AI Enabled Slice Resource Management for Improved QoS in WiFi-based IoT Networks

REIJER VAN HARTEN, University of Twente, The Netherlands

As new IoT services are being developed, the number of connected devices with different QoS requirements in many WiFi networks is ever-increasing. The current standard of using different access categories is often not well suited to provide the wide variety of QoS requirements of the different connected devices. Network slicing, where network airtime is divided into segments called slices, is a potent technology to meet diverse QoS requirements in IoT networks. Traffic flows with different QoS requirements are assigned to slices and their resources are managed to prioritize flows over one another and subsequently meet their requirements. Proposed solutions have considered a limited number of slices, however, and QoS diversity in IoT requires eight or more slices to efficiently meet QoS requirements. Managing the resources of a larger number of slices is a complex task, but AI can be used to handle this complexity. Therefore, we have implemented a DRL agent using DDPG that can divide the airtime between the slices in a simulated IoT network with eight slices and manages to nearly provide the necessary throughput requirements in a network where resources are limited.

Additional Key Words and Phrases: Wireless network slicing, Deep reinforced learning, Software-defined networking, IoT, QoS, WiFi

1 INTRODUCTION

Over the last decade, there has been a vast increase in the number of devices connected to the internet. This trend also applies to the number of connected IoT devices, which hit 10.0 billion in 2019 and is expected to increase to 30.9 billion by 2025 [7]. Similarly, the number of different applications of IoT has increased as well. Many of these applications have varying quality of service (QoS) requirements to function optimally, including requirements for throughput, latency and reliability. Meeting these requirements can often be crucial for the performance of the IoT device [12].

The current standard of providing the QoS requirements in a wireless network is by using the EDCA access categories defined in IEEE 208.11e [1]. This approach is not suitable to ensure all the QoS requirements are met. It can only distinguish between four fixed categories of traffic and provides a fixed configuration for each of them. This is not enough to support the variety of QoS requirements and the range of network dynamics that may occur in modern IoT Networks [9].

One relatively new method of managing the different requirements for the devices connected to an access point is to use *network slicing*. This solution aims to categorize the connected devices with similar QoS requirements into groups called *slices*. The available airtime in the network is then divided proportionally between the slices by assigning each slice a *quantum* value which determines how much airtime the slice gets. In this time period, the slice gets complete access to the network. This allows us to configure the network to support more than the four fixed QoS categories in IEEE 208.11 [10].

Multiple existing works focus on the optimization of different parameters of the slicing algorithm (see section 2), but some parameters are hard to optimize mathematically. Especially when there are many different QoS requirements and a large number of slices is needed [4]. This optimization gets even harder when unstable network conditions cause the available throughput to be unpredictable and the parameters constantly need to be reconfigured to keep supporting the QoS requirements as best as possible. When a network becomes overloaded with traffic, an access point should use the best possible parameters to control the slices and to ensure maximum bandwidth while maintaining the QoS requirements. Because WiFi connections often exist in a non-static environment where mobile devices can move around and connection quality can vary over time, adjustments to these parameters can preferably be performed in real-time without the need for human intervention.

In this work, we address this challenge by proposing a DRL-based slice resource management solution which will learn about the current network's dynamics and will be able to predict the best division of airtime between the slices to ensure their QoS requirements. Our proposed solution will be able to support eight different slices because this allows for a relatively wide range of QoS requirements to be supported. For instance, with eight slices, we can support all combinations of the requirements for throughput, latency and reliability. Each slice would then correspond to a subset of 0, 1, 2 or all of the requirements, meaning $2^3 = 8$ slices would be needed. Our contributions can be summarized as follows:

- We have developed a DRL-based slice resource management algorithm to automate the process of dividing network resources (airtime) for network slicing in an SDN-controlled IoT network to meet the QoS requirements.
- We tested our solution on a simulated network with eight slices to demonstrate how machine learning can help achieve better QoS in a dynamic and adaptive manner compared to EDCA and other proposed solutions in the existing literature.

2 RELATED WORK

Relevant work includes the research done by Richart et al., who have taken big steps to analyze the performance of sliced WiFi networks. In their 2019 work [10], they describe the *Adaptive Time-Excess Round Robin* (ATERR) scheduling mechanism which aims to allocate different airtime requests of the slices and to fairly allocate airtime to each user within a slice. Given an existing set-up with slices, they propose calculations which allow for the slices to gain a requested portion of the transmission time available in the network. However, the proposed solution of providing a fixed portion of airtime to a slice does not give any guarantees about the available throughput under varying and dynamic channel conditions.

The team has since then extended their work to propose a mathematical solution aiming to guarantee all the QoS requirements [9]. This solution cannot guarantee throughput rates in practice either,

TScIT 39, July 7, 2023, Enschede, The Netherlands
2023.

because the total throughput of the wireless channel can be unpredictable. Wireless channel conditions can fluctuate wildly and the Modulation and Coding Scheme (MCS) has large effects on the available bandwidth as well.

In the 2019 study done by Isolani et al. [3], a solution for slice resource management is proposed which uses two slices: A QoS slice which focuses on providing a set of requirements, and a *best effort* slice which contains all the other network flows. In 2020, the team proposed a mathematical solution for an arbitrary number of slices [4]. But this solution takes too long to compute for more than seven slices, making it impractical for controlling IoT networks in real-time.

Although research exists on the application and benefits of WiFi network slicing, these solutions do not perform well under unpredictable network conditions. A solution involving machine learning can consider more variables, such as the MCS and the current network status and can be used to dynamically adjust the slices' configuration to meet the QoS requirements.

Using Deep Reinforcement Learning (DRL) to manage slices has already been proven to provide higher throughput rates compared to solutions from other studies implementing WiFi network slicing. Work in [12] has employed DRL to allocate resources to the network slices in an IoT network to enable dynamic support for the QoS requirements. The proposed machine learning algorithm considers the current throughput of the slices and the required throughputs of the slices. It then modifies the quantum values assigned to the slices to change the amount of airtime each slice is allowed to use and maximize the satisfaction of the individual slices. The work provides a useful starting point for finding an efficient solution. However, it only considers using three slices which is not sufficient to address the QoS diversity in an IoT network.

3 NETWORK SETUP

This work aims to find a solution for resource management in a network with one access point and eight different IoT devices with varying QoS requirements, as seen in figure 1. This network is controlled by an SDN controller and implements WiFi network slicing for eight different slices. We send one stream of data to each of the devices in the downlink, so each stream serves a different IoT application with unique QoS requirements. Each stream gets assigned a unique DSCP value, which ensures that each stream gets assigned to a unique slice.

A DRL agent is running on the controller, which communicates with the access point implementing WiFi slicing. The agent receives statistics for the throughput of each of the slices and can change the division of airtime for the slices.

The slices are scheduled using *Airtime Deficit Weighted Round Robin* scheduling. With this type of scheduling, each of the slices receives a set amount of time they are allowed to send for during each round, which is called the *quantum* value for the slice. One by one, the slices are selected to transmit data packets. The selected slice remains active as long as the expected time to send the next packet is smaller than the remaining amount of time assigned to the slice. If the total amount of time given to the slice gets exceeded, this gets subtracted from their allotted airtime in the next round [2].

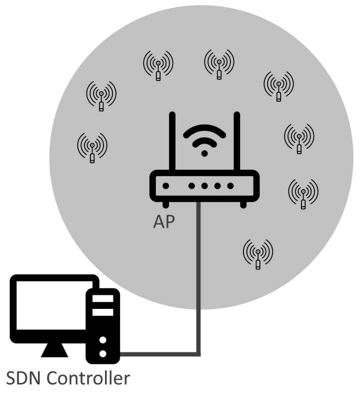

Fig. 1. Network Overview

4 PROBLEM STATEMENT

The slice resource management problem can be modelled as a Markov Decision Process, where the goal of the agent is to find policy π which aims to find the optimal action a for each state s to maximize the expected reward value q_π, where q_π is defined as follows:

$$q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t r_t \middle| S_0 = s, A_0 = a\right] \tag{1}$$

where $\mathbb{E}_\pi$ denotes the expected value of the random variable when the agent follows policy π. Furthermore, γ is the discount factor (0.8), r_t is the reward at time t and S_0 and A_0 are the initial state and action respectively.

The action, state and reward are defined as follows:

- *Action:* Actions are defined as a vector of 8 values, where each value determines the new quantum value for one of the slices. It can be represented by

$$a_t = (q_1, q_2, \ldots, q_8) \tag{2}$$

  where q_i determines the quantum value for slice i at time $t+1$. Because the agent does not learn to provide QoS guarantees for scheduling delays by itself, we define the maximum airtime to be assigned to each of the slices as 10000 μs. This ensures that the scheduling delay remains small. Furthermore, we limit the minimum airtime to 250 μs because the system becomes more unpredictable when too small quantum values are used, making the training process more difficult.

$$\forall q_i \in a_t, q_i \in [250, 10000] \tag{3}$$

- *State:* States are given by a vector of 8 values, where each value represents the average throughput of one of the slices, as measured over the last 4 seconds.

$$s_t = (tp_1, tp_2, \ldots, tp_8) \tag{4}$$

- *Reward:* We defined and tested two different reward functions. The first reward function aimed to maximize the smallest ratio of the achieved throughput to the required throughputs for any of the slices. This reward function is shown below:

$$reward = 100 * \min_i \frac{throughput_i}{requirement_i} \tag{5}$$

with $i \in [1, 8]$. Because the reward is based on the slice which gets the least of its required throughput, this ensures that the agent will learn to optimize for the requirements of all of the slices.

The second reward considered the sum of the squared differences between the throughput requirements and the measured throughputs of the slices:

$$reward = 100 - \sum_{i=0}^{8} \max(0, (requirement_i - throughput_i)^2) \tag{6}$$

Using this function, the predictions by the agent were much less accurate, which is why we opted to use the first reward function instead.

5 PROPOSED SOLUTION

Our proposed solution for controlling the slices in a WiFi-based IoT network is to use a DRL agent to manage the slices. In the work of Zia et al. [12], the authors used a deep Q-learning agent to predict what actions to take. In each time step, the agent takes an action which determines for each slice if the quantum value should be reduced, increased or kept the same. Because a deep Q-learning agent can only predict one action at a time, the number of actions which could be taken by the agent was equal to 3 to the power of the number of slices. So this approach does not scale to a solution which can solve the problem for eight slices.

Instead, we propose to use a *deep deterministic policy gradient* (DDPG) agent. Such an agent can predict values from a continuous action domain [6]. DDPG has already been proven to be a successful tool in similar tasks, such as managing transmission power in a 5G network [8]. In theory, such a DDPG agent would be able to directly predict the quantum values for each individual slice, vastly reducing the number of required output nodes and allowing for more accurate control over the quantum values.

5.1 Deep Deterministic Policy Gradient

DDPG is an actor-critic, model-free machine learning algorithm [6]. The term actor-critic refers to the fact that the algorithm uses two different neural networks to predict different values. Firstly, the actor network takes the current state as input and predicts which action should be taken to maximize rewards. Specifically, it tries to find action a for which

$$a = \arg\max_a Q(s, a) \tag{7}$$

where $Q(s, a)$ is the expected reward for taking action a in state s. Secondly, the critic model takes the state and the predicted action as input and evaluates the action by predicting the expected reward value for performing the given action in the given state. This expected reward has been defined in equation 1.

To stabilize the learning process, we introduce two more neural networks: a target actor and a target critic network. These networks are more stable because their parameters are updated using a soft update function, which can be seen in this equation:

$$\phi \leftarrow (1 - \tau)\,\phi + \tau\theta \tag{8}$$

where ϕ is the target network's parameters, θ is the original network's parameters and τ is a small number between 0 and 1.

At each timestep, the following steps are performed:

1. The agent takes an action based on the current state. This action is a vector of the quantum values: One value for each of the slices. We add Gaussian noise to the predicted action to allow for exploration of the action space. We then restrict the quantum values to be between 250 and 10000.
2. The quantum values of the network are updated and we receive the reward and the next state from the environment.
3. The agent then stores the current state, action, reward and next state in a replay buffer. Afterwards, we take a batch of 64 random samples from the replay buffer for learning.
4. The critic model's parameters are updated to minimize the loss across the samples in the batch.
5. The actor model's parameters are updated to maximize the expected reward. We define the loss of the actor as the negation of the expected reward, which will be predicted by the critic.
6. The target networks are updated using the soft update function.

The pseudocode given in algorithm 1 gives a complete overview of the steps described above.

Algorithm 1 DDPG

```
Randomly initialize critic network $Q$ and actor network $\mu$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target networks $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$.
Initialize replay buffer $R$.
Observe the initial state $s_1$.
for $t = 1,2,\ldots,t_{end}$ do
    Calculate standard deviation $\sigma \leftarrow 2500 - \frac{t}{300000*(2500-100)}$.
    Sample noise $N_t \leftarrow X_1, X_2, \ldots, X_n$ with $X_i \sim N(0, \sigma^2)$.
    Select action $a_t \leftarrow \mu(s_t|\theta^\mu) + N_t$.
    Ensure $a_t$ bounds: $a_t \leftarrow \max(250, \min(10000, a_t))$.
    Execute action $a_t$ and observe reward $r_t$ and new state $s_{t+1}$.
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$.
    Sample a random batch of 64 transitions $(s_i, a_i, r_i, s'_i)$ from $R$.
    Calculate target value $y_i \leftarrow r_i + \gamma Q'(s'_i, \mu'(s'_i|\theta^{\mu'})|\theta^{Q'})$.
    Update critic parameters using loss:
        $L = \frac{1}{64}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update actor parameters using loss:
        $L = -\frac{1}{64}\sum_i (Q(s_i, \mu(s_i|\theta^{mu})|\theta^Q))$
    Update the target networks:
        $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
        $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
end for
```

Parameter	Value
(Target) actor network layer sizes	(8, 256, 128, 8)
(Target) critic network layers sizes	(16, 256, 128, 1)
Hidden layer activation function	ReLU
Output activation function (actor)	Softmax
Output activation function (critic)	linear
Learning rate (actor)	0.001
Learning rate (critic)	0.002
Optimizer	Adam
Discount factor γ	0.8
Soft update parameter τ	0.001
Replay buffer size	50000
Replay buffer sample size	64
Noise standard deviation σ (interval)	[100, 250]

Table 1. Training parameters

5.2 Deep Neural Network Architecture

We have discussed the four required deep neural networks in the DDPG algorithm. We will now discuss the architecture of these networks and the values of the hyperparameters. An overview of these parameters is shown in table 1.

First, the actor and the target actor networks have an input layer of 8 nodes, two hidden layers with 256 and 128 nodes, and one output layer with 8 nodes. The hidden layers use the ReLU activation function and the output layer uses the Softmax activation function. The outputs are then scaled to represent quantum values between 250 and 10000. The actor's learning rate is 0.001.

Second, the critic and the target critic networks have a similar structure to the actor network: They have two hidden layers with 256 and 128 nodes which use the ReLU activation function. The input layer consists of 16 nodes and the output layer has one node which uses the linear activation function. The critic's learning rate is 0.002.

The networks are implemented using the Python Keras library and use the Adam optimizer [5]. We set the discount factor γ to 0.8 and the soft update parameter τ to 0.001. Our replay buffer keeps track of the previous 50000 samples.

The standard deviation of the Gaussian noise added to the predicted actions decreases during training. It starts at 250 and decreases linearly to 100 during the first 300,000 timesteps.

6 PERFORMANCE ANALYSIS

To analyze the performance of the slice resource management, we set up the DRL agent in a simulated environment. We ran the DDPG algorithm as described in section 5 and used the simulation to estimate the resulting state and reward after every action taken by the agent.

The simulated network had eight simulated connections with different DSCP values and each of the connections was given different throughput requirements, as can be seen in table 2. Because each connection had a different DSCP value, only one connection was served by each of the eight slices.

DSCP value	Required Throughput (Mb/s)
0	2.0
8	0.1
18	0.3
20	1.5
30	6.0
44	4.0
46	3.0
48	2.5

Table 2. Throughput requirements for each of the slices

To ensure that the actor would learn to take optimal actions, even when the throughput of a network is constantly changing, we simulated fluctuations in the throughput of the network. Therefore, on each timestep, we added a random value between -0.5 and 0.5 to the throughput of the previous timestep. The total throughput was always bounded between 18 and 22 Mb/s.

$$tp_t = \max(18, \min(22, tp_{t-1} + R_t)) \tag{9}$$

where R_t is a random value sampled from a uniform distribution.

$$R_t \sim U(-0.5, 0.5) \tag{10}$$

The action taken by the actor defines the quantum values to be used for each of the slices. We can estimate the resulting throughputs for each of the slices by dividing the throughput over the slices proportionally to the quantum values.

$$tp_{s,t} = \frac{quantum_{s,t}}{\sum_i quantum_{i,t}} * tp_t \tag{11}$$

where $tp_{s,t}$ is the throughput of slice s at time t, $quantum_{s,t}$ is the quantum value assigned to slice s by the actor at time t and tp_t is the total throughput of the simulated network at time t.

Finally, we calculated the reward value as shown in equation 5. The calculated new throughputs and the reward value were then stored in the replay buffer to be used for learning.

The code for the setup can be found on GitHub [11].

6.1 Results

We have run our DRL solution on a simulated environment for 1,000,000 timesteps. After every 500 timesteps, we took the average throughput for each of the slices over this period. These averages are plotted in figure 2.

The figure shows that the agent initially did not have any understanding of the environment and was predicting a wide range of quantum values, causing unstable throughput rates for the slices. The first two slices that were being optimized were slices 8 and 18, which both got assigned the minimum possible quantum values after timestep 300,000, probably because the throughputs of these slices were easily met and lowering their quantum values did not influence the reward value. Finally, after 700,000 timesteps the algorithm converged and was able to maintain fairly constant throughput rates for each of the slices. Four of the slices got their required throughput rates. The average throughput rates of the other four slices, taken over the final 5000 timesteps, were within 0.15 Mb/s of the throughput requirements of the slices, or 7.3%, as

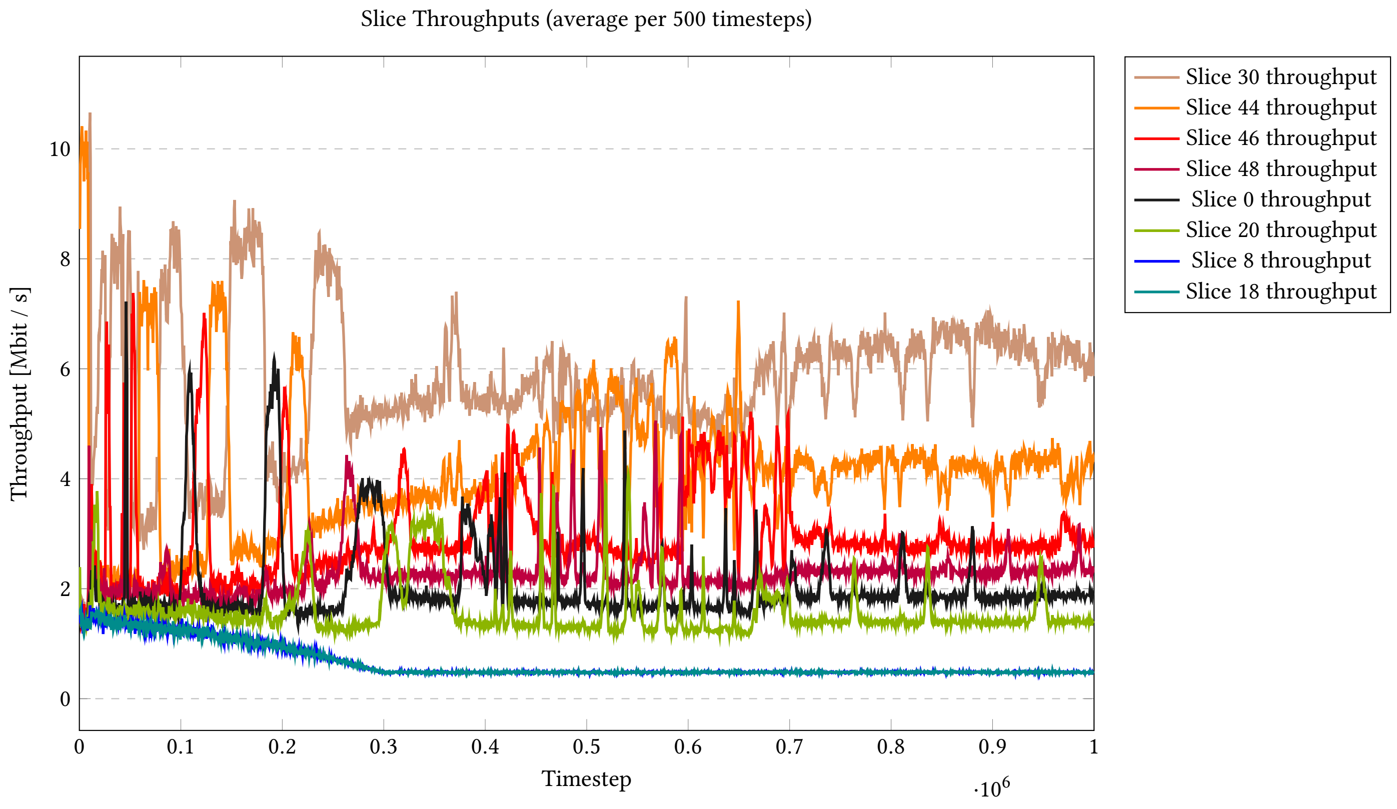

Fig. 2. Slice throughputs in the simulated environment

DSCP value	Required Throughput (Mb/s)	Achieved Throughput (Mb/s)	Difference (Mb/s)	Percentual Difference
0	2.00	1.89	-0.11	-5.4%
8	0.10	0.48	+0.38	+377%
18	0.30	0.48	+0.18	+59%
20	1.50	1.39	-0.11	-7.3%
30	6.00	6.12	+0.12	+1.9%
44	4.00	4.35	+0.35	+8.8%
46	3.00	2.85	-0.15	-5.0%
48	2.50	2.35	-0.15	-6.1%

Table 3. Achieved throughputs in the simulated environment (mean over the last 5000 timesteps)

shown in table 3. In these final 5000 timesteps, the average total throughput of the network was 19.9 Mb/s, which was only 0.5 Mb/s bigger than the sum of the required throughputs of the slices.

Figure 3 shows the reward values given to the agent. Again, we show the average of every 500 timesteps. As we saw before in figure 2, the agent converged after 700,000 timesteps. The figure also shows that the agent never managed to receive the maximum reward value of 100. In the final 5000 timesteps, it got an average reward of 85.4. So, in accordance with our reward calculation, there was still room for improvement.

6.2 Discussion

Even though the DRL agent was unable to find the right quantum values to provide 100% of the throughput requirements of the slices, we still believe our implementation can be considered successful. Because the total bandwidth of the network was only 3% larger than the sum of the throughput requirements of the slices, we consider the small differences between the required and the achieved throughputs to be acceptable. However, future studies can try to improve the accuracy of our implementation, for example by trying different parameters for the neural network.

One aspect worth mentioning is that the agent takes a long time to learn about the network dynamics. In a real network, conditions

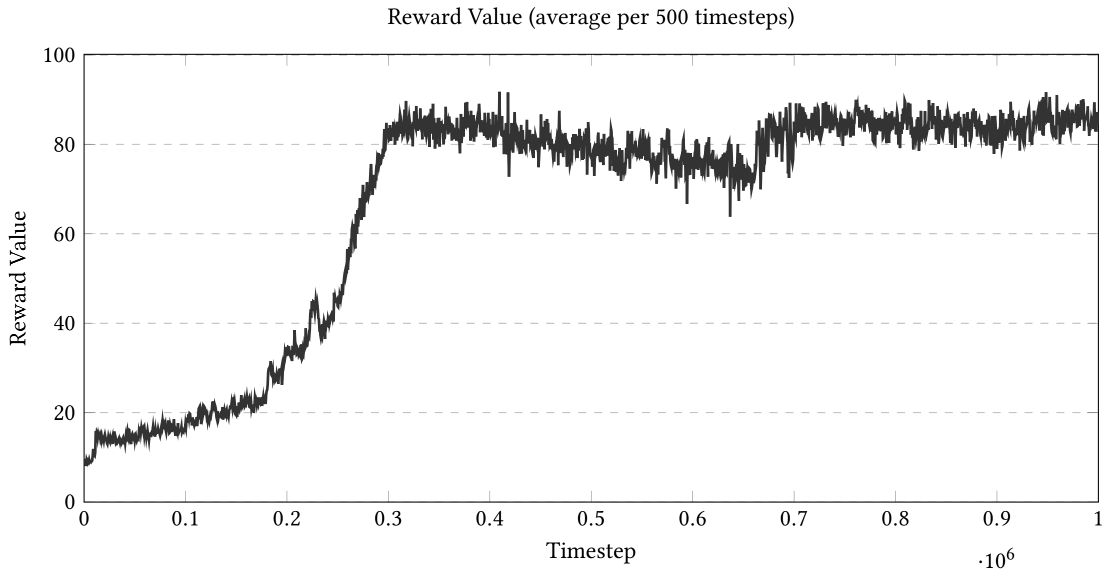

Fig. 3. Reward value in the simulated environment

might be even more dynamic which might make it take even longer for the agent to learn. And even if the agent converges in the same number of timesteps (700,000), this would still take very long in a non-simulated environment. In a real network, we need multiple seconds to measure the new throughputs of the slices before being able to calculate the reward value. Therefore, it would probably take multiple days before the DRL learning algorithm converges.

Because of this long learning time, it is important to find a solution which can quickly adapt to changing network environments and avoid having to restart the training process when faced with different requirements. Potentially, our solution might be able to learn how to adapt if it gets trained on a wide variety of network setups with varying requirements, but this requires further research to determine. This research could also investigate if there are benefits to providing the agent with input values other than the current throughputs of the slices. Maybe the agent can adapt to a new network environment more quickly if we give it other information, such as the current MCS values used for the different connections.

One final point of discussion is that the agent does not consider different priorities between the QoS requirements. If the throughput of the network gets too low to provide the throughput requirements for all of the slices, the agent treats them equally and takes the same percentage of bandwidth from all of them. This problem can be addressed by changing the reward function to a function which puts more importance on the requirements of slices with higher priority.

7 CONCLUSION

Because of the rising number of IoT applications with different QoS requirements, it is increasingly more complex to manage all QoS requirements in WiFi networks. Network slicing is a technology which aims to provide a wide variety of QoS requirements by dividing the airtime of the network over slices with varying QoS requirements. To manage the division of airtime between the slices, we proposed a DRL-based solution for slice resource management in an SDN-controlled wireless IoT network with 8 slices. This is an improvement over earlier studies because it allows supporting more slices than before, which ensures that a wider variety of QoS requirements can be provided. We proposed the implementation of A DRL agent using DDPG and tested our approach by running the agent in a simulated network. The agent was able to evenly divide the network resources between the different slices according to their throughput requirements but was not able to guarantee that all of the requirements were met in a network with limited bandwidth.

Future research should focus on determining the agent's ability to adapt to changing network dynamics and QoS requirements. Furthermore, research can be done on the effects of using different input metrics or changing the hyperparameters used for the DRL agent and the neural networks. This might help it adapt to different network setups more quickly, as well as further increase its accuracy. Finally, an improved reward value might be considered to allow the agent to prioritize between the different QoS requirements of the connected devices.

REFERENCES

[1] 2005. *IEEE Std 802.11e-2005.* IEEE. OCLC: 956670291.

[2] Estefania Coronado, Roberto Riggio, Jose Villalon, and Antonio Garrido. 2018. Lasagna: Programming Abstractions for End-to-End Slicing in Software-Defined WLANs. In *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM).* IEEE, Chania, Greece, 14–15. https://doi.org/10.1109/WoWMoM.2018.8449797

[3] Pedro Heleno Isolani, Nelson Cardona, Carlos Donato, Johann Marquez-Barja, Lisandro Zambenedetti Granville, and Steven Latre. 2019. SDN-based Slice Orchestration and MAC Management for QoS delivery in IEEE 802.11 Networks. In *2019 Sixth International Conference on Software Defined Systems (SDS).* IEEE, Rome, Italy, 260–265. https://doi.org/10.1109/SDS.2019.8768642

[4] P. H. Isolani, N. Cardona, C. Donato, G. A. Perez, J. M. Marquez-Barja, L. Z. Granville, and S. Latre. 2020. Airtime-Based Resource Allocation Modeling for Network Slicing in IEEE 802.11 RANs. *IEEE Communications Letters* 24, 5 (May 2020), 1077–1080. https://doi.org/10.1109/LCOMM.2020.2977906

[5] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. http://arxiv.org/abs/1412.6980 arXiv:1412.6980 [cs].

[6] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2019. Continuous control with deep reinforcement learning. http://arxiv.org/abs/1509.02971 arXiv:1509.02971 [cs, stat].

[7] Knud Lueth. 2020. State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time. https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/

[8] Gabriel Pimenta de Freitas Cardoso, Paulo Henrique Portela de Carvalho, and Paulo Roberto de Lira Gondim. 2023. Deep reinforcement learning for resource allocation of mobile communication systems with device-to-device underlay. *International Journal of Communication Systems* (March 2023), e5476. https://doi.org/10.1002/dac.5476

[9] Matías Richart, Javier Baliosian, Joan Serrat, and Juan-Luis Gorricho. 2020. Resource Allocation and Management Techniques for Network Slicing in WiFi Networks. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*. 1–6. https://doi.org/10.1109/NOMS47738.2020.9110407 ISSN: 2374-9709.

[10] Matías Richart, Javier Baliosian, Joan Serrat, Juan-Luis Gorricho, and Ramón Agüero. 2019. Slicing in WiFi Networks Through Airtime-Based Resource Allocation. *Journal of Network and Systems Management* 27, 3 (July 2019), 784–814. https://doi.org/10.1007/s10922-018-9484-x

[11] Reijer van Harten. 2023. Empower-RL. https://github.com/reijervharten/empower-rl

[12] Kamran Zia, Alessandro Chiumento, Paul Havinga, Roberto Riggio, and Yanqiu Huang. 2023. QoS Aware Slice Resource Management using Deep Reinforcement Learning in IoT Networks. (2023).