Deep Reinforcement Learning for Dynamic Network Slicing in IEEE 802.11 Networks

Sibren De Bast*, Rodolfo Torrea-Duran*, Alessandro Chiumento†, Sofie Pollin*, and Haris Gacanin‡
*ESAT - TELEMIC, KU Leuven, Leuven, Belgium
†CONNECT, Trinity College, Dublin, Ireland
‡Nokia Bell Labs, Antwerp, Belgium

***Abstract*—Network slicing, a key enabler for future wireless networks, divides a physical network into multiple logical networks that can be dynamically created and configured. In current IEEE 802.11 (Wi-Fi) networks, the only form of network configuration is a rule-based optimization of few parameters. Future access points (APs) are expected to have self-organizational capabilities, able to deal with large configuration spaces in order to dynamically configure each slice. Deep Reinforcement Learning (DRL) can achieve promising results in highly dynamic and complex environments without the need for an operating model, by learning the optimal strategy after interacting with the environment. However, since the number of possible slice configurations is huge, achieving the optimal strategy requires an exhaustive learning period that might yield an outdated slice configuration. In this paper, we propose a fast-learning DRL model that can dynamically optimize the slice configuration of unplanned Wi-Fi networks without expert knowledge. Enhanced with an off-line learning step, the proposed approach is able to achieve the optimal slice configuration with a fast convergence, which is attractive for dynamic scenarios.**

***Index Terms*—network slicing, deep reinforcement learning, Wi-Fi networks**

I. Introduction

Due to the tremendous growth of indoor data traffic [1], IEEE 802.11 (i.e., Wi-Fi) constitutes a crucial technology for providing next-generation indoor services, such as massive Internet-of-Things, broadband wireless, and ultra-reliable communications, each of which has a diverse set of performance requirements [2]. Currently, multiple Wi-Fi access points (APs) are deployed without planning, with a large set of (often inter-dependent) parameters that can be tuned to optimize the network for different services. These parameters affect the network's spectral efficiency, and time, space, and frequency footprint by selecting the channel, bandwidth, modulation and coding scheme, or spatial multiplexing scheme. One promising solution to support the diverse set of requirements for multiple applications under a flexible infrastructure is network slicing. Unlike traditional network service delivery, network slicing divides physical network into multiple isolated end-to-end logical networks, each of which contains the necessary physical resources and configuration for network service delivery [3]–[5]. The creation, configuration, and management of the slices is done dynamically and on-the-fly. Traditional approaches for slice configuration in wireless networks use rule-based optimization techniques, which typically tackle only a single configuration parameter at a time and require static operating models that need to be updated constantly [6]. These models do not scale to networks with numerous diverse slices, a large number of configuration parameters, or very dynamic deployment scenarios. Furthermore, configuring the many inter-dependent parameters of APs at run-time cannot be done with traditional model-free approaches. For this purpose, future APs are foreseen to have self-organizational capabilities aided by machine learning (ML) that will observe the environment and act upon this by selecting the best configuration for all parameters dynamically. Reinforcement learning (RL) with Q-learning has been successfully used in wireless networks where a complete knowledge or model of the network environment is not available or it is extremely time-consuming to obtain [7]. For instance, due to the unplanned deployment of Wi-Fi APs and the increasing interference in ISM bands, obtaining a model of the network environment is not feasible in practice. However, the prohibitively high number of configurable parameters for network slicing and its dynamic behaviour results in a gigantic state-action space that cannot be learned completely within a steady period of the network. One way to deal with the gigantic state-action space is with a deep Neural Network (NN). The NN can be used to learn complex interactions in the environment and can predict the reward obtained by taking a certain action, hence avoiding the need to explore the whole state-action space. This is called deep reinforcement learning (DRL) [8] and it is shown to speed up convergence and increase robustness in wireless networks [9]–[11].

To this end, we present a fast-learning DRL model to dynamically optimize the network slice configuration without the need for network planning, even in scenarios with several network parameters to be configured and multiple users with varying requirements. We consider a scenario where multiple flows or slices need to be served over a single network, subject to unknown and dynamic interference. In order to speed up convergence, our approach includes an off-line training step that can even eliminate the on-line learning time, crucial for fast quality of service optimization in dynamic networks. To the best of our knowledge, this is the first time DRL is used to tackle dynamic network optimization for multiple applications, i.e. network slicing, in Wi-Fi networks. To summarize, the contributions of this work are:

- We propose a DRL model for fast optimization of Wi-Fi networks, without a-priori network information about the specific deployment scenario;
- We enhance the proposed DRL model with Double DQN [12] and fitted Q-iteration. While the first one reduces overestimations and improves performance, the second one reduces the on-line learning time;
- We show that our approach is able to find the optimal slice configuration at run-time, even in scenarios with multiple slices that have different throughput requirements.

II. Related Work

The application of DRL to wireless multichannel access networks in general has been prolific in the past. This is of importance as these networks share the basic medium access control mechanism with Wi-Fi networks. For instance, in [10], from selecting one (good/bad) channel at a time and learning from this experience, a user tries to obtain as much successful transmissions using a deep Q-network. In [11] users also follow a binary strategy (ACK/NACK) to update a deep Q-network in a central unit. This allows each user to adjust its transmission parameters while considering the multi-user learning in a fully distributed manner. In [9], DRL is used to solve the problem of time-sharing among multiple networks (including Wi-Fi) using different MAC protocols. By a series of observations (success/collision), actions (transmit/idle), and rewards, the proposed algorithm is able to learn the optimal MAC strategy even without knowledge of the MAC strategies of the other networks. In network slicing however, a simple binary model (good/bad, ACK/NACK, success/collision, transmit/idle) cannot capture all the variety of possibilities when dealing with such heterogeneous slices, that depend on different parameters or may require a more granular control of the user's quality of service.

In other papers, the application of DRL to Wi-Fi networks remains focused on the problem of the LTE and Wi-Fi coexistence. For instance, RL is used in [13] and [14] and DRL in [16] to learn the best allocation of resources between LTE and Wi-Fi networks. However, both the APs and base stations know the protocol used by the other network. Alternatively, our problem is not model-aware in the sense that the APs have no a-priori knowledge of the specific scenario nor of the networks that might be causing interference.

Other papers tackle specific releases of the 802.11 standard. For instance [15] enhances channel assessment capabilities in 802.11ac networks by altering the bandwidth and channel configuration using RL; while in [17], the MCS and frame size in a 802.11n network are jointly optimized with RL. Both papers rely on a period of interaction with the environment that is stable and sufficiently long to provide the optimal solution. Relying on such a long learning period will render in our case an outdated slice configuration.

In general, most papers that use RL or DRL for network optimization have a unidimensional view, which is not our case since our problem requires the optimization of different inter-dependent parameters subject to unpredictable sources of interference. More importantly, the problem of network slicing requires a high convergence speed due to the dynamic nature of the slices. This will require an off-line learning step to improve the on-line convergence speed.

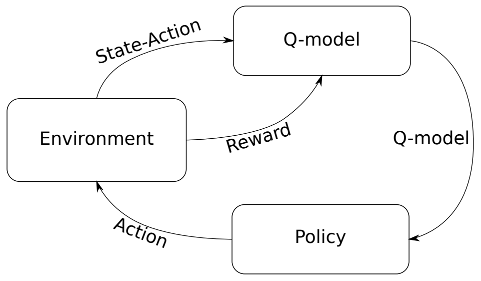

Fig. 1. An overview of the Q-learning algorithm. The environment reports the state-action and reward to the Q-model. Afterwards, this model is updated with the new information and passed on to the policy. Finally the policy uses this model to decide which action has to be taken. One full loop is one step for the system.

III. Deep Reinforcement Learning for Wi-Fi Network Slicing

A. Q-Learning

Q-Learning is a well-known unsupervised learning technique which aims to find the optimal strategy through interactions with the environment that results in the highest accumulated reward. Fig. 1 gives an overview of the Q-learning approach.

The strategy is learned through a state-action value function Q that represents the expected reward of taking an action a while being in state s. The action to take is chosen by the policy π.

$$Q^{\pi}(s,a) = \mathbb{E}\{R_t | s_t = s, a_t = a\}. \tag{1}$$

The optimal state-action value $Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$ follows the Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}\{r_{t+1} + \gamma \max_{a'} Q^*(s',a') | s_t = s, a_t = a\}, \tag{2}$$

where s' is the new state after taking action a, and r_{t+1} is the reward that the system receives for going from state s to s'. In Wi-Fi networks, the state of the environment contains all valuable parameters that can be logged by the APs (e.g.: MCS, channel selection).

For the environment to be able to allocate a reward to a certain state-action pair, a reward function has to be defined. The reward function defines the optimization objective of Q-learning. Therefore it will depend on parameters that are desirable to be optimized, e.g. throughput and latency are parameters that can used for defining a reward function in Wi-Fi networks. Note that without defining a proper reward function, Q-learning will not be able to optimize the environment.

In order to build the Q-function, we estimate each of the elements $q(s,a)$ by iteratively taking action a_t from state s_t at time t such that

$$q(s_t,a_t) = q(s_t,a_t) + \alpha(r_{t+1} + \gamma \max_{a'} q(s_{t+1},a') - q(s_t,a_t)), \quad (3)$$

where $\alpha \in (0,1)$ is the learning rate and $\gamma \in [0,1)$ is the discount factor. An ϵ-greedy policy can be adopted in which an action $a_t = \arg\max_a q(s_t,a)$ with probability $1-\epsilon$ and a random action with probability ϵ, is taken. This is done to avoid that the system is stuck in a local optimum. Typically, when the system is starting to learn, a large value of ϵ is adopted to allow for more exploration. After some time, the value of ϵ is reduced in order to allow for more exploitation.

B. Deep Reinforcement Learning (DRL)

The main drawback of Q-learning is the slow convergence rate due to its iterative nature and the fact that it does not rely on any prior information when it encounters a new state. For network slicing, the number of possible states and actions renders a simple Q-learning approach infeasible. With DRL, a deep neural network (NN) approximates the state-action value function $q(s,a,\theta) \approx Q^*(s,a)$ where the vector θ contains the weights of the NN. This NN is trained to minimize the prediction error given by its loss function, where the target output of the NN is defined by

$$y_t = r_{t+1} + \gamma \max_{a'} q(s_{t+1},a';\theta'). \quad (4)$$

This method, where the Q-function is modelled with the help of NNs, is called Deep Q-Networks (DQN) [8]. To further improve DQN for the specific context of Wi-Fi networks with diverse slices, we detail some techniques we have chosen to use in our approach below. These additional techniques are crucial to use DQN for Wi-Fi network slicing.

1) Experience Replay: A Wi-Fi network environment could operate in a stable context for a long time. If the Q-model would be trained on the data of this static environment for a long time, the NN weights would over-fit and the model would lose its effectiveness when the scenario changes. To combat this, Experience Replay is used. The main idea of experience replay is to train the DRL agent with knowledge acquired on-the-fly and stored in a buffer. The buffer is filled with previous experiences, which are composed of an action, a state, a reward and a new state that the agent has encountered. At every step the agent takes, a new entry is added to the buffer. Typically, a batch of randomly selected experiences is used at each step to train the agent. Experience replay is used during on-line learning for several reasons, among them to speed up the learning, break undesirable temporal correlations, and increase data usage and computation efficiency [18].

2) Double DQN: Wi-Fi users expect a certain Quality-of-service (QoS) in the network. Therefore the convergence speed and stability of the system are from utmost importance to measure the performance of the DRL agent. To improve the performance of DQN, Double DQN (DDQN) was introduced in [12]. In standard DQN, the same NN is used to select and to evaluate an action, resulting in overoptimistic reward estimations. To prevent this, DDQN proposes the use of two value functions resulting in two sets of weights, θ and θ' [12]. One is used to determine the action, while the other is used to evaluate its reward. This results in

$$y_t = r_{t+1} + \gamma q(s_{t+1}, \arg\max_a q(s_{t+1},a;\theta_t);\theta'_t) \quad (5)$$

DDQN improves the stability and overall performance of the learned model. This will improve the QoS for Wi-Fi users in comparison to a model that uses DQN.

3) Fitted Q-iteration: Building reliable knowledge using on-line learning might require numerous iterations before achieving a stable point, or even a point with reasonable performance. In our highly dynamic scenario, this will result in a slice configuration that is not up to date. Therefore instead of updating the Q-values on-line, Fitted Q-iteration [19], also called Fitted Q-learning, performs an off-line training step. It does this by taking a big number of learning samples that are generated in an off-line way, and trains the NN with this data. While generating the data for the Fitted Q-iteration, the actions can be chosen randomly. It has been shown that Fitted Q-iteration allows for a faster and more reliable convergence, which is key for network slicing.

IV. System Model

This section describes the implemented Wi-Fi scenario, the optimization parameters, and the details of the DDQN method.

A. Simulated Network Environment

We develop a simulation scenario with a Wi-Fi network having $N \in [2,5]$ users. We define for each user its own application requirements in a dynamic fashion. Thus, each user represents a slice with a specific performance requirement– in our scenario, throughput.

The users are located in a building with two floors, each floor consisting of a 3-by-3 grid of square rooms and each being 10-by-10 meters. The users and APs are randomly placed on the ground floor. On the first floor an interfering network is randomly placed, which uses the same frequency channel. Figure 2 shows one of the possible random scenarios. The network environment was implemented in the network simulator ns-3. Each simulation instance initialization generates a random slice configuration for a given environment (users location, channel conditions, interference power, required throughput, and number of users).

B. Parameter Selection

Each user is assigned a network slice and each slice requires a certain, randomly chosen throughput, which ranges from 1 to 30 Mbps. Note that a slice can never achieve a throughput higher than its required throughput, since this determines how many packets are offered at the AP for this slice. A slice can however achieve a lower throughput due to bad link quality or interference.

To decide upon the right action, the DDQN agent will need to know the state of the Wi-Fi network. The parameters and

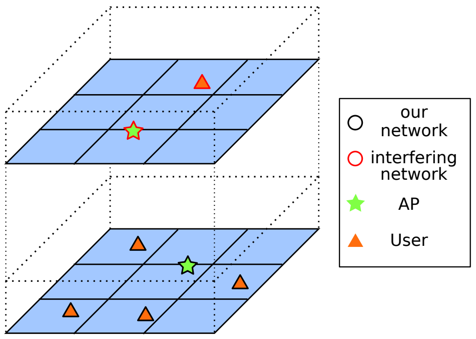

Fig. 2. One of the possible scenario configurations. The N users and AP are placed at a random position on the ground floor. The interfering AP and user are placed at a random location on the first floor. Each user is treated as a slice with throughput requirement T_s^R.

their ranges used as the environment state can be found in Table I. The range of parameters is used to normalize the data before we use it to train the NN, this improves the numerical stability and training speed of the NN. To acquire the complete state of the network, these eight parameters are logged for each slice. The total number of parameters used by the DDQN algorithm is 40, since our model supports up to five slices. When the Wi-Fi network contains less than five slices, zeros are added instead to the data. Adding support for more slices can easily be done by enlarging the NN to support more slices as input. Note that for each extra user the model has to be trained with data for these amount of slices.

TABLE I
THE PARAMETERS OF THE WIRELESS NETWORK USED AS THE STATE OF THE ENVIRONMENT.

State-Parameter	Range	unit
MCS index	$[0, 7]$	-
CCA threshold	$[-90, -70]$	dBm
TX Power	$[11, 20]$	dBm
Required Throughput	$[1, 30]$	Mbps
Achieved Throughput	$[1, 30]$	Mbps
RTT	$[0, 1000]$	ms
Frequency channel	$\{36, 38, 40\}$	-
RSSI	$[-100, 0]$	dBm

Wi-Fi networks have a lot of tunable parameters for each slice. Choosing which parameters to optimize for is an important task [20]. In the dense deployment of Wi-Fi systems, there are various inter-dependent parameters that are all affected by interference. For instance, interference can partly be alleviated by choosing the right channel and bandwidth. In 802.11n/ac/ax, this can be done dynamically while using bandwidths ranging from 20 MHz to 160 MHz. However changing the bandwidth in a dynamic way creates a trade-off in transmission range and probability of interference. [15] proposes as a potential solution a careful balance between the clear channel assessment (CCA) sensitivity level, the modulation and coding scheme (MCS) and the transmit power (TxP). Therefore our DDQN agent tries to achieve the required rate by configuring these parameters for each network slice together. The possible values for each parameter are: MCS $\in \{0, 1, 2, ..., 7\}$, CCA $\in \{-90, -85, -80, -75, -70\}$ dBm, TxP $\in \{11, 12, 13, ..., 20\}$ dBm.

C. Slice Optimization: Action and Reward

The goal of the system is to match the actual throughput with the required throughput by selecting the best slice configuration. This can be done by taking an action at each step. The possible actions that can be taken are:

- $MCS_{down}, MCS_{equal}, MCS_{up}$ for MCS selection;
- $CCA_{down}, CCA_{equal}, CCA_{up}$ for CCA selection; and
- $TxP_{down}, TxP_{equal}, TxP_{up}$ for selecting the transmit power.

If a *down* action is chosen, the chosen value moves one down in the list of possible values, it moves one up for an *up* action and stays the same for an *equal* action. In the case that an action would cause a parameter to move outside the predefined intervals, the action is disregarded. If there is a change in the environment, the system optimises the slice configuration towards that new context. Note that new slices can be created or deleted at run-time. In our scenario, this corresponds to adding or changing a user with a specific throughput requirement. This change in environment is shown in the state of the system. We note here that the agent sees a context change by perceiving a change in the network state.

In the slice optimization problem, the reward should account for the achieved throughput. Therefore, we define the reward function as

$$r_t = \sum_{i=1}^{n} \frac{T_{t,i}^A - T_{t-1,i}^A}{T_{t,i}^R}, \qquad (6)$$

where $T_{t,i}^A$ is the achieved throughput at time step t for slice i and $T_{t,i}^R$ is the required throughput at time step t for the same slice, where a time step equals 100 ms. The achieved throughput in $t-1$ is subtracted from the achieved throughput in time instant t, that way the reward function will only be positive for actions that improve the throughput of the network, which is only possible as long as the target throughput is not achieved. The reward is normalised by dividing by the required throughput. This is done for numerical stability and better performance of the NN.

D. Q-model: Neural Network architecture

To model the Q-function, a Neural Network is used. This Q-model estimates the rewards for all possible actions in a certain state. Therefore, the input size of the NN is equal to number of logged parameters in the state of the network. In this case there are five slices with eight logged parameters, so the input size of the NN is 40. The output of the network estimates the reward for each action. There are nine different action for each of the five slices, therefore, the output size is 45. To Model the Q-function, a simple Feed-Forward network was selected with 5 hidden layers. The number of neurons in these layers are 512, 256, 196, 128, and 96 respectively. As optimiser of the NN "Adam" was selected, with an initial learning rate of 0.001. DDQN need a well-chosen discount factor in order to operate, for this application $\gamma = 0.5$ showed to be the most effective.

V. Performance Evaluation

In the following sections we explore different cases, and the learning behaviour of the proposed system. In Section V-A the effect of increasing the number of training samples for off-line training is explored. Section V-B evaluates the performance of the proposed approach when taking more than one action simultaneously. In these two sections, the results are averaged over 40 random instances of the scenario described in Section IV. We also searched the optimal solution for these 40 scenarios with an exhaustive search, since there are scenarios where no solution can meet the required throughput. Finally, in Section V-C we analyse the performance of a dynamic scenario where the environment changes constantly.

A. Off-line training (Fitted Q-iteration)

In this section, we analyse the effect of performing off-line training. We evaluate the performance of the proposed approach trained with a dataset of a specified size. A training sample consists of one step in one specific scenario, giving the state parameters from Table I, for an initial state s and the next state s', following action a. The results of this experiment can be seen in Fig. 3. The case of $\text{Size}_{training} = 0$ *does not* use Fitted Q-learning. As expected, a larger training set leads to a better performance. This shows that knowledge, characterizing the typical behaviour of Wi-Fi networks, is useful to speed up the learning and can be learned off-line.

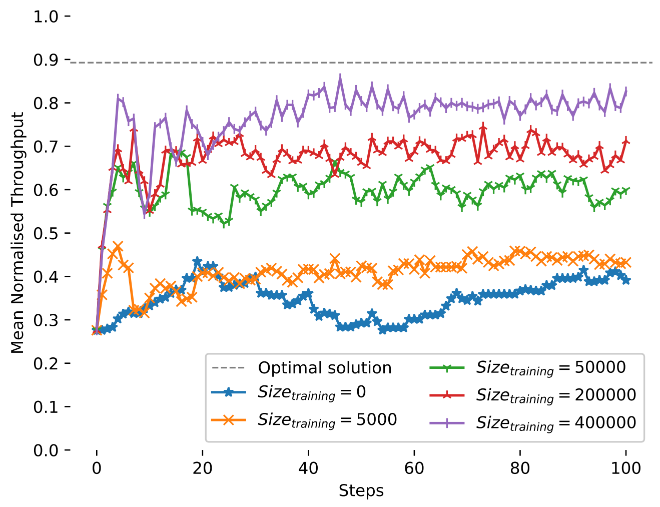

Fig. 3. The effect of the number of samples used during Fitted Q-learning, on the convergence speed. A bigger training size leads to better performance of the system.

B. Multiple simultaneous actions

This section evaluates the performance of three possible strategies for taking actions. In the first strategy, one action is taken for the whole network, this changes one parameter for the whole network at each step. In the second one, an action is taken *for each* slice at each step. And in the last strategy one action is taken per parameter per slice, possibly changing all parameters for each slice at each step.

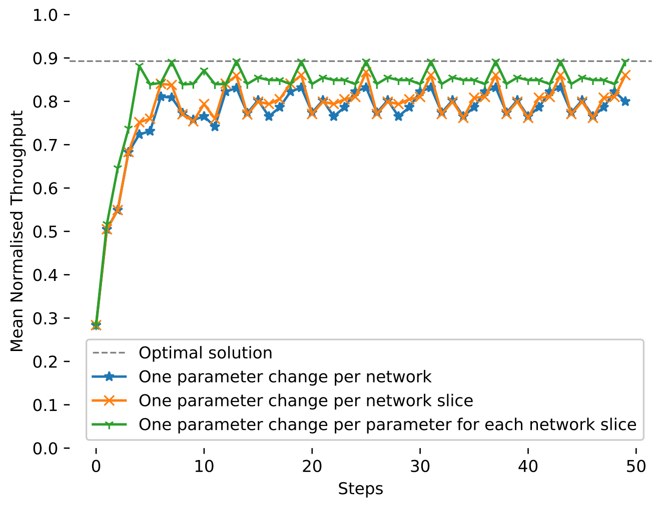

Fig. 4. Mean normalized throughput for taking a single action, taking one action per slice and taking an action for each tunable parameter per slice. Taking multiple actions at the same time makes for a higher convergence speed, provides a higher throughput and reduces the fluctuations in the system.

As can be seen in Fig. 4, taking one action per parameter for each slice increases the speed of convergence and the stability of the system. It also shows that this method is able to reach the optimal solution. However, the system cannot distinguish the contribution to the reward from multiple individual actions, therefore in this case the proposed approach is not able to learn from this experience. Due to this fact, taking multiple actions is only advised once the system is already trained sufficiently and the need to explore the state-space is small enough.

C. Dynamic environments

In this section we analyse the performance of the system in a dynamic environment. This means that we randomly change the environment every 20 steps, changing both the interference context, throughput requirements and location per user, and measure the actual throughput and required throughput in Fig. 5. The network is: a) pre-trained, using Fitted Q-iteration, with 2×10^5 samples and changes multiple parameters at each step, b) pre-trained with 2×10^5 samples and changing one parameter at each step for the whole network, and c) untrained and changing a single parameter in each step.

We can observe that the pre-trained system that takes multiple actions is able to achieve a throughput equal or close to the required throughput in just a few steps and it remains stable until a new environment is detected. However in other cases the pre-trained system taking only one action has less consistent results and a chance of oscillation as can be seen between step 40 and 60. The untrained agent is only able to optimize after encountering a couple of different scenarios. This clearly shows the benefits of pre-training the model using fitted Q-iteration, as well as taking multiple actions instead of a single one.

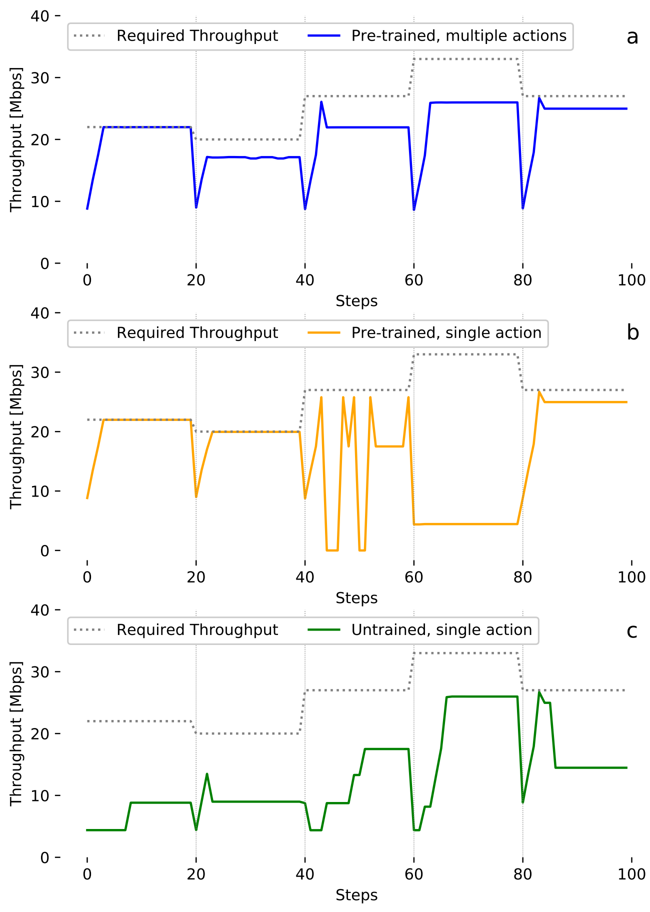

Fig. 5. Every 20 steps the environment changes to create a dynamic environment. The figure shows how the proposed approach can deal with this dynamism and tries to meet the new set of requirements.

VI. Conclusions

We present a fast-learning DRL model that can dynamically optimize the network slice configuration in Wi-Fi networks. Our network slices require each a different network configuration. Therefore the search space consists of all possible Wi-Fi parameter configurations multiplied by the number of slices in the network, giving a large state-action space.

Our approach starts from a DQN agent. We further enhance it with Double Deep Q-Networks [12] and experience replay, which improves convergence speed and stability. Moreover, by applying Fitted Q-iteration, the system learns a generic model that has been learned from other Wi-Fi scenarios, and covers the typical performance model of a randomly deployed Wi-Fi network subject to interference in the ISM band. Using this model, DDQN can optimize fast at run-time, without the need of specific AP deployment information or knowledge about the neighbouring interfering networks.

The proposed model-free approach is able to dynamically optimize various Wi-Fi parameters per slice. Our ns-3 based simulation results show that the proposed DDQN approach results in a high performance, even in scenarios in which there are many network parameters to be configured from multiple users with varying requirements.

References

[1] "Cisco Visual Networking Index: Forecast and Methodology, 20162021", *Cisco white paper*, Sep. 2017.

[2] H. Gacanin and M. Wagner, Artificial Intelligence Paradigm for Customer Experience Management in Next-Generation Networks: Challenges and Perspectives, IEEE Network Magazine

[3] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network Slicing in 5G: Survey and Challenges", *IEEE Communications Magazine*, vol. , no. , May 2017.

[4] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira, "Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges", *IEEE Communications Magazine*, vol. , no. , May 2017.

[5] X. Li, M. Samaka, H. A. Chan, D. Bhamare, L. Gupta, C. Guo, R. Jain, "Network Slicing for 5G: Challenges and Opportunities", *IEEE Internet Computing*, vol. 21, no. 5, 2017.

[6] S. Vassilaras, L. Gkatzikis, N. Liakopoulos, I. N. Stiakogiannakis, M. Qi, L. Shi, L. Liu, M. Debbah, and G. S. Paschos, "The Algorithmic Aspects of Network Slicing", *IEEE Communications Magazine*, vol. , no. , Aug. 2017.

[7] A. Chiumento, C. Desset, S. Pollin, L. Van der Perre and R. Lauwereins, "Impact of CSI Feedback Strategies on LTE Downlink and Reinforcement Learning Solutions for Optimal Allocation," IEEE Transactions on Vehicular Technology, 2017, volume 66, pages 550-562.

[8] V. Mnih et al, "Human-level control through deep reinforcement learning", *Nature*, no. 518, Feb. 2015, pp. 529 – 533

[9] Y. Yu, T. Wang, S. C. Liew, "Deep-Reinforcement Learning Multiple Access for Heterogeneous Wireless Networks", Dec. 2017.

[10] S. Wang, H. Liu, P. H. Gomes, and B. Krishnamachari, "Deep Reinforcement Learning for Dynamic Multichannel Access in Wireless Networks", *IEEE Transactions on Cognitive Communications and Networking*, Feb. 2018.

[11] O. Naparstek and K. Cohen, "Deep Multi-User Reinforcement Learning for Distributed Dynamic Spectrum Access", Nov. 2017.

[12] H. van Hasselt , A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning", *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*, 2016.

[13] Y.-Y. Liu and S.-J. Yoo, "Dynamic Resource Allocation Using Reinforcement Learning for LTE-U and WiFi in the Unlicensed Spectrum", *Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, Jul. 2017.

[14] N. Rupasinghe and I. Guvenc, "Reinforcement Learning for Licensed-Assisted Access of LTE in the Unlicensed Spectrum", *IEEE Wireless Communications and Networking Conference (WCNC)*, 2015.

[15] S. Jang, K. G. Shin, and S. Bahk, "Post-CCA and Reinforcement Learning Based Bandwidth Adaptation in 802.11ac Networks", *IEEE Transactions on Mobile Computing*, vol. 17, no. 2, Feb. 2018.

[16] U. Challita, L. Dong, and W. Saad, "Proactive Resource Management in LTE-U Systems: A Deep Learning Perspective", Feb. 2017.

[17] K. Zhou, "Robust Cross-layer Design with Reinforcement Learning for IEEE 802.11n Link Adaptation", *IEEE International Conference on Communications (ICC)*, Jul. 2011.

[18] R. Liu and J. Zou, "The Effects of Memory Replay in Reinforcement Learning", *ICML 2017 Workshop on Principled Approaches to Deep Learning*, 2017.

[19] M. Riedmiller, "Neural Fitted Q Iteration First Experiences with a Data Efficient Neural Reinforcement Learning Method", *European Conference on Machine Learning*, pp. 317–328 2005.

[20] A. Ligata, E. Perenda and H. Gacanin, "Quality-of-Experience Inference for Video Services In Home Wi-Fi Networks," IEEE Communication Magazine, March 2018.