn to take actions that lead to higher reward values (e.g., successful transmissions) while avoiding actions that lead to lower reward values (e.g., collisions). This paradigm presents several advantages for collision avoidance, including the ability to learn from experience, optimize long-term performance, handle complex and dynamic environments, explore and exploit different strategies, and adapt to different contexts. RL algorithms can learn from the state of the network and adapt to changing network conditions, optimizing cumulative rewards over time. This makes RL well-suited to handle the complexity of Wi-Fi networks, find optimal solutions, and navigate different scenarios. Additionally, RL is flexible and adaptable, making it a powerful tool for collision avoidance in Wi-Fi networks. Therefore, since there is no optimal solution for the CW optimization (BEB is known to be suboptimal) [12]–[14], RL may offer more effective collision avoidance strategies. However, despite the increasing state-of-the-art contributions presented, RL-based algorithms are complex, presenting high computational requirements, require fine-tuned hyperparameters since they are sensitive to their settings, the training process might take several hours and need extensive exploration, requiring a significant amount of computing power, and present difficulties in handling continuous and high-dimensional state and action spaces [15].

RL algorithms have addressed their limitations through various approaches. To mitigate high computational requirements, techniques like parallelization (i.e., distributed computing) and hardware acceleration have been employed to leverage the power of multiple processors or machines, and hardware acceleration using specialized processors like graphics processing units (GPUs) or tensor processing units (TPUs) to expedite the training process [16]. To tackle the need for extensive training data and exploration, methods such as experience replay,

where past experiences are stored and randomly sampled during training, allowing for efficient and effective utilization of data [17]. Exploration strategies, such as epsilon-greedy or Thompson sampling, strike a balance between exploiting learned knowledge and exploring new possibilities [18]. Moreover, curiosity-driven or intrinsic motivation learning techniques have been explored to encourage agents to explore novel states or seek out new experiences, thereby reducing the reliance on vast amounts of explicit training data [19]. Sensitivity to hyperparameters is addressed through techniques like grid search, systematically exploring different hyperparameter combinations to find optimal settings [20]. More advanced approaches, such as Bayesian optimization, leverage probabilistic models to search the hyperparameter space intelligently [21]. Furthermore, algorithmic advancements like automatic hyperparameter tuning methods, utilizing meta-learning or reinforcement learning itself, have been developed to automate the selection of hyperparameters, enhancing the performance of the algorithms [22]. Difficulties in handling continuous and high-dimensional spaces are overcome using deep neural networks and parameterization methods. Deep neural networks are used to function approximation, allowing for learning policies directly from raw sensory inputs [23]. Continuous action spaces are handled through parameterization methods, such as using Gaussian distributions or policy parameterization [24]. Additionally, actor-critic architectures, combining policy and value function approximators, have shown promise in effectively handling high-dimensional state and action spaces [25]. These ongoing advancements aim to improve the efficiency and performance of reinforcement learning algorithms across different domains.

This work proposes using deep reinforcement learning (DRL) algorithms to optimize the CW value selection and improve the performance of Wi-Fi networks by maintaining a stable throughput and minimizing collisions[1]. More specifically, we propose an RL-aided centralized solution aimed at finding the best CW value that maximizes the overall Wi-Fi network's throughput by properly setting CW values, especially in dense environments, with dozens to hundreds of stations. The proposed solution takes actions, i.e., selects a CW value based on the observation metric adopted. In this work, we study and compare the use of two distinct DRL algorithms, namely, DQN and DDPG, and two different observation metrics, namely, averaged normalized transmission queues' level of all associated stations and the collision probability, to tackle the CW optimization problem. DQN was chosen because it is relatively simple and has a discrete action space. However, despite its simplicity, DQN generally displays performance and flexibility that rivals other methods [26]. DDPG was selected since it is a more complex method that represents actions as continuous values, yielding an exciting comparison with DQN [27]. By using DDPG, we want to explore the hypothesis that the proposed solution can adjust the CW value more precisely and in a more fine-grained fashion with a continuous action space. Therefore, the main contributions of this work are as follows:

1) Adoption of the averaged normalized transmission queue level as observation information used by the RL agent to take actions.
2) The comparison and assessment of two different observation metrics, namely, the averaged normalized transmission queue level and the collision probability.
3) A CW optimization solution that applies to any of the 802.11 standards.

The remainder of the paper is organized as follows. Section II discusses related work. Section III presents a brief machine learning overview. Section IV describes the materials and methods used in the simulations. Section V presents the simulation results. Finally, Section VI presents conclusions and future works.

II. Related Work

The literature presents adequate and excellent contributions of ML methods applied to CW optimization in wireless networks. For example, in [28], the authors propose a CW optimization mechanism for IEEE 802.11ax under dynamically varying network conditions employing DRL algorithms. The DRL algorithms were implemented on the NS-3 [29] simulator using the NS3-gym [30] framework, which enables integration with python frameworks [30]. They proved to have efficiency close to optimal according to the throughput result that remained stable even when the network topology changed dynamically. Their solution uses the collision probability, i.e., the transmission failure probability, to observe the overall network's status.

To allow channel access and a fair share of the unlicensed spectrum between wireless nodes, the authors in [31] propose an intelligent ML solution based on CWmin (minimum CW) adaptation. The issue is that aggressive nodes, as they refer to in the paper, try to access the medium by forcefully choosing low CWmin values, while CSMA/CA-based nodes have a fixed CWmin set to 16, leading to an unfair share of the spectrum. The intelligent CW ML solution consists of a random forest, which is a supervised classification algorithm. Simulations were conducted on a C++ discrete-event-based simulator called CSIM [32] to evaluate the algorithm's performance. It was possible to obtain high throughput efficiency while maintaining fair allocation of the unlicensed channels with other coexisting nodes.

In [33], the authors present a Deep Q-learning algorithm to dynamically adapt CWmin to random access in wireless networks. The idea is to maximize a network utility function (i.e., a metric measuring the fair use of the medium) [34] under dynamic and uncertain scenarios by rewarding the actions that lead to high utilities (efficient resource usage). The proposed solution employs an intelligent node, called node 0, that implements the DQN algorithm to choose the CWmin for the next time step from historical observations. The simulation was conducted on NS-3 to evaluate the performance against the following baselines: optimal design, random forest classifier, fairness index, optimal constant, and standard protocol (with its CWmin fixed at 32). Two scenarios were considered for

[1]The source code for the reproduction of the results is available on: https://github.com/sheila-janota/RLinWiFi-avg-queue-level

the simulation. The first scenario uses two states and follows a Markov process for CWmins of all nodes except node 0. The RL algorithm and random forest classifier reach outstanding performance for this case. The second scenario considers five states, and CWmins of all nodes different from node 0 follow a more complex process. The RL algorithm achieves utility close to optimal when compared to a supervised random forest classifier.

In [10], the authors propose an ML-based solution using a Fixed-Share algorithm to adjust the CW value to improve network performance dynamically. The algorithm comprises CW calculation, Loss/Gain function, and sharing weights. The algorithm considers the present and recent past conditions of the network. The NS-3 network simulator was used to evaluate the proposed solution, and the performance metrics used were average throughput, average end-to-end delay, and channel access fairness. The Fixed-Share algorithm achieves excellent performance compared to the other two conventional algorithms: binary exponential backoff (BEB) and History-Based Adaptive Backoff (HBAB).

To optimize CW in a wireless local area network, [35] presents three algorithms based on genetic fuzzy-contention window optimization (GF-CWO), which is a combination of fuzzy logic controller and a genetic algorithm. The proposed algorithm is intended to solve issues related to success ratio, packet loss ratio, collision rate, fairness index, and energy consumption. Simulations were conducted in Matlab in order to evaluate the performance of the proposed solution, producing better results when compared to the BEB.

To avoid packet collisions in Mobile ad hoc networks (MANETs) [36], [37], the authors of [38] propose a Q-learning-based solution to optimize the CW parameter in an IEEE 802.11 network. The proposed CW optimization method considers the number of packets sent and the collision generated from each station. Simulation results show that selecting a good CW value improves the packet delivery ratio, channel access fairness, throughput, and latency. The benefits are even more significant when the queue size is less or equal to 20.

In [39], a different approach to controlling the CW value, named contention window threshold, is used. It employs deep reinforcement learning (DRL) principles to establish a threshold value and learn the optimal settings under various network scenarios. The method used is called the smart exponential-threshold-linear backoff algorithm with a deep Q-learning network (SETL-DQN). Its results demonstrate that this algorithm can reduce collisions and improve throughput.

The authors of [40] apply DRL to the problem of random access in machine-type communications networks adopting slotted ALOHA access protocols. Their proposed solution aims at finding a better transmission policy for slotted ALOHA protocols. The proposed algorithm learns a policy that establishes a trade-off between user fairness and the overall network throughput. The solution employs centralized training, which makes the learned policy equal for all users. Their approach uses binary feedback signals and past actions to learn transmission probabilities and adapt to traffic scenarios with different arrival processes. Their results show that the proposed algorithm outperforms the classical solution, i.e., the exponential backoff, in terms of user fairness and overall throughput. Our proposed solution differs from this one because it tackles the problem of random access in slot-free and contention-based networks, aiming to maximize the network's throughput and, consequently, the throughput of individual users.

Differently from the previous works, in this work, we propose leveraging an alternative observation metric that is based on the average of all stations' normalized transmission queue levels, which seems a more informative and straightforward way to understand and capture the overall network's status, to train DRL agents to take actions aiming at finding the best CW value, which in turn, optimizes the network performance. Moreover, we compare the observation metric proposed in [28] (i.e., the collision probability observed by the network) with the proposed one, i.e., the stations' averaged normalized transmission queue levels. The normalized transmission queues' level provides more information, taking into account the congestion level of the network and the behavior of all nodes, not just the individual node. It also leads to better performance, allows for adaptation to changing network conditions, and avoids myopic decisions. In contrast, using the collision probability alone can result in suboptimal decisions that do not take into account the impact on the overall network performance. Therefore, the averaged normalized transmission queues' level seems to be a more suitable metric for RL solutions to solve the collision avoidance problem in Wi-Fi networks.

The ML-based approaches presented in this section show how well ML algorithms can be applied to reach optimal performance in the wireless network field. Therefore, this motivated us to study and propose a DRL-based solution to reduce collisions by optimizing CW in different scenarios.

III. Machine Learning Overview

ML algorithms, also called models, have been widely applied to solve different problems related to optimization in wireless network communications systems [41]–[44]. These algorithms construct a model based on historical data, known as a training dataset, to perform tasks, for example, solving optimization problems, without being explicitly programmed to do so [45], [46]. ML algorithms can provide self-management, self-learning, and self-optimizing solutions for an extensive range of issues in the context of dynamic resource allocation, spectrum resource management, wireless network optimization, and so much more [47].

The learning process of an ML model is called training, and it is used for the model to gain knowledge (i.e., infer a solution) and achieve the desired result. It is possible to classify the ML model learning based on the type of its training, also called learning paradigm [48]. The learning paradigms can be classified as **Supervised learning**, **Unsupervised learning**, and **Reinforcement Learning (RL)**. Fig 2 shows the relations between the ML paradigms. Therefore, next, we provide a brief overview of these learning paradigms.

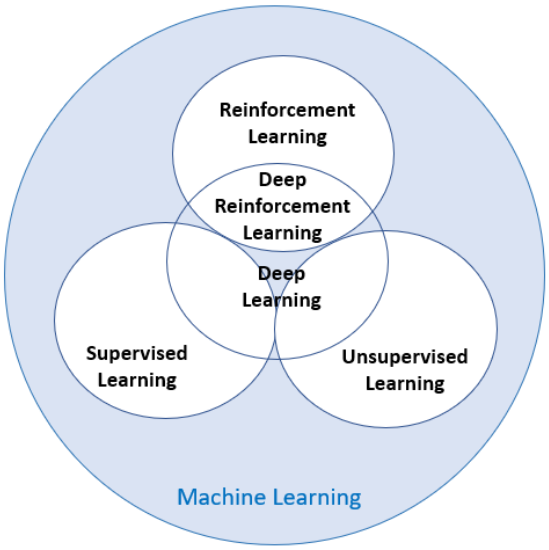

Fig. 2. ML learning paradigms based on the type of training.

A. Supervised Learning

In this paradigm, the ML model uses labeled data during the training phase. Each input sample is accompanied by its desired output sample, called a label. It is suitable for applications with plenty of historical data [49]. Some well-known algorithms following this paradigm are linear and logistic regression, support vector machine (SVM), K-nearest neighbors (KNN), and artificial neural networks (ANN).

B. Unsupervised Learning

Here in this paradigm, the ML model learns to find (sometimes hidden) useful patterns by exploring the input data without labels (i.e., without expected output values). The model is trained to create a small representation of the data [49]. This learning paradigm includes the following algorithms: K-means, isolation forest, hierarchical clustering, and expectation–maximization.

C. Reinforcement Learning

In this learning paradigm, an agent, in this context, the ML model, learns by continuously interacting with the environment and decides which action to take based on its own experience, mapping the current observed state of the environment to an action. The agent aims to learn a function known as policy in this context that models the environment and maps observed states into the best actions. The agent performs a decision-making task by trial and error in a self-learning manner [49]. This paradigm includes the following algorithms: Q-learning, Deep Q-learning, policy gradient learning, deep deterministic policy gradient, and the multi-armed bandit.

The environment is where the information (observed state and reward) is produced, and it has a dynamic nature compared to supervised and unsupervised learning paradigms. The Markov Decision Process (MDP) is generally adopted to represent the environment because it has a mathematical structure suitable for modeling decision-making problems [49]. It

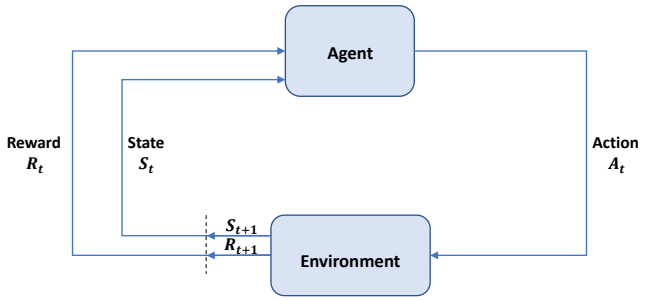

Fig. 3. An RL agent interacting with the environment.

consists of a tuple of five elements $\mathbf{M} = \{\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{P}, \gamma, \boldsymbol{R}\}$, where $\boldsymbol{S}$ is the state space, $\boldsymbol{A}$ is the action space, $\boldsymbol{P}$ is the transition probability, γ is the discount factor, and $\boldsymbol{R}$ is the reward. A reward is a positive or negative numeric value that indicates the quality of the action taken at a particular state [45]. The higher the reward, the better the action taken at that state. Conversely, the lower the reward, the worse the action. RL algorithms aim to find a policy that maximizes the total future reward. Fig 3 shows the interaction of the agent with the environment, which occurs in the following way: the agent observes (senses) the current state of the environment, S_t, based on this observation, the agent selects an action, A_t, and executes the action in the environment, this action on the environment returns information (i.e., results) in the form of reward, R_{t+1} and next-state, S_{t+1}, reached due to the action taken at the t-th time interval.

To better explain the RL learning cycle shown in Fig 3, we provide a classic, well-known RL algorithm presented in the literature, that is, the Q-learning algorithm, which has its pseudo-algorithm shown in Algorithm 1. Q-learning is a tabular RL method where the training process occurs basically in a table with the rows as the states, the columns as the actions, and each element inside the table as the Q-value. The goal of Q-learning is to find an optimal Q-value (i.e., measures the quality of the action) that maximizes the reward through iterative updates of the Q-table $Q_t(s, a)$ using the Bellman equation presented in (1) [50].

$$Q_{t+1}(s,a) = (1-\alpha)\left(Q_t(s,a)\right) + \alpha\left[r + \gamma max\left(Q_t(s',a')\right)\right], \tag{1}$$

where s is the state, a is the action, r is the reward, α is the learning rate that undertakes an interval of values between $[0, 1]$, γ is the discount rate and uses values between $[0, 1]$, s' is the next-state, a' the next action, and $max(Q_t(s', a')$ represents the possible maximum reward based on the next-state s' and next-action a'.

The basic idea of how Q-learning works after training is shown in Fig 4. For each action that is fed into the Q-table, there is a corresponding action, which is the action corresponding to the maximum Q-value for that given input state. In other words, given a line (the state), which action (the column) has the highest Q-value?

Next, we explain how the policy is learned through a process of exploration-exploitation of the environment.

Policy: is a rule that helps the agent select the best action

Algorithm 1 Q-learning

▷ Parameters initialization
Learning rate, α
Discount rate, γ
Exploration rate, ϵ, ϵ_{Min}, ϵ_{Decay}
Initial state s_0
Action space size, j
Initialize Q-table with zero for all possible (state, actions) pair $Q_t(s,a) \leftarrow 0$

for each Episode **do**
    Reset the state to its initial state $s \leftarrow s_0$

    if random.uniform (0, 1)<ϵ **then**
        $a \leftarrow$ uniform random integer in [0, j]
    else
        $a \leftarrow$ argmax($Q_t(s)$)
    end if

    After selecting the action a and applying it to the environment, a new reward r and next-state s' are returned from the environment.

    ▷ Learning stage : $Q_{t+1}(s,a)$ is updated with new q-values
    $Q_{t+1}(s,a) \leftarrow (1-\alpha)(Q_t(s,a)) + \alpha[r + \gamma max(Q_t(s',a'))]$
    $s \leftarrow s'$

    Less exploration of the environment is achieved by decreasing ϵ.
    if $\epsilon > \epsilon_{Min}$ **then**
        $\epsilon \leftarrow \epsilon \times \epsilon_{Decay}$
    end if
end for

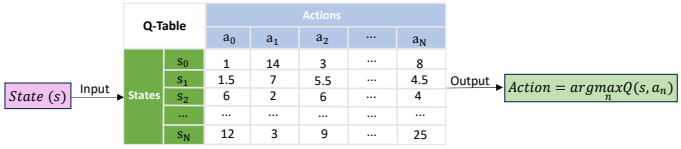

Fig. 4. Q-learning.

in a specific state. The primary objective of an RL algorithm is to learn a policy that maximizes the expected cumulative reward. The policy is a function that gives the probability of taking a given action when the environment is in a given state. The policy is learned by employing a method of exploring unknown actions in a given state and exploiting the current acquired knowledge. There must be a trade-off between exploration of the environment and exploitation of the learned policy [51]. Simple exploration-exploitation methods are the most practical and used ones. One such method is the ϵ-greedy, where $\epsilon \in [0,1]$ is a parameter controlling the amount of exploration and exploitation [52]. Normally, ϵ is a fixed hyperparameter, but it can have its value decreased so that the agent explores the environment progressively less. Eq. 2 summarizes the ϵ-greedy exploration-exploitation mechanism used by RL algorithms to learn the best policy.

$$\text{Action} = \begin{cases} \text{Random action (Exploration)}, & \text{if random number} < \epsilon, \\ \text{Best long-term action (Exploitation)}, & \text{otherwise.} \end{cases} \quad (2)$$

Exploration: in this phase, the agent randomly selects an action from a uniformly distributed random variable with the number of possible values equal to the number of actions. Non-optimal actions are chosen to explore the environment, i.e., uncharted actions in a given state.

Exploitation: in this phase, the agent selects the action with the maximum quality value for that given state, i.e., it selects the action that has the best long-term effect in maximizing the expected cumulative reward.

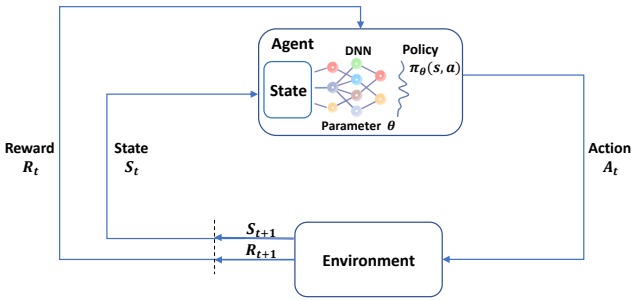

Fig. 5. DRL Structure.

D. Deep Reinforcement Learning

DRL is an improved extension of RL that integrates deep learning (DL), i.e., ANNs, with reinforcement learning algorithms [53]. This integration happens because RL algorithms present limitation problems related to state/action spaces, computational, and sample complexity [54]. That is, RL algorithms are not scalable and are limited to low-dimensional data issues, i.e., problems with a small number of actions and states [55]. Therefore, the integration with DL improves the scalability issue and makes RL algorithms support high-dimensional data tasks. Compared to conventional RL, DRL explores a large dimensional neural network to speed up convergence. Fig 5 shows the deep reinforcement learning structure. The interaction with the environment occurs in the same way as with the RL agent. The only difference is that now the agent is an ANN model. DRL includes the following algorithms: DQN, DDPG, Twin Delayed Deep Deterministic Policy Gradient (TD3), and Double Deep Q-learning Network (DDQN). This work focuses on using DQN and DDPG algorithms to assist in the optimization of CW with the primary goal of reducing node collisions while improving network performance. This way, next, we present a brief overview of these two DRL algorithms.

1) Deep Q Network: DQN is an off-policy DRL algorithm based on Q-learning with discrete action space and continuous state space [56]. It is the result of incorporating deep learning into RL since, in many practical situations, the state space is high-dimensional and cannot be solved by traditional RL algorithms, For example, Q-learning has scaling issues to larger amounts of state and action space for being a tabular method. Therefore, DQN was developed to fix the Q-learning's scaling problem. Being an off-policy algorithm means that DQN uses an experience replay memory, where the agent learns from a batch of randomly selected prior experiences instead of the most recent one [57]. This random set of past experiences mitigates the bias that might stem from the fact that some environments have a sequential nature [57]. Here the

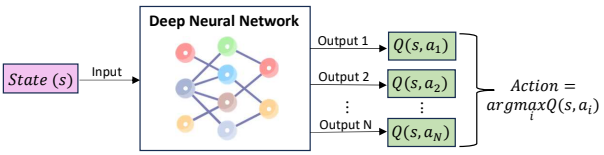

Fig. 6. Deep Q-learning (DQN).

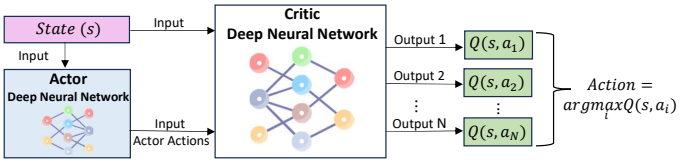

Fig. 7. Deep Deterministic Policy Gradient (DDPG).

agent is represented by a deep neural network(DNN) that uses the state as the input of the neural network, and the outputs are the Q-values corresponding to the possible actions, where the action taken by the agent is that which maximizes the Q-values, as shown in Fig 6.

The Q-value represents the quality of the action in a given state. The idea is for the neural network to output the highest Q-value for the action that maximizes the cumulative expected reward. However, using a single neural network renders training very unstable [53]. The trick to mitigate this problem and add stability to the training process is using two neural networks, predictive and target networks. They have the same structure (i.e., number of layers and neurons in each layer and activation functions) but have their weights updated at different times. The weights of the target network are not trained. Instead, they are periodically synchronized with the weights of the predictive network. The idea is that fixing the target Q values (outputs of the target network) for several updates will improve the predictive network's training stability. DQN employs batch training and experience replay memory, making the agent learn from randomly sampled batch experiences. It also employs the ϵ-greedy exploration-exploitation mechanism.

2) Deep Deterministic Policy Gradient: DDPG is another off-policy DRL algorithm with continuous action and state spaces proposed in [58]. It is the result of the combination between deterministic policy gradient (DPG) and DQN algorithms, the former related to the actor-critic algorithm [25], [59]. DQN avoids instability during the Q-function learning by employing a replay buffer and a target network. DQN has a discrete action space, while DDPG extends it to a continuous action space. The algorithm simultaneously learns a Q-function and a policy. Since DDPG inherits from the actor-critic algorithm, it is a combination of both policy (actor) and Q-value (critic) functions, where the actor takes actions according to a specific policy function, and the critic plays the role of an evaluator of the action taken [25]. Fig 7 presents a general view of how DDPG works. Here, the actor takes as input the state and outputs an action. The critic receives as input the state and the action from the actor, which are used to evaluate the actor's actions, and outputs Q-values corresponding to the set of possible actions. The Q-values outputted by the critic indicate to the agent how good the action taken by the actor for that specific state was.

DDPG consists of four networks: actor prediction, critic prediction, actor target, and critic target networks. The target networks have their weights copied from the prediction networks periodically. As with DQN, this procedure is adopted in order to stabilize the learning process, moving the unstable problem of learning the action-value function to a stable supervised learning problem [58].

Similarly to DQN, DDPG uses an experience replay memory to minimize correlations between samples. Regarding the policy aspect of exploration and exploitation, DDPG differs from DQN. Since DDPG works in continuous action space, exploring such space constitutes a significant problem. However, as it is an off-policy algorithm, the exploration problem can be treated independently from the learning algorithm [58]. DDPG creates an exploration policy that adds a noise value to the actor policy to solve this issue. By default, the noise is added following the Ornstein-Uhlenbeck process [60].

IV. Applying DRL to the CW Optimization

In order to apply DRL algorithms (DQN and DDPG) to optimize Wi-Fi networks, we propose a centralized approach to solving the CW optimization in this work. Our proposed approach consists of a centralized algorithm (i.e., the agent), which is a module running on the Wi-Fi access point (AP) that observes the state of the network (i.e., the environment) and chooses suitable CW values (i.e., the actions) to optimize the network's performance (i.e., the reward). Next, we present some details on the agent and its inputs and outputs values.

1) **Agent:** the agent represents the proposed DRL algorithms (i.e., DQN and DDPG). The agent is chosen to run on the AP since it has a general view of the whole network and then can control the stations associated with it through beacon frames in a centralized way. Therefore, as can be noticed, this is a centralized approach where the AP decides the best CW value that will be used across the network.
2) **Current state** (s): the current state of the environment is the status of all stations associated with the AP. So, it is impossible to get this information because of the nature of the problem. Therefore, we model the problem as a Partially Observable Markov Decision Process (POMDP) instead of an MDP one. POMDP assumes the environment's state cannot be perfectly observed [61].
3) **Observation** (o): is the acquired information from the network based on the averaged normalized transmission queues' level of all associated stations or on the probability of collision proposed in [28]. We will employ and compare the performance attained by these two different types of information obtained from the network. Sections IV-A and IV-B explain how each one of these observation values is calculated.
4) **Action** (a): the action corresponds to the CW value. Since the CW value is directly connected to the network performance, longer back-off periods lead to longer

waiting times for retransmitting packets in case of collisions, degrading the spectrum usage and the network performance. The RL scheme brings the idea that for every action, there is a related maximized reward. Thus, applying RL concepts to optimize the CW value aiming to maximize the network's throughput is what this work proposes. Therefore, we use the CW value as the RL action. As we compare DRL algorithms with discrete and continuous action spaces, the actions are integer values between 0 and 6 in the discrete case and real values within the interval $[0, 6]$ in the continuous case. The set of actions follows these specific values (i.e., 0 through 6) to define the back-off interval used by IEEE 802.11a for a station to retransmit its packet, establishing the limit condition of retransmission retries. Therefore, the CW value to be broadcast to the stations can be obtained through the application of (3). This interval is selected so that the action space is within 802.11 standard's CW range, which ranges from 15 up to 1023. An action, a, taken by the agent in the state, s, makes the environment switch to its next state, s', with a given transition probability, $T(s'|s, a)$. Note that the action, a, is mapped into the CW value by using equation (3). When using (3), the CW interval, which is broadcast by the AP, is in the range of 15 to 1023, according to the 802.11 standard.

$$CW = \left\lfloor 2^{a+4} \right\rfloor - 1. \tag{3}$$

5) **Reward** (r): the reward is defined as the normalized network throughput, which can be observed at the AP, by the agent, by taking action, a, in the state, s. Therefore, the reward is a real value in the interval $[0, 1]$. This normalized metric is obtained by dividing the actual throughput by its expected maximum.

A. Averaged transmission queues' level

Since the metric characterizing the environment should provide the best possible understanding of the current network's status, we propose using the averaged normalized transmission queues' level of all associated stations, $\bar{Q}_{NL}$, as observation. This metric is adopted as observation because it offers a more direct way of obtaining information about the overall network status. Next, we describe how it is calculated. Conversely, as will become clear next, the collision probability requires the AP to determine the number of transmitted and correctly received frames, which is, in turn, determined based on the number of transmitted or received acknowledgment frames.

The normalized level of the transmission queue of the i-th station, $Q_{NL}^{(i)}$, is calculated according to (4), where (i) indicates the station number. The queue level, $Q_l^{(i)}$, is normalized (i.e., divided) by the maximum queue size value of that station, $Q_{max}^{(i)}$.

$$Q_{NL}^{(i)} = \frac{Q_l^{(i)}}{Q_{max}^{(i)}}. \tag{4}$$

The measurement of $Q_{NL}^{(i)}$ is carried out at predefined intervals at each station and indicates the result of the currently chosen CW value on the network's performance. For example, a value close to 1 indicates the queue is full, meaning the station cannot transmit packets as quickly as it receives them from the upper layers. On the other hand, if it is close to 0, the queue is almost empty, indicating the station can access the medium as frequently as necessary. A high $Q_{NL}^{(i)}$ value indicates a high number of collisions. Conversely, a low value indicates a small number of collisions. The normalized level of the transmission queue of each station, $Q_{NL}^{(i)}$, is concatenated (i.e., piggybacked) to data frames sent to the AP so that the agent has access to this information. At the AP, the agent normalizes the sum of $Q_{NL}^{(i)}$ coming from the stations by the total number of stations associated with the AP, $N_{stations}$, as shown in (5). This is the observation used by the agent to gather insights into the network's status.

$$\bar{Q}_{NL} = \frac{1}{N_{stations}} \sum_{i=1}^{N_{stations}} Q_{NL}^{(i)}. \tag{5}$$

B. Collision Probability

Another metric characterizing the environment, proposed in [28], is the probability of collision, p_{col}, observed by the network. It can also be interpreted as the probability of transmission failure. This probability is calculated based on the number of transmitted, N_t, and correctly received, N_r, frames, as shown in (6).

$$p_{col} = \frac{N_t - N_r}{N_t}. \tag{6}$$

This collision rate approximates the actual probability of collision as the number of frames used to calculate it increases. Thus, this rate represents the probability of a frame not being received due to another station transmitting a frame at the same time. These probabilities are calculated within the interaction periods and provide information on the performance of the selected CW value.

C. Centralized DRL-based CW Optimization Method

The proposed method has three stages. The first is a pre-learning stage, where the legacy Wi-Fi contention-based mechanism manages the network. This stage is used to initialize the observation buffer with information that will be used to train the DRL algorithm being used (either DQN or DDPG). Next, in the learning stage, the agent chooses CW values (i.e., actions) according to what is shown in Algorithm 2.

The mean, μ, and variance, σ^2, of the history of recent observation values (either the averaged normalized queues' level or the collision probability) are calculated as a preprocessing step. Moving average with window and stride of fixed sizes is used to calculate both statistics. This calculation renders the observation into a two-dimensional vector for each stride of the moving average. Therefore, the agent is trained based on this two-dimensional vector of observations as illustrated in Fig 8.

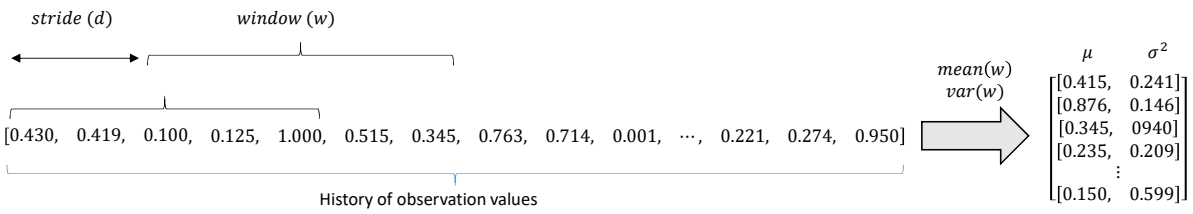

Fig. 8. Bi-dimensional observation vector used in the preprocessing phase.

Exploration of the environment is enabled by adding a noisy factor to each action the agent takes. This noisy factor decreases throughout the learning stage. This addition of noise is different for each of the two considered DRL algorithms. When DQN is used, the noisy factor corresponds to the probability of taking a random action instead of an action predicted by the agent. In DDPG's case, the noisy factor comes from a Gaussian-distributed random variable and is added directly to the action taken by the agent. As mentioned, this is done to find a trade-off between exploring the environment and exploiting the acquired knowledge, which chooses the action that maximizes future rewards.

The last stage is called the operational stage. This stage starts when the training is over. The user defines the training stage's period by setting the variable $trainingPeriod$. At this stage, the noisy factor is set to zero, so the agent always chooses the action it learned to maximize the reward. At this stage, as the agent has already been trained, it does not receive any additional updates to its policy, so rewards are unnecessary.

Finally, it is essential to mention that the hyperparameters of the DRL models (i.e., learning rate, reward discount rate, batch size, epsilon decay) need to be fine-tuned for the agent to achieve its optimal performance. Lastly, as both DQN and DDPG employ replay memory (called in this work as the experience replay buffer, E), a size limit has to be configured for this memory. The replay memory stores past interactions of the agent with the environment, i.e., it records the current state, the action taken at that state, the reward received in that state, and the next state resulting from the action taken. When its limit is reached, the oldest record is overwritten by a new one (i.e., it is implemented as a circular buffer).

A brief description of the pseudo-code shown in Algorithm 2 is provided next. The initialization phase goes from lines 1 to 13. It initiates the input parameters such as observation and replay buffers, agent weights, initial state, noisy factor, CW value, and variables used to select the current stage and the type of observation employed.

Then, the pre-learning stage (lines 15 to 24) starts by filling in the observation buffer with the selected type of observation metric (either the averaged normalized queues' level or the collision probability). The flag $useQueueLevelFlag$ sets the type of observation value to be calculated. In this stage, the observation value (either the averaged normalized transmission queues' level or the network's collision probability) results from the application of the legacy Wi-Fi contention-based mechanism. After $envStepTime$, which is the period between interactions of the proposed algorithm with the environment, has elapsed, the algorithm selects the CW value, and then, the observation value results from the application of the DRL-based CW optimization algorithm. Therefore, one should note that the instructions in lines 15 to 24 are shared between the pre-learning and learning stages.

Next, from lines 26 to 29, the information in the observation buffer is pre-processed by applying a moving average operation to the data, which results in a two-dimensional vector, $observation$, with the mean, μ, and variance, σ^2 of the data in the buffer. With the pre-processed data, the action function, A_θ, returns the action, a. The action is then used to determine a new CW value that will be broadcast to all associated stations. The instructions in lines 26 to 29 are shared between the learning and operational stages. The difference between these two stages is how the action, a, is selected. In the learning stage, the $trainingFlag$ is equal to True, which tells the action function, A_θ, to choose actions following the exploration-exploitation approach described previously. In the operational stage, the $trainingFlag$ is equal to False, which tells the action function, A_θ, to choose actions optimizing the network's performance. In the learning stage, a noisy factor is used to explore the environment, which does not happen in the operational stage.

After that, from lines 31 to 38, the algorithm is solely in the learning stage. In this section of the algorithm, the DRL agent learns from previous experiences; that is, it has its weights, θ, updated by using mini-batches of samples randomly picked from the replay buffer, E. These mini-batches are composed of the current μ, σ^2, and action, a, values, the reward (i.e., the normalized throughput), r, and the previous values of μ and σ^2.

Finally, in line 42, the algorithm checks if the training period has elapsed. If so, the $trainingFlag$ is set to False, and the algorithm enters the operational stage. In this stage, the noisy factor is disabled, and the algorithm only exploits the acquired knowledge.

D. Experimentation Scenario

The proposed centralized DRL-based CW optimization solution is implemented on NS3-gym [30], which runs on top of the NS-3 simulator [29]. NS3-gym enables the communication between NS-3 (c++) and OpenAI gym framework (python) [62]. NS-3 is a network simulator based on discrete events mainly intended for academic research. It contains the implementation of several wired and wireless network standards [29]. In this work, we use version NS-3.29 of the NS-3 simulator. The DRL algorithms used here were implemented with TensorFlow and PyTorch.

Algorithm 2 DRL-based CW Optimization

▷ ### Initialization ###
Initialize the observation buffer, O, with zeroes
Initialize the weights, θ, of the agent
Get the action function, A_θ, which the agent uses to choose the action according to the current stage
Initialize the algorithm's interaction period with the environment, $envStepTime$
Initialize the number episodes, $N_{episodes}$
Initialize the number of steps per episode, N_{spe}
Initialize the training stage period, $trainingPeriod$
Set *trainingFlag* $\leftarrow True$ to tell the algorithm is in the training stage
Initialize the experience replay buffer, E, with zeroes.
trainingStartTime $\leftarrow$ *currentTime*
lastUpdate $\leftarrow$ *currentTime*
$\mu_{prev} \leftarrow 0$ (previous mean value)
$\sigma^2_{prev} \leftarrow 0$ (previous variance value)
Set $useQueueLevelFlag \leftarrow True$ to use the averaged normalized transmission queues' level as observation.
CW $\leftarrow 15$

for $e = 1, ..., N_{episodes}$ **do**
    Reset and run the environment, i.e., reset and run the NS-3 simulator
    for $t = 1, ..., N_{spe}$ **do**

▷ ### Pre-learning stage ###
        if $useQueueLevelFlag$ == $True$ **then**
            Get $Q^{(i)}_{NL}$ received from each associated station
            Calculate $\bar{Q}_{NL}$
            observation $\leftarrow \bar{Q}_{NL}$
        else
            $N_t \leftarrow$ get number of transmitted frames
            $N_r \leftarrow$ get number of received frames
            observation $\leftarrow \frac{N_t - N_r}{N_t}$
        end if
        O.append(*observation*)

        if $currentTime \geq lastUpdate + envStepTime$ **then**

▷ ### Learning and operational stages ###
            $\mu, \sigma^2 \leftarrow preprocess(O)$
            $a \leftarrow A_\theta(\mu, \sigma^2, trainingFlag)$
            $CW \leftarrow 2^{a+4} - 1$
            Broadcast the new CW value to all associated stations

            if $trainingFlag$ == $True$ **then**
                $N_{RP} \leftarrow$ get the number of received packets.
                $tput \leftarrow \frac{N_{RP}}{envStepTime}$
                $r \leftarrow normalize(tput)$
                E.append((μ, σ^2, a, r, μ_{prev}, σ^2_{prev}))
                $\mu_{prev} \leftarrow \mu$
                $\sigma^2_{prev} \leftarrow \sigma^2$
                $mb \leftarrow$ get random mini-batch from E
                Update θ based on mb
            end if

            lastUpdate $\leftarrow$ *currentTime*
        end if

▷ ### Makes the transition between learning and operational stages ###
        if $currentTime \geq trainingStartTime + trainingPeriod$ **then**
            $trainingFlag \leftarrow False$
        end if
    end for
end for

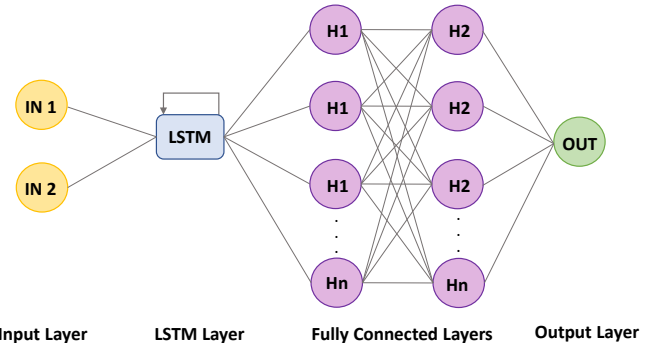

Fig. 9. Architecture of the Deep Learning Network.

The system model considered in this work is depicted in Fig 10. We consider a linear topology comprised of one AP and several stations transmitting packets. The AP plays the role of the DRL agent, selecting a new CW value according to the current observation. The stations send data packets to the AP, and the deployment of the stations can happen statically or dynamically. So, two scenarios are considered, one with static topology and the other with dynamic topology.

Table I presents the NS-3 parameters necessary to create the environment in which the agent will learn. Apart from those parameters, we assume single-user transmissions, a packet load adjusted to saturate the network with constant bit rate (CBR) UDP traffic of 150 Mbps, instant and faultless transference of network information, e.g., the normalized queue level of each station, $Q^{(i)}_{NL}$, or the number of transmitted packets, N_t, of each station, to the DRL agent, and that each station receives the selected CW value instantly. The last two assumptions allow the assessment of the proposed solution in an idealized scenario before going to more realistic ones. Realistic scenarios will require the transmission of $Q^{(i)}_{NL}$ or N_t from each station to the AP and the periodic broadcast of the chosen CW to all stations through beacon frames. The only differences we foresee in the results presented here are a slower convergence of the agent and a slightly smaller throughput due to the transmission of the required overhead, i.e., $Q^{(i)}_{NL}$ or N_t and CW.

Table II presents the agent parameters used in NS3-gym for the experiments. These parameters were empirically found through several simulations. The network architecture used by both DRL algorithms has one recurrent long short-term memory (LSTM) layer and two fully connected layers leading to an $8 \times 128 \times 64$ topology. An illustration of the network architecture is shown in Fig 9

The simulation experiment was executed for 15 episodes of 60 seconds. This configuration was also found by employing cross-validation strategies. The recurrent long short-term memory layer allows the DRL algorithms to consider past observations when predicting the best action in a given state. The moving average window is half the size of the observation history memory, and the stride is a fourth of its size.

Each one of the experiments consisted of 15 executions of 60-second long simulations, where the first 14 executions were part of the training stage (thus, the training stage period,

TABLE I
NS-3 ENVIRONMENT CONFIGURATION PARAMETERS.

Configuration Parameter	Value
Wi-Fi standard	IEEE 802.11ax
Number of APs	1
Number of static stations	5,15, 30 or 50
Number of dynamic stations	increases steadily from 5 to 50
Frame aggregation	disabled
Packet size	1500 bytes
Max Queue Size	100 packets
Frequency	5 GHZ
Channel BW	20 MHz
Traffic	constant bit-rate UDP of 150 Mbps
MCS	HeMcs (1024-QAM with a 5/6 coding rate)
Guard Interval	800 [ns]
Propagation delay model	ConstantSpeedPropagationDelayModel
Propagation loss model	MatrixPropagationLossModel
Simulation time	60 [s]

TABLE II
NS3-GYM AGENT CONFIGURATION PARAMETERS.

Configuration Parameter	Value
DQN's learning rate	$4x10^{-4}$
DDPG's actor learning rate	$4x10^{-4}$
DDPG's critic learning rate	$4x10^{-3}$
Reward discount rate	0.7
Batch size	32
Replay memory size	18,000
Size of observation history memory	300
$trainingPeriod$	840 [s]
$envStepTime$ (i.e., interaction interval)	10 [ms]

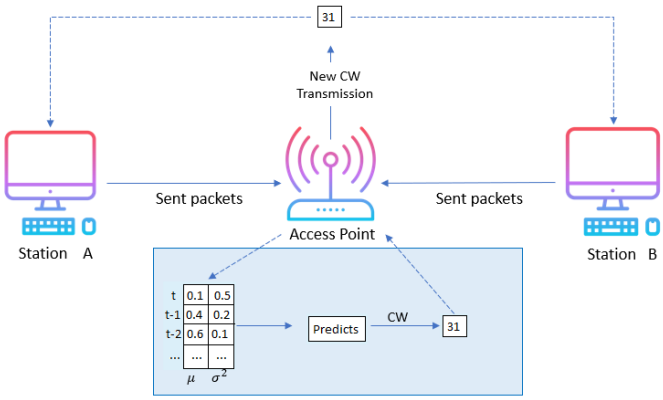

Fig. 10. Scenario for assessing the proposed centralized DRL-based CW optimization method.

$trainingPeriod$, is equal to $14 \times 60[s] = 840[s]$), and the last one was the operational stage. Each one of the simulations consisted of 10 [ms] interaction intervals (i.e., $envStepTime$). Algorithm 2 was run between these interaction intervals.

V. SIMULATION RESULTS

This section presents and discusses the results obtained during the experiments. The performance of the proposed DRL algorithms is compared against the BEB algorithm, which is used in 802.11 wireless networks. Simulations were executed on NS-3 and NS3-gym simulators considering static and dynamic scenarios. The graphical results allow a better evaluation of the network efficiency achieved by the proposed centralized DRL-based CW optimization method in both scenarios. Next, we separate the results and discussion into two sets, static and dynamic. Our study considers and compares the averaged normalized transmission queues' level of all stations and the probability of collision as two different types of observation metrics to train the DRL-based CW optimization agent. The simulation results presented in this section demonstrate the performance achieved by the DRL-based solution employing either one of the observations in the previously mentioned scenarios.

A. Metrics and their connection with the reward function

In this work, we assess two metrics: the network's overall throughput and the CW value. The network's overall throughput measures how much data is successfully transmitted over a period over the network. The CW value is a random parameter that determines how long a station must wait before it can transmit data. Therefore, one can see that a longer CW value means that stations must wait longer before transmitting, which can lead to lower throughput. However, a longer CW value means a lower chance of collisions. On the other hand, the smaller the CW value, the shorter the waiting time and, consequently, the higher the chance of collisions, which is detrimental to the network's overall throughput.

The DRL agent employs the CW values as actions and the average of the normalized level of the transmission queues of the stations or the collision rate as observations. Its reward

function is the normalized network's overall throughput. The agent's objective is maximizing its received reward over time based on observations of the environment. Therefore, the reward's connection with the throughput is straightforward, i.e., the higher the reward, the higher the network's overall throughput.

The reward's connection with the CW value is also direct if we understand that the longer the CW value, the lower the throughput. It happens because a station cannot transmit during the CW wait time, penalizing its throughput since it is calculated during a given period. On the other hand, the higher the CW value, the lower the chance of collisions, and vice versa. Therefore, the DRL agent aims to find a CW value that maximizes the network's overall throughput while minimizing frame collisions., i.e., a trade-off between throughput and collisions.

B. Static scenario

In this scenario, the number of stations associated with the AP is kept constant throughout the experiment. As it is a static scenario, the optimal CW value should be constant, and so should the throughput. Therefore, this scenario is used to prove this hypothesis and to evaluate possible improvements over 802.11's BEB algorithm.

Fig 11 shows the throughput achieved by the network for different numbers of stations. As can be seen, the network throughput decreases as the number of stations increases when BEB is employed. On the other hand, when either DQN or DDPG is used with either one of the observation metrics, it remains practically constant as the number of stations increases, proving the hypothesis. In this scenario, the graph shows that the throughput values achieved by our DRL-based solution with four different numbers of stations (5, 15, 30, 50) are very similar when using either one of the metrics. This is indicated by the matching points on the graph, demonstrating a close value of throughput. The improvement over BEB varies from 5.19% for 5 stations to 48.5% for 50 stations. As can be seen, DDPG has a slightly better performance than DQN (with either one of the observation metrics), which can be explained due to its capability to choose any real CW value within the $[0, 6]$ range. In the static scenario, the performance achieved by the solution employing either one of the two observation metrics shows a minimal difference. When compared, there is no significant variance in their behavior.

Fig 12 shows the mean CW value and its variance for 15 simulation episodes when DQN or DDPG is used with either one of the two observation metrics considered. This experiment considers 30 stations in the static scenario. The mean CW is the arithmetic mean of all CW values selected during one episode and serves as an indicator of how well the training agents can adjust and select the CW values to a more stable one. It shows that as the agent learns, it exploits more of the acquired/learned knowledge than explores the environment with random actions. Therefore, as the DRL agent learns, the number of random actions decreases, decreasing the variance of the CW values. It is pretty clear that the 14 episodes selected for the learning stage are enough for the

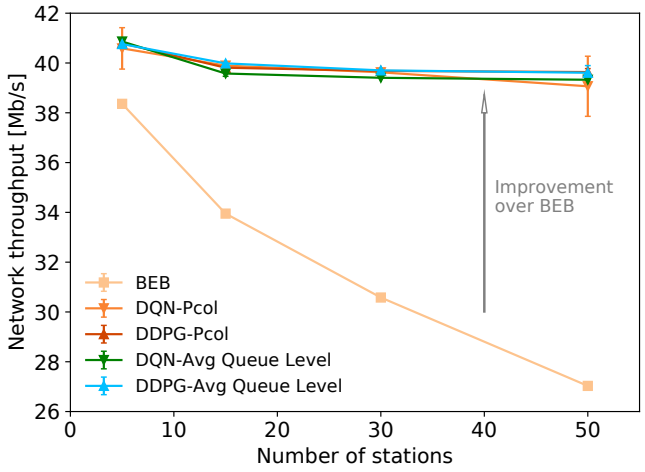

Fig. 11. Comparison of the network throughput for the static scenario.

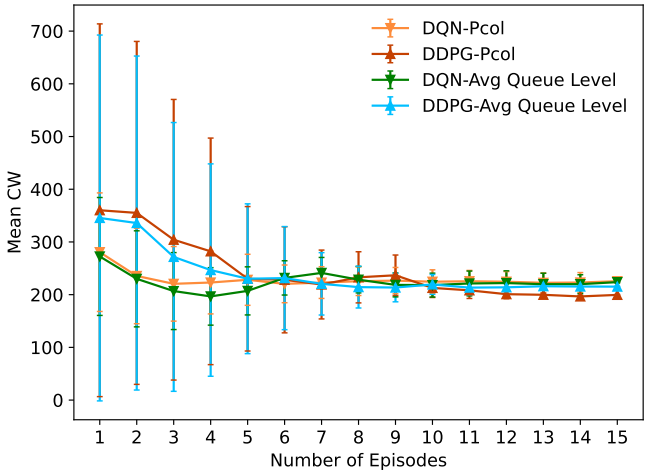

Fig. 12. Mean CW value for 30 stations in the static scenario.

proposed solution to converge to an optimal and practically constant mean CW value for both DRL algorithms, regardless of the observation metric considered. We can see that the mean value of all considered options (DQN/DDPG and Average Tx Queue level/Pcol) stabilizes around the same value after the 10^{th} episode. It is also possible to see that the CW variance of all options decreases along the learning stage, meaning that initially, the algorithm explores the environment more (i.e., it takes uncharted actions). Then, as the number of episodes progresses, it exploits more of the acquired knowledge, which maximizes the received reward. Finally, the variance during the final episodes of the learning stage is small because the proposed algorithm correctly selected the CW value, which maximizes the throughput. In the final episodes, where the algorithm exploits more than explores the environment, DDPG presents a smaller variance than DQN, meaning it is more assertive in choosing the ideal CW value.

C. Dynamic scenario

In the dynamic scenario, the number of stations increases progressively throughout the experiment, going from 5 to 50. The higher the number of stations, the higher the collision

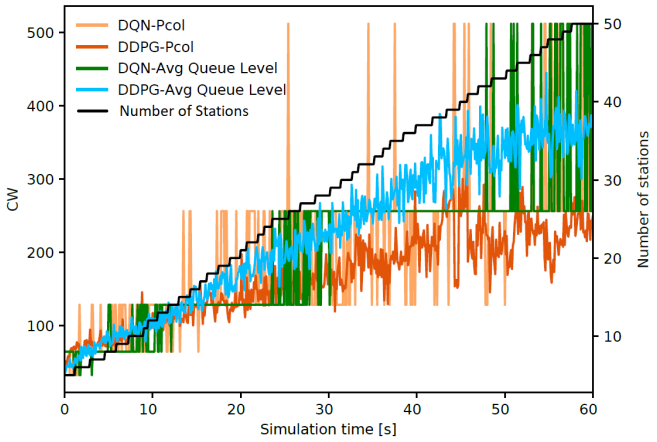

Fig. 13. Selected CW value for different numbers of stations in a dynamic scenario.

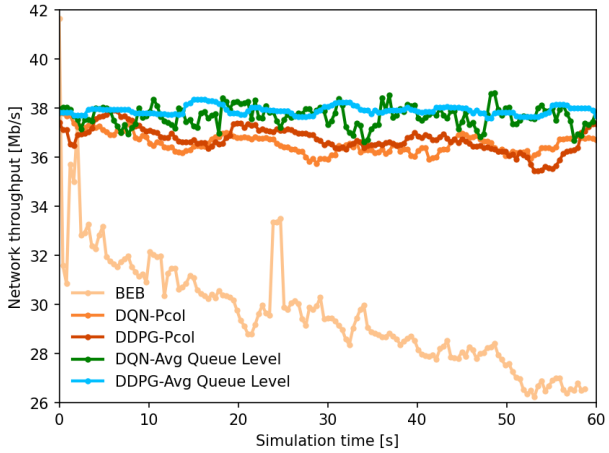

Fig. 14. Comparison of the instantaneous network throughput as the number of stations increases from 5 to 50.

probability. This experiment assessed whether the DRL algorithms appropriately act upon network changes.

Fig 13 depicts the chosen CW value for the dynamic scenario, where the number of stations progressively increases from 5 to 50. As shown, the chosen CW value increases as the number of stations increases, meaning the back-off interval has to be increased to accommodate the transmissions of the higher number of stations, mitigating the number of collisions. As can be noticed, DQN jumps between discrete neighbor CW values. At the same time, DDPG continuously increases the CW value, reaching a lower CW value for 50 stations independently of the adopted observation metric, which positively reflects on the attained throughput. When comparing DQN for both observation metrics, it is possible to see a similarity in selecting the adequate CW value. Furthermore, DDPG with the averaged normalized transmission queues' level as observation presents higher CW values than when the probability of collision is used. This behavior, which is supported by the results in Fig 14, can be partially explained by the fact that the normalized transmission queues' level of all associated stations offers more information than just the collision probability. This metric takes into account the congestion level of the network, and subsequently helps the DRL-based solution select a CW value that allows the network to achieve higher throughput. This insight suggests that the use of this metric in the RL solution for collision avoidance could result in improved network performance.

Fig 14 compares the instantaneous network throughput in the dynamic scenario when the number of stations grows from 5 to 50. The increased number of stations alters the CW value, impacting the instantaneous network throughput. When the number of stations associated with the AP reaches 50, the throughput of the BEB drops to approximately 26 Mbps of that presented by DQN and DDPG. The proposed DRL algorithm (with either DQN or DDPG) presents an almost constant behavior, keeping a high and stable throughput as the number of stations progressively increases. Both DRL algorithms present an approximately constant throughput. Comparing the two observation metrics reveals a difference in the network throughput. Using the collision probability as the observation, the resulting throughput is approximately 37.5 Mbps, presenting an increase of 43.68% over BEB, whereas using the averaged normalized transmission queues' level yields a throughput of up to 37.98 Mbps with an increase of 45.52% over BEB. This difference suggests that the averaged normalized transmission queues' level is a superior observation metric for dynamic scenarios. However, it is essential to note that both observation metrics achieve significantly better throughput values than the standard BEB algorithm. This finding underscores the potential value of utilizing RL solutions for collision avoidance in Wi-Fi networks.

Fig 15 confirms that both DRL algorithms significantly enhance the network's throughput in comparison to the BEB algorithm in dynamic scenarios. The degree of improvement varies from 6.84% (using either observation metric) for 5 stations, to 25.56% for the collision probability metric when 50 stations are in use. Notably, when the averaged normalized transmission queues' level is employed for 50 stations, the throughput is enhanced by as much as 28.43%, thereby indicating the superior performance of this observation metric over the collision probability in dynamic scenarios with 50 stations. These findings demonstrate once more the potential value of using DRL algorithms to address collision avoidance challenges in dense Wi-Fi networks.

VI. Conclusions

Wireless network transmissions are prone to various impairments (e.g., interference, path loss, channel noise, etc.) that lead to packet loss and collisions, making re-transmission and channel access mechanisms required. Furthermore, in environments with a dense number of stations, more collisions will occur while the stations attempt to access the wireless channel. Consequently, the network efficiency and channel utilization will both degrade. This work proposes a centralized solution that employs DRL algorithms (i.e., DQN and DDPG) to optimize the CW parameter from the MAC layer. Regarding the number of stations associated with the AP, two experimental scenarios are considered for assessing the

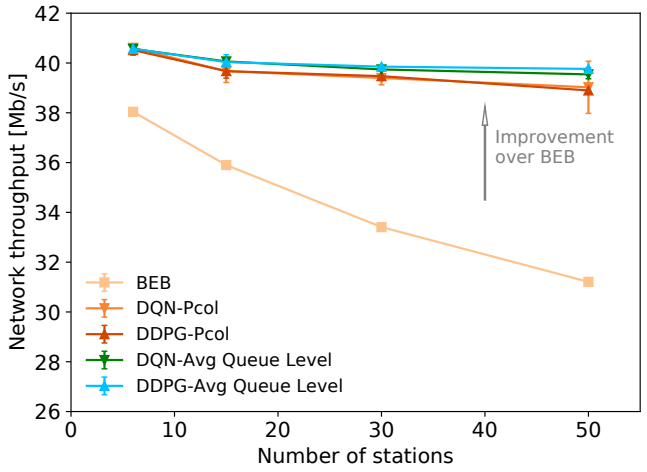

Fig. 15. Comparison of the network throughput for the dynamic scenario.

proposed centralized DRL-based CW optimization solution: static and dynamic. Simulation results show that the proposed solution outperforms the 802.11 default BEB algorithm by maintaining a stable throughput while reducing collisions. Moreover, the results attest to DQN's and DDPG's superior performance compared to BEB for both scenarios, regardless of the observation metric used and the number of stations associated with the AP. Our results show that the difference increases as the number of stations increased, with DQN and DDPG showing a 45.52% increase in throughput with 50 stations compared to BEB.

Additionally, the results show that DQN and DDPG had similar performances, which means that either could be used in a solution deployed in Wi-Fi APs. However, since DQN presents a lower computational complexity and lower training period, it would be the preferable choice. Furthermore, the presented results show that the network's performance can be dramatically improved when CW is chosen based on information from the network, such as the level of the transmission queues or the network's probability of collision.

Regarding the two considered observation metrics used for capturing the network's status, our simulation results show that in the static scenario, they confer similar performance when applied to the DRL-based algorithms. However, in the dynamic case, the averaged normalized transmission queues' level grants the algorithms higher throughput when compared to the observation based on the collision probability. These findings suggest that the averaged normalized transmission queues' level provides more information to the DRL agent than the collision probability. Moreover, the centralized DRL-based solution for selecting CW values outperformed the classical 802.11 BEB algorithm, regardless of which observation metric was used. Therefore, it can be concluded that either observation metric is suitable for obtaining the network's status and improving the CW selection, which ultimately leads to better network utilization and higher throughput.

Future work could use other ML algorithms to optimize CW, such as Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO). These two algorithms make an interesting topic of investigation because they both use advantage instead of Q-Value as the operator. The difference between them is that SAC is off-policy, and PPO is on-policy. In addition, a study that included these methods could be interesting because DQN and DDPG are both off-policy and use a Q-value operator, which would contrast with SAC and PPO, potentially providing precious insights into the effect of these different operators in the final result. Moreover, another research direction would be the assessment of decentralized DRL-based solutions.

REFERENCES

[1] X. Guo, S. Wang, H. Zhou, J. Xu, Y. Ling, and J. Cui, "Performance evaluation of the networks with wi-fi based tdma coexisting with csma/ca," *Wireless Personal Communications*, vol. 114, no. 2, pp. 1763–1783, 2020. doi: 10.1007/s11277-020-07447-3

[2] W. Auzinger, K. Obelovska, I. Dronyuk, K. Pelekh, and R. Stolyarchuk, "A continuous model for states in csma/ca-based wireless local networks derived from state transition diagrams," in *Proceedings of International Conference on Data Science and Applications*. Springer, 2022. doi: 10.1007/978-981-16-5348-3_45 pp. 571–579.

[3] G. Wang and Y. Qin, "Mac protocols for wireless mesh networks with multi-beam antennas: A survey," in *Future of Information and Communication Conference*. Springer, 2019. doi: 10.1007/978-3-030-12388-8_9 pp. 117–142.

[4] Z. Beheshtifard and M. R. Meybodi, "An adaptive channel assignment in wireless mesh network: the learning automata approach," *Computers & Electrical Engineering*, vol. 72, pp. 79–91, 2018. doi: 10.1016/j.compeleceng.2018.09.004

[5] S.-C. Wang and A. Helmy, "Performance limits and analysis of contention-based ieee 802.11 mac," in *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*. IEEE, 2006. doi: 10.1109/LCN.2006.322129 pp. 418–425.

[6] M. Yazid, N. Sahki, L. Bouallouche-Medjkoune, and D. Aïssani, "Modeling and performance study of the packet fragmentation in an ieee 802.11 e-edca network over fading channel," *Multimedia Tools and Applications*, vol. 74, no. 21, pp. 9507–9527, 2015. doi: 10.1007/s11042-014-2131-y

[7] P. Patel and D. K. Lobiyal, "A simple but effective collision and error aware adaptive back-off mechanism to improve the performance of ieee 802.11 dcf in error-prone environment," *Wireless Personal Communications*, vol. 83, pp. 1477–1518, 2015. doi: 10.1007/s11277-015-2460-9

[8] ——, "An adaptive contention slot selection mechanism for improving the performance of ieee 802.11 dcf," *International Journal of Information and Communication Technology*, vol. 10, no. 3, pp. 318–349, 2017. doi: 10.1504/IJICT.2017.083272

[9] R. A. da Silva and M. Nogueira, "Mac protocols for ieee 802.11 ax: Avoiding collisions on dense networks," *arXiv preprint arXiv:1611.06609*, 2016. doi: 10.48550/arXiv.1611.06609

[10] Y. Edalat and K. Obraczka, "Dynamically tuning ieee 802.11's contention window using machine learning," in *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2019. doi: 10.1145/3345768.3355920 pp. 19–26.

[11] E. Bjornson and P. Giselsson, "Two applications of deep learning in the physical layer of communication systems [lecture notes]," *IEEE Signal Processing Magazine*, vol. 37, no. 5, pp. 134–140, 2020. doi: 10.1109/MSP.2020.2996545

[12] H. Anouar and C. Bonnet, "Optimal constant-window backoff scheme for ieee 802.11 dcf in single-hop wireless networks under finite load conditions," *Wireless Personal Communications*, vol. 43, no. 4, pp. 1583–1602, 2007. doi: 10.1145/1164717.1164765

[13] M. A. Bender, J. T. Fineman, S. Gilbert, and M. Young, "How to scale exponential backoff: Constant throughput, polylog access attempts, and robustness," in *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2016. doi: 10.1137/1.9781611974331.ch47 pp. 636–654.

[14] H. Al-Ammal, L. A. Goldberg, and P. MacKenzie, "Binary exponential backoff is stable for high arrival rates," in *Annual Symposium on Theoretical Aspects of Computer Science*. Springer, 2000. doi: 10.1007/3-540-46541-3_14 pp. 169–180.

[15] M. Kulin, T. Kazaz, E. De Poorter, and I. Moerman, "A survey on machine learning-based performance improvement of wireless networks: Phy, mac and network layer," *Electronics*, vol. 10, no. 3, p. 318, 2021. doi: 10.3390/electronics10030318

[16] C. Silvano, D. Ielmini, F. Ferrandi, L. Fiorin, S. Curzel, L. Benini, F. Conti, A. Garofalo, C. Zambelli, E. Calore *et al.*, "A survey on deep learning hardware accelerators for heterogeneous hpc platforms," *arXiv preprint arXiv:2306.15552*, 2023. doi: 10.48550/arXiv.2306.15552

[17] Q. Cai, C. Cui, Y. Xiong, W. Wang, Z. Xie, and M. Zhang, "A survey on deep reinforcement learning for data processing and analytics," *IEEE Transactions on Knowledge amp; Data Engineering*, vol. 35, no. 05, pp. 4446–4465, may 2023. doi: 10.1109/TKDE.2022.3155196

[18] G. Aridor, Y. Mansour, A. Slivkins, and Z. S. Wu, "Competing bandits: The perils of exploration under competition," *arXiv preprint arXiv:2007.10144*, 2020. doi: 10.48550/arXiv.2007.10144

[19] X. Lu, B. V. Roy, V. Dwaracherla, M. Ibrahimi, I. Osband, and Z. Wen, "Reinforcement learning, bit by bit," *Foundations and Trends® in Machine Learning*, vol. 16, no. 6, pp. 733–865, 2023. doi: 10.1561/2200000097

[20] S. Shekhar, A. Bansode, and A. Salim, "A comparative study of hyper-parameter optimization tools," in *2021 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*. Los Alamitos, CA, USA: IEEE Computer Society, dec 2021. doi: 10.1109/CSDE53843.2021.9718485 pp. 1–6.

[21] P. I. Frazier, "A tutorial on bayesian optimization," *arXiv preprint arXiv:1807.02811*, 2018. doi: 10.48550/arXiv.1807.02811

[22] A. Karras, C. Karras, N. Schizas, M. Avlonitis, and S. Sioutas, "Automl with bayesian optimizations for big data management," *Information*, vol. 14, no. 4, p. 223, 2023. doi: 10.3390/info14040223

[23] P. Beneventano, P. Cheridito, R. Graeber, A. Jentzen, and B. Kuckuck, "Deep neural network approximation theory for high-dimensional functions," *arXiv preprint 2112.14523*, 2021. doi: 10.48550/arXiv.2112.14523

[24] J. Zhu, F. Wu, and J. Zhao, "An overview of the action space for deep reinforcement learning," in *Proceedings of the 2021 4th International Conference on Algorithms, Computing and Artificial Intelligence*, 2021. doi: 10.1145/3508546.3508598 pp. 1–10.

[25] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in neural information processing systems*, vol. 12, 1999.

[26] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[27] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv.1509.02971*, 2015. doi: 10.48550/arXiv.1509.02971

[28] W. Wydmański and S. Szott, "Contention window optimization in ieee 802.11 ax networks with deep reinforcement learning," in *2021 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2021. doi: 10.1109/WCNC49053.2021.9417575 pp. 1–6.

[29] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and tools for network simulation*. Springer, 2010, pp. 15–34.

[30] P. Gawłowicz and A. Zubow, "Ns-3 meets openai gym: The playground for machine learning in networking research," in *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2019. doi: 10.1145/3345768.3355908 pp. 113–120.

[31] A. H. Y. Abyaneh, M. Hirzallah, and M. Krunz, "Intelligent-cw: Ai-based framework for controlling contention window in wlans," in *2019 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*. IEEE, 2019. doi: 10.1109/DySPAN.2019.8935851 pp. 1–10.

[32] Y. Xiao, M. Hirzallah, and M. Krunz, "Distributed resource allocation for network slicing over licensed and unlicensed bands," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2260–2274, 2018. doi: 10.1109/JSAC.2018.2869964

[33] A. Kumar, G. Verma, C. Rao, A. Swami, and S. Segarra, "Adaptive contention window design using deep q-learning," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021. doi: 10.1109/ICASSP39728.2021.9414805 pp. 4950–4954.

[34] X. Fu and E. Modiano, "Learning-num: Network utility maximization with unknown utility functions and queueing delay," in *Proceedings of the Twenty-second International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, 2021. doi: 10.48550/arXiv.2012.09222 pp. 21–30.

[35] I. A. Qureshi and S. Asghar, "A genetic fuzzy contention window optimization approach for ieee 802.11 wlans," *Wireless Networks*, vol. 27, no. 4, pp. 2323–2336, 2021. doi: 10.1007/s11276-021-02572-8

[36] T. K. Saini and S. C. Sharma, "Prominent unicast routing protocols for mobile ad hoc networks: Criterion, classification, and key attributes," *Ad Hoc Networks*, vol. 89, pp. 58–77, 2019. doi: 10.1016/j.adhoc.2019.03.001

[37] S. Giannoulis, C. Donato, R. Mennes, F. A. de Figueiredo, I. Jabandžic, Y. De Bock, M. Camelo, J. Struye, P. Maddala, M. Mehari *et al.*, "Dynamic and collaborative spectrum sharing: The scatter approach," in *2019 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*. IEEE, 2019. doi: 10.1109/DySPAN.2019.8935774 pp. 1–6.

[38] N. Zerguine, M. Mostefai, Z. Aliouat, and Y. Slimani, "Intelligent cw selection mechanism based on q-learning (misq)." *Ingénierie des Systèmes d Inf.*, vol. 25, no. 6, pp. 803–811, 2020. doi: 10.18280/isi.250610

[39] C.-H. Ke and L. Astuti, "Applying deep reinforcement learning to improve throughput and reduce collision rate in ieee 802.11 networks," *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 16, no. 1, pp. 334–349, 2022. doi: 10.3837/tiis.2022.01.019

[40] M. A. Jadoon, A. Pastore, M. Navarro, and F. Perez-Cruz, "Deep reinforcement learning for random access in machine-type communication," in *2022 IEEE Wireless Communications and Networking Conference (WCNC)*, 2022. doi: 10.1109/WCNC51071.2022.9771953 pp. 2553–2558.

[41] R. Mennes, F. A. P. De Figueiredo, and S. Latré, "Multi-agent deep learning for multi-channel access in slotted wireless networks," *IEEE Access*, vol. 8, pp. 95 032–95 045, 2020. doi: 10.1109/ACCESS.2020.2995456

[42] R. Mennes, M. Claeys, F. A. P. De Figueiredo, I. Jabandžić, I. Moerman, and S. Latré, "Deep learning-based spectrum prediction collision avoidance for hybrid wireless environments," *IEEE Access*, vol. 7, pp. 45 818–45 830, 2019. doi: 10.1109/ACCESS.2019.2909398

[43] M. Ahmed Ouameur, L. D. T. Anh, D. Massicotte, G. Jeon, and F. A. P. de Figueiredo, "Adversarial bandit approach for ris-aided ofdm communication," *EURASIP Journal on Wireless Communications and Networking*, vol. 2022, no. 1, pp. 1–18, 2022. doi: 10.1186/s13638-022-02184-6

[44] M. V. C. Aragão, S. B. Mafra, and F. A. P. de Figueiredo, "Otimizando o treinamento e a topologia de um decodificador de canal baseado em redes neurais," *Polar*, vol. 2, p. 1. doi: 10.14209/sbrt.2022.1570823833

[45] F. Adib Yaghmaie and L. Ljung, "A crash course on reinforcement learning," *arXiv e-prints*, pp. arXiv–2103, 2021. doi: 10.48550/arXiv.2103.04910

[46] M. G. Kibria, K. Nguyen, G. P. Villardi, O. Zhao, K. Ishizu, and F. Kojima, "Big data analytics, machine learning, and artificial intelligence in next-generation wireless networks," *IEEE access*, vol. 6, pp. 32 328–32 338, 2018. doi: 10.1109/ACCESS.2018.2837692

[47] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah, "Machine learning for wireless networks with artificial intelligence: A tutorial on neural networks," *arXiv preprint arXiv:1710.02913*, vol. 9, 2017. doi: 10.48550/arXiv.1710.02913

[48] H. Yang, A. Alphones, Z. Xiong, D. Niyato, J. Zhao, and K. Wu, "Artificial-intelligence-enabled intelligent 6g networks," *IEEE Network*, vol. 34, no. 6, pp. 272–280, 2020. doi: 10.1109/MNET.011.2000195

[49] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. USA: Prentice Hall Press, 2009. ISBN 0136042597

[50] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, "Q-learning algorithms: A comprehensive classification and applications," *IEEE access*, vol. 7, pp. 133 653–133 667, 2019. doi: 10.1109/ACCESS.2019.2941229

[51] A. N. Burnetas and M. N. Katehakis, "Optimal adaptive policies for markov decision processes," *Mathematics of Operations Research*, vol. 22, no. 1, pp. 222–255, 1997. doi: 10.1287/moor.22.1.222

[52] M. Tokic and G. Palm, "Value-difference based exploration: adaptive control between epsilon-greedy and softmax," in *Annual conference on artificial intelligence*. Springer, 2011. doi: 10.1007/978-3-642-24455-1_33 pp. 335–346.

[53] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015. doi: 10.1038/nature14236

[54] A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman, "Pac model-free reinforcement learning," in *Proceedings of the 23rd international conference on Machine learning*, 2006. doi: 10.1145/1143844.1143955 pp. 881–888.

[55] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *arXiv preprint arXiv:1708.05866*, 2017. doi: 10.48550/arXiv.1708.05866

[56] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016. doi: 10.48550/arXiv.1509.06461

[57] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015. doi: 10.48550/arXiv.1511.05952

[58] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015. doi: 10.48550/arXiv.1509.02971
[59] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning*. PMLR, 2014, pp. 387–395.
[60] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," *Physical review*, vol. 36, no. 5, p. 823, 1930. doi: 10.1103/PhysRev.36.823
[61] A. R. Cassandra, "A survey of pomdp applications," in *Working notes of AAAI 1998 fall symposium on planning with partially observable Markov decision processes*, vol. 1724, 1998.
[62] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016. doi: 10.48550/arXiv.1606.01540

Sheila C. da S. J. Cruz received a bachelor's degree in computer engineering from the National Institute of Telecommunications (Inatel), Brazil, in 2016. She is currently working towards completing her master's degree at Inatel. Her research interests include digital communications, Wi-Fi, link adaptation, and machine learning. Orcid ID: 0000-0002-4905-518X

Felipe A. P. de Figueiredo received the B.Sc. and M.Sc. degrees in telecommunication engineering from the National Institute of Telecommunications (Inatel), Brazil, in 2004 and 2011, respectively. He received his first Ph.D. degree from the State University of Campinas (UNICAMP), Brazil, in 2019 and the second one from the University of Ghent (UGhent), Belgium, in 2021. He has been working on the research and development of telecommunication systems for more than 15 years. His research interests include digital signal processing, digital communications, mobile communications, MIMO, multicarrier modulations, FPGA development, and machine learning. Orcid ID: 0000-0002-2167-7286

Messaoud Ahmed Ouameur received a bachelor's degree in electrical engineering from the Institute national d'électronique et d'électricité (INELEC), Boumerdes, Algeria, in 1998, the M.B.A. degree from the Graduate School of International Studies, Ajou University, Suwon, South Korea, in 2000, and the master's and Ph.D. degrees (Hons.) in electrical engineering from the Université du Québec à Trois-Rivières (UQTR), QC, Canada, in 2002 and 2006, respectively. He has been a Regular Professor at UQTR since 2018. His research interests include embedded real-time systems, parallel and distributed processing with applications to distributed Massive MIMO, deep learning and machine learning for communication system design, and the Internet of Things with an emphasis on end-to-end systems prototyping and edge computing. Orcid ID: 0000-0003-1095-8012