RateRL: A Framework for Developing RL-Based Rate Adaptation Algorithms in ns-3

Ruben Queiros[1,2(✉)], Luís Ferreira[1,2], Helder Fontes[2], and Rui Campos[1,2]

[1] Faculdade de Engenharia da Universidade do Porto, Porto, Portugal
[2] INESC TEC, Porto, Portugal
{ruben.m.queiros,luis.m.martins,helder.m.fontes,rui.l.campos}@inesctec.pt

Abstract. The increasing complexity of recent Wi-Fi amendments is making the use of traditional algorithms and heuristics unfeasible to address the Rate Adaptation (RA) problem. This is due to the large combination of configuration parameters along with the high variability of the wireless channel. Recently, several works have proposed the usage of Reinforcement Learning (RL) techniques to address the problem. However, the proposed solutions lack sufficient technical explanation. Also, the lack of standard frameworks enabling the reproducibility of results and the limited availability of source code, makes the fair comparison with state of the art approaches a challenge. This paper proposes a framework, named RateRL, that integrates state of the art libraries with the well-known Network Simulator 3 (ns-3) to enable the implementation and evaluation of RL-based RA algorithms. To the best of our knowledge, RateRL is the first tool available to assist researchers during the implementation, validation and evaluation phases of RL-based RA algorithms and enable the fair comparison between competing algorithms.

Keywords: Wireless Networks · ns-3 · Deep Reinforcement Learning · Machine Learning Tool

1 Introduction

The new configuration parameters available in the most recent Wi-Fi amendments allied to the high variability and asymmetry of the radio channel, make Rate Adaptation (RA) and the optimal configuration of these parameters challenging. Recent RA algorithms in Wi-Fi are often developed considering simplified simulations that are not always well described. This poses a challenge for the implementation and comparison of different RA algorithms. For example, the Network Simulator 3 (ns-3) implements a significant amount of existing state-of-the-art RA algorithms. However, only a few of these algorithms can be used for recent Wi-Fi versions subsequent to IEEE 802.11n. Additionally, many

J.-L. Guisado-Lizar et al. (Eds.): SIMUtools 2023, LNICST 519, pp. 287–298, 2024.
https://doi.org/10.1007/978-3-031-57523-5_22

recent RA algorithms are based on Machine Learning (ML), which can make them difficult to implement and understand. In some cases, the authors of these algorithms do not even provide the source code or the training dataset, which poses an obstacle to accurately reproduce the obtained results. When developing novel ML-based solutions the authors should consider the following practices: 1) clearly describe the problem that the proposal is trying to solve and justify any underlying assumptions; 2) systematically devise procedures that ensure the reproducibility of results; 3) clearly state and justify the metrics used for the evaluation; 4) expose the dataset used in the training process of the algorithm; and 5) have one or more baseline models to compare with.

The use of Reinforcement Learning (RL) and other ML techniques within the wireless networks research area has emerged as a way to further improve the network Quality of Service (QoS). Leveraging these techniques, the research community has been optimising the network performance, reducing latency, and ensuring efficient resource allocation [11,17]. A high percentage of the current telecommunications infrastructures have begun to invest and test ML algorithms for supporting the network operation and business decisions [12]. However, the techniques employed are not standardized, resulting in a lack of foundational principles that would enable the transfer of knowledge from previous initiatives to more recent ones. When compared with other fields that pioneered the use of ML, such as computer vision or natural language processing, wireless networks research started using ML at a later stage. Thus, it is common within the existing literature to identify ML-based solutions that lack the use of good practices stated above. A predominant challenge relates to the scarcity of technical details provided in published works, which hampers the reproducibility of results. Furthermore, the limited availability to the source code and the dataset used, precludes the validation and extension of prior findings. These issues collectively emphasize the need for a more cohesive and systematic approach to integrate RL into wireless networks.

The main contribution of this paper is RateRL, a framework to support the development of RL-based RA algorithms. We illustrate the use of RateRL through a practical use case, employing the Data Driven Algorithm for Rate Adaptation (DARA) proposed in [15]. Our framework integrates well-known RL libraries, such as TensorFlow Agents and OpenAI Gym [3] with ns-3 [16], using ns3-gym [8] to interface all the components. RateRL enables the task automation to identify the hyperparameter configuration that maximizes the expected cumulative reward of the RL algorithm, while offering real-time feedback of the training process. The code and ns-3 scripts are publicly available, facilitating the efforts of future research works to build upon RateRL.

The rest of the paper is organised as follows. Section 2 presents the background. Section 3 addresses the related work. Section 4 explains the RateRL framework. Section 5 illustrates the use of RateRL considering DARA as the use case. Finally, Sect. 5 provides some concluding remarks and points out the future work.

2 Background

This section refers to some background concepts that are necessary to understand RateRL. It starts with an overview of RL, namely the Q-learning algorithm and existing frameworks to implement these techniques. Then, a brief description of the network simulator ns-3 is provided.

2.1 Reinforcement Learning

The Fig. 1 illustrates the RL model, which consists of the environment and agent elements. These elements communicate based on action, state and reward signals. Based on the observations (states) retrieved from the environment, the agent learns to make decisions (actions) and it evaluates these decisions by considering a reward. Learning the policy that maps an action for an environment state that maximises the total cumulative reward for each (action, state) combination, is the objective of RL. The literature has a variety of learning algorithms available. The recent works that address the RA problem [5,6,14,15] use the classic Q-learning algorithm and show better results when compared to traditional algorithms such as Minstrel-HT [7], the default RA algorithm for Linux systems.

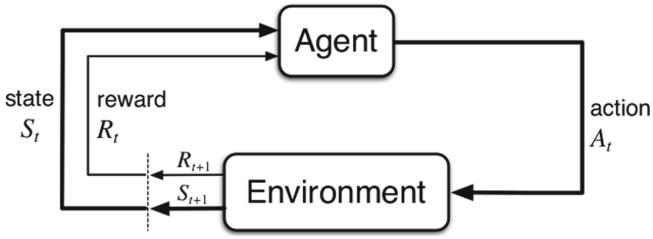


Fig. 1. Reinforcement Learning loop diagram.

As a model-free algorithm, Q-learning [18] learns by making mistakes and does not rely on any past environment information. It is only in a discrete action space that Q-learning works. Finding the best policy that maximises the expected cumulative reward is the objective of Q-learning. The Q-function values are updated using Eq. 1, through trial and error. The expected cumulative reward when the agent selects action a in state s is represented using $Q(a, s)$, considering that following actions are chosen according to the policy that was learnt. The reward for taking action a in state s is represented by $r(a, s)$, and $\max_{a \in A} Q(a, s_{new}), \forall a \in A$ is the maximum possible reward of the new state s_{new}, which is the result of the current action; s_{new} is the new state and a_{all} represents every $a \in A$. The learning rate α determines the rate at which new values update the total Q-value. Finally, the discount factor $\gamma \in [0, 1]$ determines the importance of future rewards in the calculation of the expected cumulative reward.

$$Q(a,s) \leftarrow (1-\alpha)Q(a,s) + \alpha[r(a,s) + \gamma \max_{\forall a \in A} Q(a, s_{new})] \quad (1)$$

2.2 Reinforcement Learning Frameworks

There are multiple and extensively validated frameworks available to implement Q-learning or other RL-based learning algorithm. TensorFlow [1] and PyTorch [13] are two prominent deep learning frameworks widely adopted for building and training ML models, including RL agents. TensorFlow, developed by Google, offers a comprehensive ecosystem of tools and libraries that facilitate the creation of neural networks and the optimisation of their parameters. TensorFlow Agents (TF-Agents) is an extension of TensorFlow specifically designed to enable the construction of RL agents. TF-Agents [9] provides reusable components for defining agent behaviour, defining the environment interaction loop, and implementing various state-of-the-art RL algorithms. Similarly, PyTorch, developed by Facebook's AI Research lab, provides a dynamic computational graph that simplifies model development and experimentation. Gym is an open-source toolkit developed by OpenAI, designed to facilitate the development and testing of RL algorithms. It provides a collection of environments (simulated scenarios) that allow researchers and developers to experiment with and benchmark various RL techniques as well as to develop their own custom environment making it a valuable resource that can interface with the popular network simulator ns-3.

2.3 The Network Simulator 3

ns-3 [16] is a well-known open-source network simulation tool and one of the most used wireless network simulators. ns-3 was driven by a desire to model networks in a way that best suits network research and learning. It aims to provide highly accurate and scalable network simulation capabilities for studying various aspects of networking protocols, communications methods, and network behaviour. Hence, it was not developed with ML or artificial intelligence in mind. There is currently no official framework for integrating with prevalent ML tools. Initial community efforts [8,19] were made to bridge the gap between the network simulator and popular ML frameworks such as TensorFlow, PyTorch and Gym. The resulting tools, named ns3-ai and ns3-gym, are detailed in Sect. 3.

3 Related Work

The related work is presented in this section. First, we overview existing reinforcement learning frameworks for networking and then we review the state of the art in RL-based solutions for the RA problem.

3.1 Reinforcement Learning Frameworks for Networking

To the best of our knowledge, few frameworks that integrate Reinforcement Learning frameworks with network simulators have been proposed in the state of the art. In [8] the authors develop a sockets-based interface between ns-3 and OpenAI Gym to encourage the usage of RL in networking research. To overcome the lack of flexibility imposed by ns3-gym, the authors in [19] proposed ns3-ai, a Python module that allows the interconnection between any artificial intelligence framework with ns-3, using a higher efficiency mechanism based on shared-memory. However, since this is a high flexibility and efficiency data interaction framework researchers using ns3-ai have to adapt to implement their own development environment to address their specific problem, resulting in slower algorithm development or simplistic simulation environments, which reduces the solution realism. On the other hand, the authors in [2] developed the PRISMA framework, which is tailored to the distributed packet routing problem, on top of ns3-gym, extending the problem to a Multi-Agent Deep Reinforcement Learning approach. Therefore, researchers would need to modify PRISMA's design if the objective was to address the RA problem.

3.2 Reinforcement Learning for Rate Adaptation

The RA problem has been addressed using several different solutions, in particular for Wi-Fi networks. Some use classical heuristic approaches [4,7] while recent works have been using RL-based techniques [5,6,10,14,15]. In [5], the authors developed a Q-learning based link adaptation solution that addresses RA together with other configuration parameters such as the channel bandwidth and number of spatial streams, outperforming state of the art solutions in terms of throughput. Despite implementing it in a network interface card and evaluating their solution in an experimental setting, the authors do not mention any standard framework that was used to implement their solution. Also, the source code is not publicly available. In [6,14] the authors use ns-3 to implement their RL-based RA solutions with the help of ns3-gym and ns3-ai frameworks. However, both works do not provide any information with regards to the training process or hyperparameter configuration. Moreover, despite simplistic simulation scenarios for evaluation, they are not sufficiently well described, posing an obstacle to accurately reproduce the obtained results.

3.3 Summary

Despite the good results of recent works that boost the overall network performance, the difficulty to replicate the results of the proposed solutions is common among existing works. Typically, they lack implementation details – i.e. the code is not open source – and training process description. Moreover, within the identified works, the results of the proposed solutions are not compared with other RL-based RA algorithms.

4 RateRL Framework

In this section, we present the RateRL framework, including its architecture and components.

The RateRL framework was designed considering design principles similar to the PRISMA framework [2], such as the achievement of realistic wireless networks simulation environments due to the usage of ns-3 and the development with a modular approach, which makes fast prototyping of RL-based RA algorithms possible.

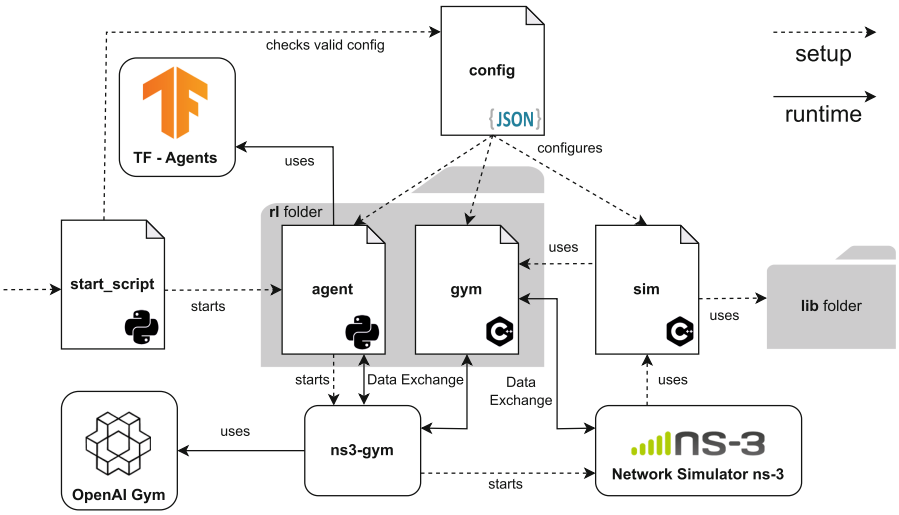


Fig. 2. RateRL architecture diagram.

In Fig. 2 we present the RateRL architecture and how its components interact with each other. The programming languages used to code RateRL were Python and C++. The Python files hold every essential setup procedures and parsing mechanisms required for the proper execution of the system, the core ML components and most of the data collection processes. The files written in C++, are mainly related to the configuration and utilisation of the ns-3 simulator together with some files that interact with the ns3-gym interface.

We now detail the RateRL components and instruct the reader on how to use RateRL. The starting point of the RateRL framework is the `start_script` file. A detailed description of the role of each component is presented below:

- `start_script` checks if the configuration file is valid. It also sets up the results folder where the logs of the simulation or training are stored. Finally, it starts the agent component.
- `config` is a JSON file that holds the configuration parameters relevant for each of the other files within the framework, such as the `agent`, `gym` and `sim`.

- `agent` interacts with the TF-Agents framework using the environment data that is retrieved from the ns-3 simulation. It starts the ns-3 environment using the ns3-gym interface.
- `sim` defines the ns-3 simulation script. It uses the `lib` folder, where multiple utility functions are defined to ease aspects such as the simulation configuration and data collection.
- `gym` configures the ns-3 environment defining the observation and action data shapes. It executes the action that comes from the `agent`, and establishes the data collection methods that are used to collect the observation from the ns-3 simulation as well as the metrics to calculate the reward.

The installation process is documented in the public repository, inside the `install` folder, and it assumes the user does not have any of the required dependencies installed. The user can run the agent in two different modes:

- **Training Mode** begins by addind the trajectories gathered from the simulation to the replay buffer. A trajectory comprises three components: 1) a time step which represents the first observation of the environment; 2) the action step which represents the chosen action after taking into account the previous time step; and 3) the subsequent time step, which represents the new observation and the reward consequent from the previous action step). Until the end of the episode, which in this case also signifies the conclusion of the simulation, this replay buffer is filled while the simulation is running. During this procedure, the agent is trained by randomly selecting a set number of trajectories defined by the hyperparamenter *batch size* from the replay buffer, updating the weights accordingly and raising the train step counter. Next, the user can modify how frequently training occurs, how many episodes the simulation runs, and how epsilon greedy adapts during training. It is feasible to save the progress using a checkpoint so that the policy's present state can be received at a later time, should the user choose to pause the training or finish it earlier.
- **Evaluation Mode** assumes a fixed epsilon greedy factor of 0 to avoid exploratory attempts, and loads the previously trained policy. However, this mode is not prepared for simulation scenarios that dynamically change requiring an online learning approach. This will be the subject of future work.

Regardless of the mode used, RateRL saves simulation logs with the throughput of every existing communication link as well as the nodes positions, with a configurable periodicity.

5 Using the RateRL Framework

In this section we use the implementation, training and evaluation of the DARA algorithm [15] as a use case to illustrate the utilisation of the RateRL framework.

5.1 DARA Overview

DARA [15] is a RL-based RA algorithm developed for the IEEE 802.11n amendment. It considers scenarios with Single Input Single Output and fixed channel bandwidth of 20 MHz, using long Guard Interval. The valid actions are the first 8 Modulation and Coding Schemes (MCS). The state is the average SNR value considering the Acknowledgement frames that originate from the receiver node. Finally, the reward is a function of the Frame Success Ratio (FSR) and the chosen MCS, to value the highest possible MCS without compromising the FSR.

5.2 Simulation Settings

We configured the preliminary validation scenario defined in [15]. In this scenario, we have a stationary node and a moving node. In the beginning of the simulation, the nodes start close to each other, and their distance increases throughout the simulation period. In this way we stimulate the algorithm with a wide range of SNR values. The algorithm is then compared in terms of throughput with other RA algorithms implemented in ns-3 such as Minstrel-HT (MIN) and the Ideal (ID) algorithm. All the other main simulation configuration parameters are presented in Table 1.

Table 1. Simulation Configuration Parameters.

Configuration Parameter	Value
Wi-Fi Standard	IEEE 802.11n
Propagation Delay Model	Constant Speed
Propagation Loss Model	Friis
Frequency	5180 MHz
Channel Bandwidth	20 MHz
Transmission Power	20 dBm
Wi-Fi MAC	Ad-hoc
Traffic	UDP, generated above link capacity
Packet Size	1400 Bytes of UDP Payload

5.3 Training and Hyperparameter Tuning

DARA was trained and evaluated on a ASUS ROG G14 Laptop with a Ryzen 9 5900HS (8 cores up to 4.6 GHz), 32 GB RAM and a NVIDIA RTX 3060 GPU. In this illustrative example the hyperparameter configuration chosen is defined in Table 2. The simulations were 60 s long, for a total of 15 episodes.

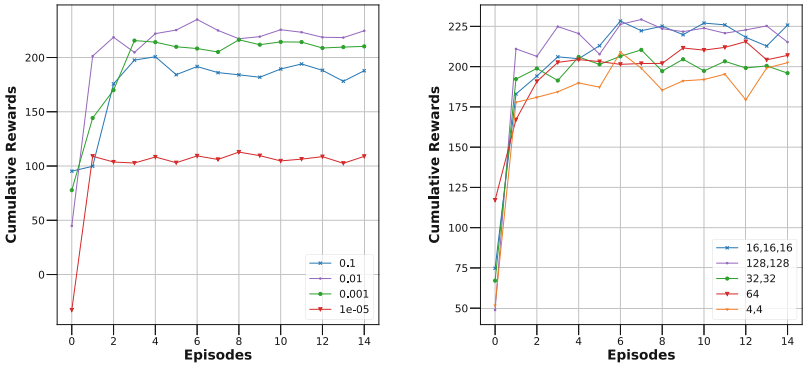
The hyperparameter tuning is an essential part of any machine learning model training. To this end, we assess how different values could benefit the

Table 2. DQN Learning Algorithm main Parameters.

Parameter	Value
Observation Space	One-dimensional scaled float (0.0–1.0)
Action Space	One-dimensional integer (0–7)
Optimiser	Adam
Loss Function	Mean Square Error
Epsilon Greedy	Fixed at 0.1
Discount Factor	Fixed at 0.5
Replay buffer	size of 10^6
Batch Size	64

final DARA performance. In this work, comparisons between different values of learning rate and the hidden layer architecture of the neural network were carried out.

These comparisons were not extensive, thus the performance of DARA could be further improved with a more in depth tuning, despite the simple scenario, which shows that good results are achievable with few training episodes. However, the objective of this work is to show that RateRL can be used to compare different hyperparamenter configurations and assess their impact in the performance of the algorithm.



(a) Learning Rate comparison training. (b) Hidden Layer architecture training.

Fig. 3. Hyperparameter tuning trainings.

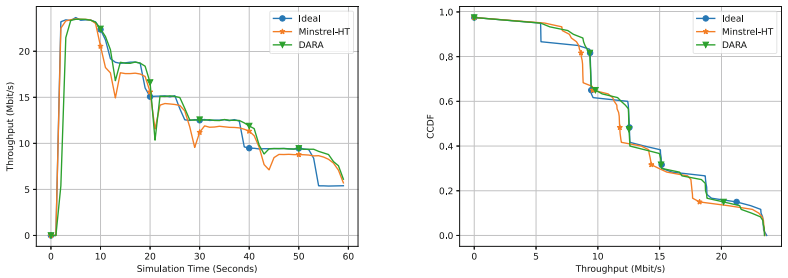
Figure 3a shows the cumulative reward over 15 training episodes with 4 different learning rate configurations, using two hidden layers with 32 units each. The results show that a learning rate of 0.01 is consistently better than the other options. After defining the used learning rate value, additional trainings were performed to fine tune the hidden layer architecture.

Figure 3b shows the results of this training, with 5 different options being evaluated. Despite the similarities in performance it was decided to choose as the final configuration the one which finished with highest cumulative reward by the end of the 15 episodes training. Therefore, a learning rate of 0.01 and a neural network with 3 hidden layers of 16 units each was defined.

5.4 Simulation Results

Using the resulting policy from the training that was detailed in the previous sections, we used RateRL to evaluate how the performance of DARA is compared to other popular RA algorithms such as Minstrel and Ideal. Fig. 4a shows the throughput throughout the simulation period and Fig. 4b its complementary cumulative distribution function.

The results show that RateRL can be used to evaluate the performance and comparison of RL-based RA algorithms with other state of the art RA solutions. The average throughput of Ideal was of 13.45 Mbit/s and Minstrel was of 13.07 Mbit/s. DARA achieved an average throughput of 13.52 Mbit/s, an increase of 3.4% over Minstrel and similar throughput when compared with Ideal. To conclude, we managed to successfully implement train and evaluate DARA using RateRL.



(a) Throughput (b) Throughput CCDF

Fig. 4. Simulation Results using the resulting hyperparamenter configuration.

6 Conclusion

This paper presented RateRL, the first framework designed for assisting the development, validation and evaluation of RL-based RA algorithms. We demonstrated the use of RateRL in the whole development cycle of an RL-based RA algorithm, using the state of the art DARA algorithm as a use case. Our objective with RateRL is to provide a framework for developing future RL-based RA solutions and enable their direct comparison with state of the art or related

solutions. The RateRL framework is open source and it is publicly available on Gitlab[1].

As future work, we plan to migrate to ns3-ai to support other popular ML frameworks. Also, we aim to extend RateRL to use other popular RL algorithms such as Deep Deterministic Policy Gradient and Proximal Policy optimisation.

Acknowledgements. This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project LA/P/0063/2020. The first author thanks the funding from FCT, Portugal under the PhD grant 2022.10093.BD.

References

1. Abadi, M., et al.: {TensorFlow}: a system for {Large-Scale} machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 265–283 (2016)
2. Alliche, R.A., Barros, T.D.S., Aparicio-Pardo, R., Sassatelli, L.: PRISMA: a packet routing simulator for multi-agent reinforcement learning. In: 2022 IFIP Networking Conference (IFIP Networking), pp. 1–6. IEEE (2022)
3. Brockman, G., et al.: OpenAI gym (2016)
4. Byeon, S., Yoon, K., Yang, C., Choi, S.: STRALE: Mobility-aware PHY rate and frame aggregation length adaptation in WLANs. In: IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, pp. 1–9 (2017). https://doi.org/10.1109/INFOCOM.2017.8056965
5. Chen, S.C., Li, C.Y., Chiu, C.H.: An experience driven design for IEEE 802.11ac rate adaptation based on reinforcement learning. In: IEEE INFOCOM 2021 - IEEE Conference on Computer Communications, pp. 1–10 (2021). https://doi.org/10.1109/INFOCOM42981.2021.9488876
6. Cho, S.: Reinforcement learning for rate adaptation in CSMA/CA wireless networks. In: Park, J.J., Fong, S.J., Pan, Y., Sung, Y. (eds.) Advances in Computer Science and Ubiquitous Computing. Lecture Notes in Electrical Engineering, vol. 715, pp. 175–181. Springer, Singapore (2021). https://doi.org/10.1007/978-981-15-9343-7_24
7. FietKau, F.: Minstrel_HT: new rate control module for 802.11n [LWN.net] (2010). https://lwn.net/Articles/376765. Accessed 20 May 2022
8. Gawłowicz, P., Zubow, A.: 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, pp. 113–120 (2019)
9. Guadarrama, S., et al.: TF-Agents: A library for reinforcement learning in TensorFlow. https://github.com/tensorflow/agents (2018), https://github.com/tensorflow/agents. Accessed 25 June 2019
10. Karmakar, R., Chattopadhyay, S., Chakraborty, S.: SmartLA: reinforcement learning-based link adaptation for high throughput wireless access networks. Comput. Commun. **110**, 1–25 (2017)
11. Kulin, M., Kazaz, T., De Poorter, E., Moerman, I.: A survey on machine learning-based performance improvement of wireless networks: PHY, MAC and network layer. Electronics **10**(3), 318 (2021)

[1] https://gitlab.inesctec.pt/pub/ctm-win/raterl.

12. Mahmoud, H.H.H., Ismail, T.: A review of machine learning use-cases in telecommunication industry in the 5G era. In: 2020 16th International Computer Engineering Conference (ICENCO), pp. 159–163 (2020). https://doi.org/10.1109/ICENCO49778.2020.9357376
13. Paszke, A., et al.: PyTorch: an imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems, vol. 32 (2019)
14. Pratama, M.H.B., Nakashima, T., Nagao, Y., Kurosaki, M., Ochi, H.: Experimental evaluation of rate adaptation using deep-Q-network in IEEE 802.11 WLAN. In: 2023 IEEE 20th Consumer Communications & Networking Conference (CCNC), pp. 668–669. IEEE (2023)
15. Queirós, R., Almeida, E.N., Fontes, H., Ruela, J., Campos, R.: Wi-fi rate adaptation using a simple deep reinforcement learning approach. In: 2022 IEEE Symposium on Computers and Communications (ISCC), pp. 1–3 (2022). https://doi.org/10.1109/ISCC55528.2022.9912784
16. Riley, G.F., Henderson, T.R.: The ns-3 network simulator. In: Wehrle, K., Gunes, M., Gross, J. (eds.) Modeling and Tools for Network Simulation, pp. 15–34. Springer, Berlin (2010). https://doi.org/10.1007/978-3-642-12331-3_2
17. Szott, S., et al.: Wi-Fi meets ML: a survey on improving IEEE 802.11 performance with machine learning. IEEE Commun. Surv. Tutorials **24**(3), 1843–1893 (2022)
18. Watkins, C.J., Dayan, P.: Q-learning. Mach. Learn. **8**, 279–292 (1992)
19. Yin, H., et al.: ns3-AI: fostering artificial intelligence algorithms for networking research. In: Proceedings of the 2020 Workshop on ns-3, WNS3 2020, pp. 57-64. Association for Computing Machinery, New York (2020). https://doi.org/10.1145/3389400.3389404