FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

On the Impact of Computational Delays on RL-based Rate Adaptation through Improved ns-3 Digital Twins

João Paulo Ferreira Pinto

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Hélder Martins Fontes

July 31, 2023

Abstract

Over the past 25 years since its inception, IEEE 802.11, colloquially referred to as Wi-Fi, has significantly evolved and it is today the most prevalent wireless communication technology. This evolution is the result of the increasing demand for higher throughput in communications, which led to several new amendments being added to the original standard. Rate Adaptation (RA) refers to the dynamic adjustment of the data transmission rate in a wireless connection. Traditional RA techniques, however, are becoming impracticable because of the great number of configuration parameters and variability of wireless channels. Some new techniques have been proposed, regarding the use of Machine Learning (ML) algorithms, specifically Reinforcement Learning (RL), as an alternative method for RA.

A RL-based approach for RA is a relatively new technique. In fact, state-of-the-art algorithms are typically tested in simulation under ideal conditions, with documentation describing only the fundamental stages of the algorithms. Other implementation-specific details are generally not mentioned, which makes the accurate reproduction of the algorithms an impossible task. One of these conditions that can negatively impact the effectiveness of a RA Algorithm (RAA) is computational delays, which are especially detrimental in rapidly changing network environments. Delays in algorithm execution may be detrimental to its effectiveness, thereby decreasing network performance. Computational delays not only slow down the algorithm's speed, but may also hinder their responsiveness to variations in link quality. Outdated information regarding link quality can have a negative effect on the selection of rates, resulting in a decrease of network throughput. Additionally, limitations of the device's hardware may restrict the algorithm's processing, which may result in more expressive delays and negatively impact the RAA's performance and efficiency even further.

In this dissertation, we conducted an analysis of an existing dynamic RAA that uses RL. The primary goal of this work was to investigate the impact of computational delays on the performance of RAAs. To accomplish this goal, we employed the ns-3 software to train, test, and validate the algorithm using simulations that closely mirrored real-world conditions. We incorporated delay statistics into our experiments to account for their effects. As expected, we noticed that delays can substantially degrade the algorithm's expected performance, resulting in increased packet loss and a lower average throughput, depending on the hardware and the experimental implementation techniques used. Moreover, we noted that training the algorithm with delays presented additional challenges, resulting in a less effective learning process.

Resumo

Nos últimos 25 anos desde a sua criação, a norma IEEE 802.11, popularmente conhecida como Wi-Fi, evoluiu significativamente e é atualmente a tecnologia de comunicação sem fio mais prevalente. Essa evolução é resultado da crescente procura por maiores débitos nas comunicações, o que levou à adição de várias novas emendas à norma original. Rate Adaptation (RA) refere-se ao ajuste dinâmico da taxa de transmissão de dados numa ligação sem fios. No entanto, devido à grande combinação de parâmetros de configuração, bem como à variabilidade dos canais sem fios, os métodos tradicionais têm-se tornado impraticáveis. Foram propostas algumas técnicas inovadoras, relativamente ao uso de algoritmos de *Machine Learning* (ML), especificamente *Reinforcement Learning* (RL), como um método alternativo para RA.

Uma abordagem baseada em RL para RA é uma técnica relativamente nova. De facto, as soluções do estado da arte geralmente são validadas em simulações com condições ideais, com a documentação descrevendo apenas as etapas fundamentais dos algoritmos. Detalhes específicos de implementação geralmente não são mencionados, o que torna impossível reproduzir precisamente os algoritmos em simulação. Uma dessas condições que pode afetar negativamente a eficácia de um algoritmo de RA são os atrasos computacionais, que são especialmente prejudiciais em ambientes de rede muito dinâmicos. Os atrasos computacionais não só abrandam a velocidade do algoritmo, como também podem prejudicar a sua capacidade de resposta a variações na qualidade da ligação. Informações desatualizadas sobre a qualidade da mesma podem afetar a seleção das taxas, resultando numa diminuição do *throughput* da rede. Adicionalmente, limitações no hardware do dispositivo podem restringir o processamento do algoritmo, o que pode resultar em atrasos mais expressivos e impactar ainda mais o desempenho e a eficiência do RAA.

Nesta dissertação, foi realizada uma análise de um algoritmo de *Rate Adaptation* (RAA) dinâmico pré-existente que utiliza RL. O objetivo principal desta dissertação foi investigar o impacto dos atrasos computacionais no desempenho dos RAAs. Para atingir esse objetivo, o software ns-3 foi utilizado para treinar, testar e validar o DARA por meio de simulações que refletem de perto as condições do mundo real. Para tal, incorporamos os atrasos nas simulações para estudar os seus efeitos. Como esperado, observámos que os atrasos podem degradar substancialmente o desempenho esperado do algoritmo, resultando em aumento da perda de pacotes e um menor débito da ligação, dependendo do hardware e das técnicas de implementação experimental utilizadas. Além disso, notámos que treinar o algoritmo com atrasos apresentou desafios adicionais, resultando em um processo de aprendizagem menos eficaz.

Acknowledgments

This dissertation marks not only the end of my academic journey but also the beginning of my professional path. Reflecting upon the past five years, a period that has presented both challenges and rewards, I can say I am filled with sincere gratitude towards those who have played a significant role in shaping both my academic and personal growth.

First and foremost, I would like to express my deepest appreciation to my supervisors, Hélder Fontes and Rúben Queirós, whose unwavering support and ongoing assistance have proved indispensable in the successful completion of this work, and I am truly grateful for their mentorship.

Furthermore, I extend my heartfelt gratitude to my friends for their companionship and support throughout this journey. The moments of collaboration, discussions, and shared experiences have not only made these five years more enjoyable, but also unforgettable.

Last but not least, my profound thanks go to my family. Their unconditional support, understanding, and encouragement have served as a constant source of strength for me, and defined the person I am today.

João Pinto

"The only person you are destined to become is the person you decide to be."

Ralph Waldo Emerson

Contents

List of Figures

List of Tables

Listings

Acronyms and Abbreviations

AP	Access Point
DARA	Data-driven Algorithm for Rate Adaptation
DL	Deep Learning
DRL	Deep Reinforcement Learning
IEEE	Institute of Electrical and Electronics Engineers
MAC	Medium Access Control
MCS	Modulation Coding Scheme
MIMO	Multiple-Input Multiple-Output
ML	Machine Learning
ns-3	Network Simulator 3
PDF	Probability Density Function
PHY	Physical
QAM	Quadrature Amplitude Modulation
QoS	Quality of Service
RA	Rate Adaptation
RAA	Rate Adaption Algorithm
RL	Reinforcement Learning
SISO	Single-Input Single-Output
SNR	Signal-to-Noise Ratio
Wi-Fi	Wireless Fidelity

Chapter 1

Introduction

1.1 Context

The IEEE 802.11 standard, commonly known by the brand name Wi-Fi, has seen great improvement since its original release in 1997 to keep up with the demand for wireless networks offering higher speeds and better coverage. Several versions of the IEEE 802.11 standard have emerged over the past two decades. Currently, Wi-Fi is in its 6th version (IEEE 802.11ax) and it is the most prevalent wireless communication technology. These updated versions of the standard have introduced new configuration parameters for both the physical (PHY) and medium access control (MAC) layers, such as more spatial streams and larger channel bandwidth. These advancements enable increased throughput and improved network reliability.

However, configuring these parameters optimally poses a challenge due to the channel's high variability and inherent unpredictability [1]. Among these parameters, the Modulation and Coding Scheme (MCS) plays a crucial role in determining the quality and stability of a Wi-Fi connection. Consequently, developing methods for automatic and optimised MCS configuration becomes vital in enhancing the network's Quality of Service (QoS). To address this challenge, one approach introduced is dynamic rate adaptation. This technique aims to dynamically select the most suitable physical data transfer rate based on specific measurements, such as link quality [1, 2] or based on frame loss [3] . This ensures optimal performance and QoS for wireless communication. Table 1.1 shows different configuration parameters and the corresponding PHY Rates for the IEEE 802.11n standard.

Rate Adaptation (RA) is not a new technique. In fact, algorithms for such a purpose already exist, such as Minstrel [4] and Iwlwifi [5]. Minstrel is the default Wi-Fi RA algorithm used in the Linux kernel, while Iwlwifi is the RA algorithm present in Intel chips. However, both have some drawbacks: Minstrel reacts with significant delays in scenarios where the link quality improves [1], leading to slower responses in dynamic and fast changing scenarios. Iwlwifi is also subject to some limitations, including no joint rate and bandwidth adaptation, as well as lack of scalability [1].

To address these problems with classic RA algorithms, Machine Learning (ML) techniques have been proposed as an alternative, with Reinforcement Learning (RL) being a prominent tech-

Table 1.1: IEEE 802.11n data rates.

MCS Index	Spatial Streams	Modulation Type	Coding Rate	PHY Rate (Mbit/s)			
				20 MHz Channel		40 Mhz Channel	
				800 ns GI	400 ns GI	800 ns GI	400 ns GI
0	1	BPSK	1/2	6.5	7.2	13.5	15
1	1	QPSK	1/2	13	14.4	27	30
2	1	QPSK	3/4	19.5	21.7	40.5	45
3	1	16-QAM	1/2	26	28.9	54	60
4	1	16-QAM	3/4	39	43.3	81	90
5	1	64-QAM	2/3	52	57.8	108	120
6	1	64-QAM	3/4	58.5	65	121.5	135
7	1	64-QAM	5/6	65	72.2	135	150
...	...	...	...	...	...	...	...
31	4	64-QAM	5/6	260	288.8	540	600

nique adopted. In [1], the authors propose a simple Deep Reinforcement Learning (DRL) approach for the automatic RA in Wi-Fi networks, named Data-driven Algorithm for Rate Adaptation (DARA), having achieved significant improvements regarding throughput when compared with Minstrel. However, most of the research in this field takes place in overly simplistic simulations, since network simulators do not provide accurate reproductions regarding the nodes' computational delays and hardware limitations, which can greatly affect the algorithm's overall performance and stability, which in turn pose a threat to the future deployment of this technology. All the work developed for this dissertation will use the aforementioned state-of-the-art algorithm as a case study for this problem.

1.2 Problem and Motivation

Machine Learning (ML) is currently a very prominent field in regard to RA. In spite of being a relatively new technique, ML-based algorithms have shown promising results when it comes to rate adaptation, as confirmed by the results in both [1] and [2].

However, despite improvements in performance, state-of-the-art ML-based RA algorithms often fail to consider the computational delays and processing time of real wireless devices, as most of the research has taken place in simplistic simulated scenarios, which are almost never a total accurate depiction of real-life situations and network environments. In fact, real-environment networks frequently exhibit instability and unpredictability. Therefore, experimental circumstances might present fresh problems and revelations that can negatively affect the algorithm's performance, hinder their responsiveness and affect the future deployment of this technology.

Running a RL-based algorithm in a wireless device can interfere with its normal operation, which can make the optimal MCS selection for the current link quality to be delayed by computational delays, impacting the performance of the algorithm. For example, if the algorithm takes too

long to choose an MCS, the chosen MCS may no longer be appropriate or useful for the present link quality, which is particularly impactful in fast changing network environments.

To the best of our knowledge, computational delays have not been considered or properly studied in simulated scenarios. The author in [6] raised first awareness of the execution time problem and the significance of considering computational delays when it comes to RL-based RA algorithms, a subject that has been disregarded in the state of the art. A preceding work [7] investigated the delays in the DARA method for experimental implementations and proposed a solution that significantly reduced the magnitude of such delays. However, no tests were conducted to evaluate the impact of the delays on the algorithm performance. This is where this work comes into play.

1.3 Objectives

The main goal of this dissertation is to study, evaluate and characterise the impact of computational delays on the performance of a RL-based RA algorithms. We will use a state-of-the-art algorithm, called DARA, which already has been subject to tests regarding its execution times, in an experimental setting in [7]. Additionally, this dissertation aims at evaluating the feasibility of adopting RL algorithms for RA in experimental scenarios, by using a network simulator to predict performance in real environments. Taking this into account, we can summarise our goals for this work with the following set of objectives:

- Creation of a simulation environment to train and test the RL-based algorithm in the Network Simulator 3 (ns-3), implementing the delays in the simulator that were previously characterised;
- Performance testing and evaluation of the algorithm while altering the computational delays, using ns-3 as a network simulator to reproduce and replicate particular scenarios where this type of algorithm fits;
- Assess the influence of computational delays on the efficacy of agent training.

1.4 Contributions

The main contributions of this work can be summarised as follows:

- **Enhancement of realism in simulation environments**: This work introduces a possible approach to incorporate computational delays in ns-3. By incorporating computational delays, this implementation bridges the gap between simulated scenarios and real-world complexities. It provides a more accurate representation of the challenges faced in real wireless networks, allowing for more reliable evaluations of RL-based RA algorithms. Thus, ns-3 acts as a digital twin of the physical world, where experimental scenarios can be replicated accurately and realistic predictions can be made in simulations.

- **Insights for RL-based RA algorithms**: The results provided by this work offer relevant insights into how computational delays can impact the performance of such algorithms. These results not only apply to RL-based RA algorithms, but can also have broader applications beyond the scope of rate adaptation.

1.5 Document Structure

The present document is structured in five chapters and it is organised as follows:

- Chapter 2: introduces the state-of-the-art, context, and prior relevant work in this context. It contains information on the IEEE 802.11 (Wi-Fi) standard, Reinforcement Learning, Rate Adaptation algorithms, and the essential tools and frameworks that were used to carry out our work, namely ns-3 and ns-3 gym.
- Chapter 3: details the proposed solution and methodology to address the problem, describes the simulation environment and the preliminary study, as well as a description of the implementation.
- Chapter 4: explains in greater detail the network simulations designed for validation and the results obtained, along with appropriate comparisons and analyses.
- Chapter 5: The final chapter outlines the key conclusions to be drawn from this document, as well as a glimpse into the future work of this dissertation.

Chapter 2

State of the Art

The present chapter covers the main topics deemed relevant for comprehending the problems exposed in Chapter 1. It is broken down into six sections:

- IEEE 802.11 – A brief overview of the IEEE 802.11 set of technical standards and the improvements and new functionalities introduced by the most recent releases;
- Reinforcement Learning – Explanation of how RL works and how it can be applied in the context of this work;
- Rate Adaptation – Motivation for using RA and overview of existing algorithms;
- Computational Delays – Overview of what computational delays are and their causes;
- Simulation Tools and Framework – Brief description of the simulation software used, including ns-3 and the ns-3 gym framework;
- Summary – A summary of the main ideas to be drawn from this chapter.

2.1 IEEE 802.11

IEEE 802.11 or Wi-Fi has evolved significantly over the last two decades, from 2 Mbps to gigabit speeds, representing a 1000-fold increase in throughput [8]. The standard has continuously advanced by introducing new protocols and configurable parameters, which support higher order of modulation and coding schemes such as 64 QAM, 256 QAM and 1024 QAM. Each new standard is built on previous standards with improvements in speed and reliability and, as capabilities are added to the original 802.11 standard, they become known by their version (e.g., Wi-Fi 3 and Wi-Fi 4). The most relevant amendment for this particular work is the IEEE 802.11n (also known as Wi-Fi 4) as it is the standard was used in previous research [1, 7].

A brief description of the most relevant standards is presented below:

- **IEEE 802.11 (1997)**: Released in 1997, this standard defined the protocol, which supported three physical layer technologies: infrared, operating at 1 Mbps; Frequency Hopping Spread

Spectrum (FHSS), supporting 1 Mbps; and Direct Sequence Spread Spectrum (DSSS), supporting both 1 Mbps and 2 Mbps data rates.

- **IEEE 802.11b**: This standard provided a maximum theoretical data rate of 11 Mbps. 802.11b was a direct extension of DSSS and used Complementary Code Keying (CCK) as its modulation technique, operating with frequencies between 2.4 and 2.5 GHz. The significant increase in throughput introduced by 802.11b, along with substantial price decrease, led to widespread adoption of 802.11b as a wireless local area networking technology.

- **IEEE 802.11a**: The 802.11a used the same core protocol as the original standard, however, operated at a frequency of 5 GHz with a channel bandwidth of 20 MHz. It used a 52-subcarrier Orthogonal Frequency Division Multiplexing (OFDM) and had a maximum theoretical data rate of 54 Mbps. Since the 2.4 GHz band was becoming congested, as a result of other devices using the spectrum, the 5 GHz frequency gave 802.11a a performance advantage. 802.11a products' high price, limited range, and incompatibility with 802.11b caused them to be poorly received.

- **IEEE 802.11g**: Introduced in 2003, 802.11g extended data rates up to 54 Mbit/s in the 2.4 GHz band. It is known as the successor of 802.11b because of its compatibility with this previous standard (i.e., 802.11b devices could connect to an 802.11g access point, although their connection was constrained by the capabilities of the 802.11b device) [8].

- **IEEE 802.11n**: 802.11n introduced significant improvements over the previous Wi-Fi standards, with increased network speed and reliability. Some of the features introduced by 802.11n include: the addition of multiple-input multiple-output antennas (MIMO) technology for separate spatial streams; frame aggregation; channel Bonding - technique to increase the throughput in wireless networks by means of using wider channels (20MHz to 40MHz); short Guard Interval that, when disabled, avoids signal loss from multipath effect; and a new maximum rate up to 600 Mbit/s in both the 2.4 GHz and 5 GHz bands.

- **IEEE 802.11ac**: An improvement over the previous standard, 802.11ac provided gigabit speed by strengthening the concepts of IEEE 802.11n. The standard adds supports for even wider channels, more MIMO spatial streams and high-density modulation (up to 256 QAM).

- **IEEE 802.11ax**: 802.11ax, or Wi-Fi 6, provides more wireless capacity and reliability, by using denser modulation schemes (1024 QAM and OFDMA), reduced subcarrier spacing and schedule-based resource allocation. It can reach theoretical rates of up to 9.6 Gb/s.

- **IEEE 802.11be**: IEEE 802.11be [9], also known as Wi-Fi 7, is the upcoming wireless standard that promises even greater capacity and reliability. 802.11be builds upon the advancements of Wi-Fi 6, introducing denser modulation schemes like 4096 QAM. With reduced subcarrier spacing and enhanced channel utilization, it will enable more efficient spectrum utilisation, allowing for higher data rates and improved network performance. Wi-Fi 7 is

expected to achieve theoretical speeds of up to 30 Gbit/s, making it ideal for bandwidth-intensive applications and the ever-increasing demands of modern wireless networks.

The new additions introduced by each amendment allowed the network to best fit its environment. It is important to note that it is not always appropriate to allocate the highest data rate for a Wi-Fi connection. For instance, if the link quality is low, then higher rates will inevitably induce more errors and decrease the success ratio, causing retransmissions and overall lower throughput.

The MCS is one of the parameters used to configure a Wi-Fi connection and consists of a combination of a modulation type and a coding rate. This index represents the level of throughput possible for a connection. Therefore, a lower MCS has a lower theoretical throughput, but also a lower error rate. On the other hand, a higher index allows high throughput but it is less reliable, thus should only be applied to cases where the link quality is optimal. Figure 2.1 represents the NIST BER Rate Model - a set of curves that relate the link quality to the Frame Success Ratio for each Wi-Fi data rate. In this sense, when configuring a network, there should be a trade-off between speed and reliability to achieve the best possible performance. One of the metrics that is most used to measure the wireless link quality is the Signal-to-Noise Ratio (SNR), which compares the level of a desired signal to the level of background noise, thus a higher SNR indicates a higher quality communication.

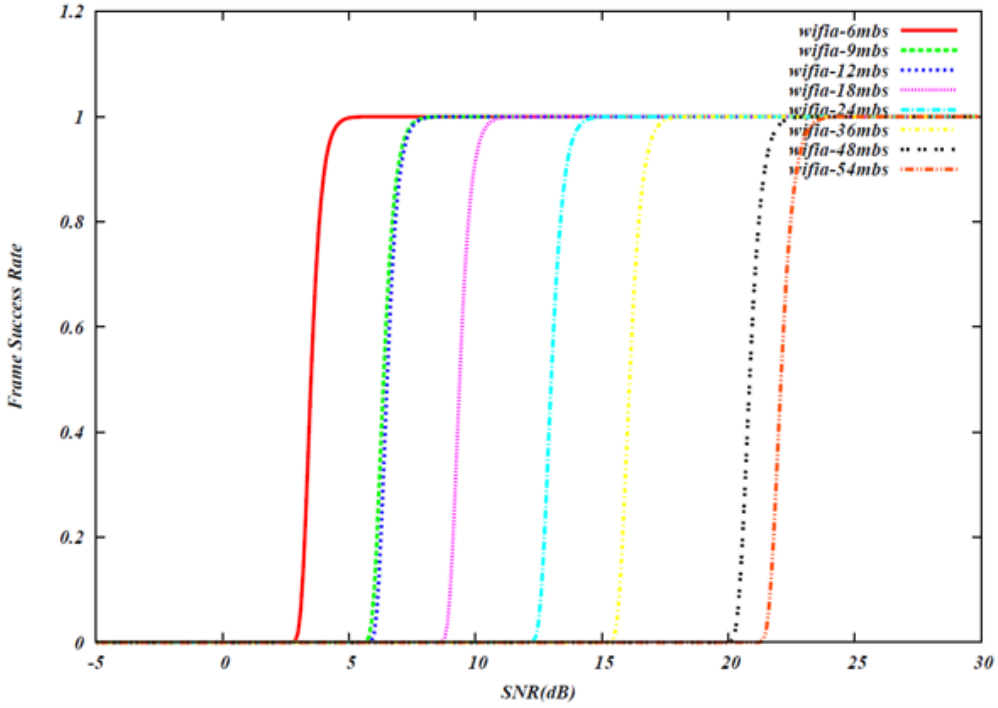


Figure 2.1: Frame Success Ratio vs. SNR (dB). [10]

2.2 Reinforcement Learning

The study of how computers can be used to replicate human learning processes is known as machine learning (ML) [11]. It is the study of how computers can acquire new knowledge and abilities, recognise existing information, and continuously improve performance and achieve self-improvement methods. For this reason, ML is an essential topic of study in artificial intelligence applications.

There are three main types of machine learning: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning [12] is a type of ML where the algorithm is trained on labeled data. As such, the algorithm learns to map inputs to outputs based on example input-output pairs. On the other hand, in unsupervised learning [12] the algorithm is trained on unlabeled data, learning how to find patterns without being given any specific output.

Reinforcement learning (RL) [13] refers to a sub-field of ML that enables systems to autonomously take actions in a dynamic environment through trial and error in order to maximise the collective rewards based on the feedback generated for individual activities. In the RL context, this feedback refers to a positive or negative notion reflected through rewards or punishments. Figure 2.2 represents the RL process through a feedback loop.

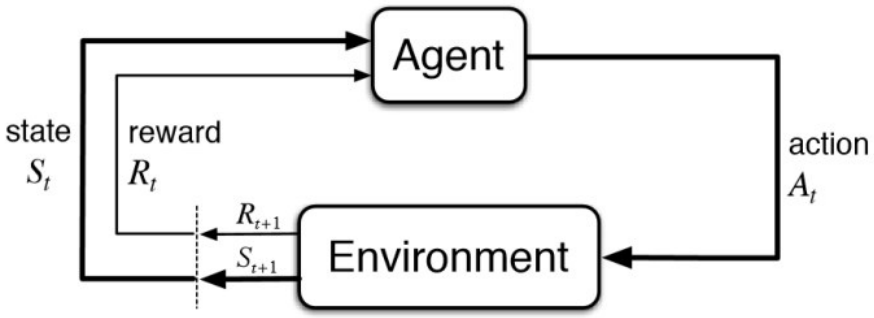


Figure 2.2: Reinforcement Learning Process. [14]

The agent and the environment make up the RL model. Signals of state, action, and reward are used to communicate between them. The (agent) evaluates the current state (s_t) and then takes an action (a_t) in an environment to achieve a specific goal. As a result of the performed task, the agent receives feedback as a reward or punishment (r_t). To summarise, the goal of RL is to learn an action policy for a given environment state that maximises the overall cumulative reward for each state/action pair.

Q-learning [15] is one the multiple algorithms available in the literature. It is a model-free algorithm since it learns through trial and error without any prior knowledge of the environment. It operates on a discrete action space and aims to find the optimal policy for maximising cumulative rewards. The algorithm updates its Q-function values using the equation 2.1:

$$Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha[r(s,a) + \gamma \cdot \max_{a \in A} \cdot Q(s_{new}, \alpha_{all}))] \tag{2.1}$$

Here, Q(s,a) represents the expected cumulative reward when the agent selects action a in state s, considering that future actions are chosen based on the learned policy. The term r(s, a) denotes the immediate reward obtained by taking action a in state s. The expression $\max_{a \in A} \cdot Q(s_{\text{new}}, \alpha_{\text{all}})$ calculates the maximum possible reward in the new state s_{new} resulting from the current action, where α_{all} represents every action in the action space. The learning rate α determines the rate at which new values update the total Q-value, while the discount factor $\gamma \in [0,1]$ determines the importance of future rewards in the calculation of the expected cumulative reward.

The main benefit of this technique is that it is focused on the long term. By having training data that is dynamically collected through the agent's response and experience, RL algorithms are able to tweak and adapt themselves to perform better, thereby maximising the rewards.

In the context of this dissertation, i.e., dynamic Rate Adaptation, the actions are different MCS, which in practice means different data rates. This action is based on the wireless environment of the device. The state is the current wireless link quality (SNR) of the given signal, and the reward is the success ratio and link throughput. To summarise, the agent will learn how to choose the proper Wi-Fi parameters in order to achieve the best link throughput possible, given a certain link condition.

Deep Reinforcement Learning

Deep Learning (DL) is another subfield of machine learning that is inspired by the neural networks that make up the human brain. Deep learning algorithms are composed of multiple layers of artificial neural networks, which allow them to learn and model increasingly complex patterns in the data. The term "deep" refers to the many layers of neural networks that are used in these algorithms. Deep reinforcement learning (DRL) combines deep learning's feature representation capability with reinforcement learning's decision-making capability to provide powerful end-to-end learning control capabilities [16]. DRL has achieved significant progress in several tasks that require processing high-dimensional data and generating optimal or near-optimal decisions over the last decade.

As explained before, in reinforcement learning, an agent learns to make decisions in an environment by interacting with that environment and receiving feedback in the form of rewards or penalties. The goal is for the agent to learn a policy that will maximise the cumulative reward over time. With DRL, this is made possible by the use of deep neural networks. In the context of this work, DRL-based algorithms can be applied to Wireless Communications in a wide variety of problems, like developing link adaptation algorithms in Wi-Fi networks.

2.3 Rate Adaptation

In order to provide reliable data reception regardless of changes in the transmission channel brought on by mobility or propagation effects, the IEEE 802.11 physical layer (PHY) defines a number of transmission features that can be selected and combined. The modulation and coding

scheme is one of these features, as well as the space-time block coding technique, the duration of the guard interval, the channel width and the number of spatial streams [17].

A transmission rate is determined by the combination of these several transmission parameters. The algorithms that choose these settings automatically are known as rate adaptation algorithms (RAAs) and may improve transmissions in terms of several metrics such as throughput, or energy consumption.

These types of algorithms have been subject to extensive research over the past years as rate adaptation is a critical aspect to a wireless network's performance. In fact, wireless channels are highly variable and can be influenced by a variety of conditions, including interference from other wireless devices, multi-path fading, and signal attenuation [4]. To summarise, rate adaptation is a trade-off problem: if the rate is too high, packets can be loss because of bit errors, but if the rate is too low, the wireless channel will not be completely utilized.

2.3.1 Classic and Heuristic-based algorithms

RAAs already exist, such as Minstrel and Iwlwifi. Minstrel is the default Wi-Fi RA algorithm used in the Linux kernel, while Iwlwifi is the RA algorithm present in Intel wireless adapters. These algorithms make rate choices taking into account wireless link metrics, such as the number of retransmissions, number of packets in the queue and Packet Error Ratio (PER). However, due to the large combination of configuration parameters, as well as the variability of wireless channels, these traditional methods are becoming impractical [1], having some limitations, which are briefly presented ahead.

The Minstrel algorithm [4] aims at achieving the highest throughput and consists of three parts: the retry chain mechanism, the rate decision process and the statistic calculations. The retry chain mechanism consists of four rate-count pairs. The algorithm sequentially transmits packets with different pairs until the packet is successfully transmitted or ultimately discarded if all attempts are unsuccessful. The rate selection step aims at choosing between normal transmission or sampling transmission. The last step, statistic calculations, is when Minstrel calculates the probability of success and expected throughput for each data rate. Some drawbacks of the Minstrel algorithm include significant reaction delays in scenarios where the link quality improves [1], which leads to slower responses in dynamic and fast changing scenarios.

Iwlwifi is the RAA present in Intel wireless chips. As mentioned in [17], MCS Scaling and Column Scaling are the two main components of the algorithm. MCS Scaling tries to improve throughput by only changing the MCS and Column Scaling aims at locating a better column, which is a combination of mode (legacy, SISO, MIMO), guard interval, and antenna configuration settings. The algorithm begins with the lowest transmission rate, having the highest reliability and alternates between MCS Scaling and Column Scaling stages, establishing a "search cycle". The main Iwlwifi limitations include no joint rate and bandwidth adaptation, as well as lack of scalability [2].

2.3.2 RL-based RAAs

ML approaches for RA have surged over the past few years, such as EDRA [2], SmartLA [18], and DARA [1].

2.3.2.1 SmartLA

SmartLA [18] is a rate adaptation algorithm for the IEEE 802.11ac wireless networking standard that is based on reinforcement learning. This algorithm tries to configure MCS, GI, CB, and the Level of Frame Aggregation to optimise the link rate. It uses the Frame Error Ratio (FER) and Bit Error Ratio (BER) as observation metrics.

The agent performs an action to change the state by configuring new values for the MCS, CB, Guard Interval (GI), and Level of Frame Aggregation (LFA) parameters. However, this does not imply that these four values are always changed. Data transmissions are made using the values in the state for the PHY and MAC parameters after the agent takes an action and the system changes to a new state. The system then computes a reward for the agent based on the BER. As a result, the lower the measured BER, the bigger the reward, positively reinforcing the agent's chosen value combination.

The authors of this work evaluated SmartLA against existing algorithms Minstrel and Minstrel-HT, having achieved remarkable results. The evaluation encompassed simulations and real-world experiments, with the simulations not fully considering constraints from actual wireless nodes and environments, such as computational delays. Overall, this comprehensive evaluation solidified the superiority of SmartLA over existing algorithms, reinforcing the reported remarkable results.

2.3.2.2 EDRA

EDRA [2] (Experience Driven Rate Adaptation) is a scalable, intelligent RA system that uses the DRL approach to optimise the MCS, CB, and NSS parameters. As observation metrics, the sub-frame loss, Received Signal Strength Indicator (RSSI), and Service Time Ratio (STR) of the bandwidth are considered.

The EDRA method was compared to the Iwlwifi driver, which incorporates the Iwl-mvm-rs adaption method, and Minstrel. They demonstrated how EDRA outperforms Iwl-mvm-rs and Minstrel by up to 821.4% and 242.8%, respectively, illustrating the effectiveness of using DRL techniques for wireless connection adaptation.

2.3.2.3 DARA

DARA [1] is a DRL algorithm that tries to maximise throughput in a Wi-Fi network. Their approach consisted of training an agent to learn the optimal MCS for a defined fixed time interval, based on the SNR of the received frames from the previous interval. For this algorithm, the authors define the DRL model as following:

- **State**: average SNR value observed. DARA keeps track of the number of received frames as well as their SNR. The average SNR for the actual time interval is then calculated. If the selected MCS is high enough that no frames are transmitted, SNR cannot be determined, so it assumes that SNR is 0 in that time interval and chooses the lowest MCS in the following time interval.
- **Action**: one of the eight MCS values available in IEEE 802.11n standard (MCS_0 to MCS_7)
- **Reward**: To prevent circumstances in which the agent picks either the lowest MCS to keep the maximum frame success ratio or the highest MCS even if frames are not delivered, the authors chose to combine the success ratio with the highest MCS achievable. The following equation is employed:

$$Reward = \frac{MCS_n}{MCS_7} \times FSR, n \in [0,1,...,7] \tag{2.2}$$

Moreover, ns-3 is used as a simulator, where a scenario is built with a saturated link of UDP traffic to ensure continuous traffic on the link for simulating the results. According to the authors, the results taken from the simulated tests conclude that DARA "achieved a similar performance when compared to its state-of-the-art counterparts, while adopting a simpler approach that does not require environment sampling and is standard-compliant". [1, Conclusions and Future Work]

2.4 Computational Delays

In this context, computational delays refer to the time taken algorithms to process and execute tasks, which can have significant implications in various applications. These delays can arise from various factors, including hardware limitations, algorithm complexity, data processing requirements, and communication overhead. Therefore, understanding and managing computational delays is crucial in designing efficient and responsive networking algorithms that can meet real-time requirements and deliver timely decision-making.

One of the primary causes of computational delays is the computational complexity of reinforcement learning algorithms. These algorithms often involve intensive mathematical operations, such as matrix multiplications, convolutions, and optimisation procedures, which can be time-consuming even on powerful hardware platforms. As the size and complexity of the learning models increase, the computational demands grow exponentially, leading to longer processing times and potential delays.

Furthermore, the availability and efficiency of hardware resources can also significantly impact computational delays. In traditional computing systems, the processing power of central processing units (CPUs) plays a critical role. However, the increasing adoption of specialised hardware accelerators, such as graphics processing units (GPUs) and tensor processing units (TPUs), can also significantly reduce the severity of computational delays.

The authors of [2] provide an overview of the query delays and training time for a DRL-based RA algorithm running on two different hardware devices. The query delay is the time interval between when the RA module begins to query the DRL agent and when it receives expected candidate rates. These delays ranged from 1.3 to 3.7 milliseconds (on average). It is worth noting that the delays described in this study were regarding powerful hardware systems.

In the dissertation [7], an experimental analysis is made of the delays resulting from running the DARA algorithm [1] in different wireless nodes from the W-iLab2 testbed, provided from the Fed4FIRE+ project [19]. Additionally, the author proposes a methodology that may be applied to reduce these computational delays and increase the efficiency of this type of algorithms. It was concluded that the two main causes of computational delays in the DARA model were the frequent usage of bash scripts and the time required to calculate the reward. His solution involved trying different implementation techniques to reduce the time it takes to access data. The results detail that the execution time was reduced significantly when compared to a simple implementation approach that disregards these delays. The final time intervals were around 60-80 ms for a complete time step (agent and environment side). This shed light on the importance of considering computational delays in this type of algorithms.

Despite the delays in DARA having been analysed in [7], there was no further research regarding how these delays affect its performance or efficiency. Additionally, both of the aforementioned studies were done for specific hardware configurations. Therefore, there is also no information in regard to how different hardware limitations would affect the performance of the algorithm under study.

To conclude, computational delays in RL-based networking algorithms are caused by factors such as algorithm complexity, hardware restrictions, data processing needs, and communication overhead. It is critical to manage and reduce these delays in networking applications in order to achieve real-time and responsive decision-making. In this regard, including these delays in simulations is critical in order to achieve a measurement of the actual impact of the computational delays on network performance when using RL-based RA algorithms and obtaining a realistic digital twin of the wireless network.

2.5 Simulation Tools and Framework

This work employs the ns3-gym framework, which combines the ns-3 network simulator and the OpenAI Gym environment. This section focuses on exploring the features of the ns-3 simulation framework and discussing the integration of ns3-gym tools.

2.5.1 ns-3

ns-3 [20] is an open-source discrete-event network simulator widely recognised in networking research. Developed from scratch in C++ using an object-oriented programming model, it has gained acceptance within the community for its reliable results [21]. Distributed under an open-source

and free software paradigm, ns-3 caters to the needs of the research and educational sectors. It fosters a collaborative model that encourages community-based contributions, module development, and validation activities across different groups.

At the very core of ns-3 is its event-based simulation methodology. The simulator processes events in the temporal order of simulation time and may generate new events as the simulation progresses. The simulation continues until there are no more events in the queue or a unique "Stop" event occurs [22]. This event-driven methodology is intended to closely resemble real-world circumstances.

ns-3 uses fundamental concepts and abstractions to simulate real-world computer and network systems. It functions as a container for applications, protocols, and network devices, simulating the tasks involved in constructing network systems, such as installing devices, connecting them to channels, configuring protocol stacks, and designating MAC addresses. These abstractions provide unified interfaces, allowing researchers to independently work on various protocol stack components while facilitating seamless integration.

The ns-3 community has developed a wide variety of networking protocols and communication technologies with comprehensive models of physical layer operations. Moreover, ns-3 provides statistical models for wireless channels, mobility, and the generation of traffic. In addition, it facilitates interaction with external systems, applications, and libraries, such as real-time testbeds.

ns-3 also includes a general tracing subsystem and an attribute configuration subsystem. The tracing subsystem monitors state changes in the network model, allowing observation of any entity's internal state and parameters during simulation. The attribute configuration subsystem, on the other hand, allows for runtime control of entity parameters and attributes, allowing for dynamic alterations. These subsystems serve as the foundation for the ns3-gym framework [23], which extends ns-3's capabilities.

2.5.2 ns-3 gym

OpenAI Gym [24] is an open-source toolkit designed for the development and comparison of RL algorithms. ns3-gym is a framework that leverages the powerful simulation capabilities of ns-3 with the flexibility of OpenAI Gym. It provides a bridge between the ns-3 network simulation environment and reinforcement learning agents, enabling researchers and developers to explore the application of reinforcement learning techniques in networking research.

The emergence of ns-3-gym framework was driven by the need to facilitate and accelerate the prototyping of RL-based networking solutions. It adheres to design principles such as scalability, low entry overhead, fast prototyping, and easy maintenance, as stated in [21]. The framework allows for running multiple ns-3 instances in a distributed environment, supporting both time and event-driven observation. It simplifies the conversion of legacy ns-3 simulation scripts to be used in the OpenAI Gym environment, reducing the entry barrier for researchers.

It provides a set of tools and APIs that allow users to define custom RL environments based on ns-3 network scenarios. These environments encapsulate the state, action, and reward spaces, providing an interface for reinforcement learning agents to interact with the simulated network.

With ns3-gym, researchers can design and train reinforcement learning agents to optimise network performance in various scenarios. For example, agents can learn to dynamically adjust network parameters, such as transmission power or routing decisions, to maximise throughput, minimise delay, or improve energy efficiency. This integration of reinforcement learning with ns-3 opens up new possibilities for addressing complex networking problems and exploring novel strategies for network optimisation.

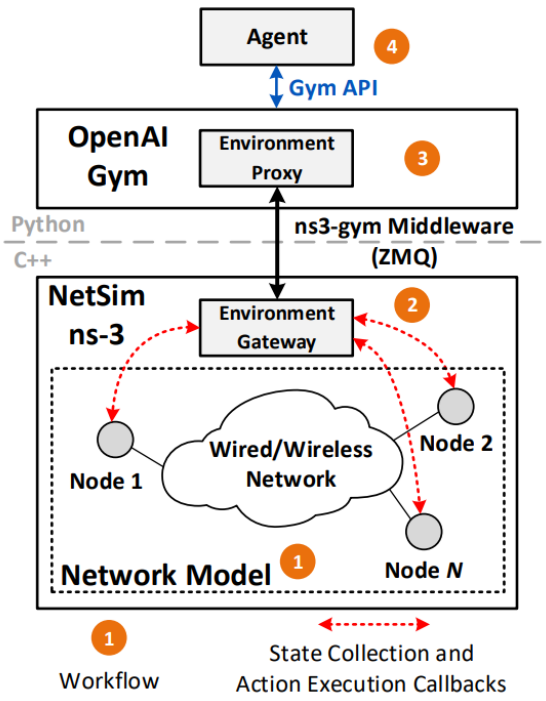


Figure 2.3: ns-3 Gym Framework. [21]

Moreover, ns3-gym provides seamless integration with popular Python frameworks such as TensorFlow and PyTorch. This allows researchers to leverage the extensive libraries and tools available in these frameworks for training and evaluating reinforcement learning agents within ns-3 simulations. The integration also facilitates easy experimentation and comparison of different reinforcement learning algorithms and techniques in the context of networking.

Overall, ns3-gym bridges the gap between ns-3 network simulations and reinforcement learning, offering a powerful framework for exploring and developing intelligent network control algorithms. By combining the strengths of ns-3 and OpenAI Gym, it enables the application of RL techniques to tackle complex networking challenges and advance the field of network optimization.

2.5.3 ns-3's Ideal Wi-Fi RA algorithm

The Ideal algorithm is a component of ns-3 that represents the optimal rate adaptation algorithm for Wi-Fi networks. It is implemented via the ns3::IdealWifiManager class [25] and provides a

streamlined and idealised rate adaptation algorithm. The Ideal RA algorithm keeps the SNR of each packet received in each station and sends this SNR back to the sender via an out-of-band mechanism. Each sender records the most recent SNR received from a receiver and uses it to select an MCS based on a set of SNR thresholds generated from a target Bit Error Ratio (BER) and MCS-specific SNR/BER curves. This simplified approach disregards practical limitations, such as implementation complexity and resource availability, and makes rate adaptation decisions centrally, resulting in all devices transmitting at the same rate.

It is important to note, while the ns3::IdealWifiManager class provides a valuable performance evaluation benchmark, it does not accurately represent the behaviour of actual rate adaptation algorithms. In order to obtain good performance in dynamic wireless environments, rate adaptation algorithms must contend with imperfect channel knowledge, feedback delays, implementation complexity, and various trade-offs. As a result, this ideal algorithm is beneficial for evaluating the upper limits of performance that can be achieved with rate adaptation, as it provides a point of comparison with other more realistic algorithms.

2.5.4 ns-3 Delay Integration

To the best of our knowledge, not a lot of research has been done regarding the inclusion of wireless nodes delays in network simulators, specifically ns-3. However, the author of the dissertation [26] provides some insight on how to model the processing times of Internet of Things (IoT) nodes in ns-3. While not directly focused on the execution times of RL-based RAAs, the work discussed provides valuable insights into how delays can be integrated in ns-3.

The author constructed a ns-3 module from scratch to simulate the processing delays for the integration of IoT node delays because it was a less invasive method than altering the already existing ns-3 files. Furthermore, the author determined that changing events after they occurred was impossible. As a result, the author chose to develop a new event that would add the delays while the packet was being tracked through the stack in order to incorporate the delay in the stack. Another option presented on [26] was to modify the scheduled events before they were created. The delays were examined and compared to a circumstance in which the developed ns-3 module was disabled. The results show that the end-to-end latency rose, indicating that the node processing delays were being accounted for.

2.6 Summary

The present chapter provided a brief introduction to the IEEE 802.11 set of technical standards, an overview of existing RAAs, a description of RL, coverage of the tools and frameworks to be used and a related work section, which included some insights on relevant approaches in the context of this dissertation.

Through the analysis of the state-of-the-art, it is possible to conclude that RL is a promising approach for dynamic rate adaptation, but more rigorous study is needed to allow the full deployment of this technology on experimental scenarios. In terms of computational delays, we learned

that additional research is required to evaluate the impact of delays and hardware capabilities on algorithm performance before these types of algorithms may be applied in experimental settings. Furthermore, we confirmed that the related research is quite limited and tends to focus on specific hardware profiles.

Chapter 3

DARA Algorithm and Delay Implementation

The core objective of this work is to comprehensively assess how computational delays affect DARA's effectiveness and efficiency, serving as a case study for similar state-of-the-art RL-based RA algorithms.

This chapter provides an in-depth examination of the proposed solution, as well as an overview of how DARA model operates. We examine the various steps involved in DARA's operation to build a picture of how the computational delays manifest throughout its execution. This chapter additionally comprises a section that explains how the integration of the delays into the ns-3 simulator was performed. Here, we outline the techniques employed to accurately integrate delays in the simulator, allowing us to conduct realistic simulations and obtain reliable results. Finally, this chapter concludes with a summary of DARA's DRL agent architecture and its key parameters.

3.1 Overview

To properly understand the different delay components, Figure 3.1 is provided as an overview of the DARA algorithm, highlighting its main operations in an experimental setting. The figure serves as a visual aid to comprehend the overall structure and functioning of the algorithm. The algorithm's operation can be divided into two main components: the agent side and the environment side.

On the agent side, two types of sessions are used: training sessions and evaluation sessions. The purpose of the training session is to search for and optimise the neural network's parameters. The evaluation session is then conducted to assess the effectiveness of the training session and measure the algorithm's overall performance. To summarise, during training, the agent updates its policy, i.e., learns from the feedback received from the environment. During evaluation, the agent decides on the appropriate action to take based on the current state, using the previously trained policy.

On the environment side, several operations are performed. These include action deployment, reward computation, and state querying. These operations are implemented through a function that is executed at each step of the algorithm. The action deployment refers to the selection and execution of the appropriate action based on the current state of the environment. The reward computation is the calculation of the reward value based on the action taken and the resulting state. Lastly, the state querying retrieves relevant information about the environment's current state to provide input to the agent for decision-making purposes.

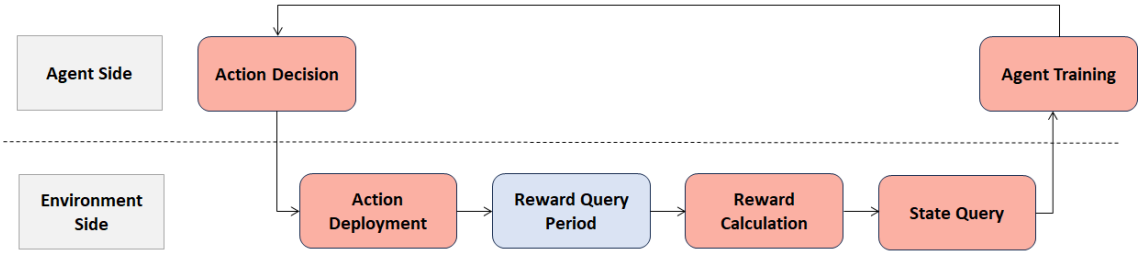


Figure 3.1: DARA Algorithm Loop[1].

The loop begins with an initial state query, as there is still no reward since the agent has yet to make any action decisions. The algorithm loop encompasses a sequence of steps in an experimental scenario:

1. The algorithm queries the state of the environment (i.e., the SNR).
2. The algorithm abandons the step function on the environment side. The agent is then trained and decides on an new action.
3. The action is converted into a rate;
4. The chosen action is deployed;
5. Query of reward statistics, including total number of frame successes and attempts;
6. Computation of the reward, based on the action taken and the resulting state;

It is important to highlight that in the simulation implementation of DARA, the system waits for the action decision from the agent. Consequently, during this processing time on the agent's side, there are no frame transmissions, no fluctuations in SNR, and the channel conditions, as well as everything else environment-related, remains constant. However, this condition does not hold in experimental scenarios, where the environment continues to evolve and update while the agent processes information. Consequently, frames are still being exchanged with outdated actions (i.e., as outdated MCS).

[1] Adapted from [7].

Computational delays are present in different parts of the DARA algorithm. The work presented in Chapter 2 concluded that there are five steps where delays occur in the DARA implementation – represented in red in Figure 3.1 – which are: the action deployment, the reward computation, the state query, the agent training and the agent decision.

The work in [7] provides exact information about the execution time of various operations in the original implementation of DARA, referred to as Baseline DARA. It also offers an analysis of the delays introduced by the updated version of DARA, known as E-DARA, which specifically aims to minimise the execution time of these operations. The delay characterization was made under an experimental setting on real wireless nodes from a testbed available through the Fed4FIRE+ project [19], specifically the w-iLab.2 testbed [27]. The results from the work in [7] are as presented in Table 3.1:

Table 3.1: Average Execution Times in E-DARA and Baseline DARA (in milliseconds).

	E-DARA	**Baseline DARA**
Environment Step	62.805	578.87
Action Deployment	15.105	———
Reward Calculation	0.246	———
State Query	0.299	———
Agent Training	21.606	20.670
Agent Decision	6.915	5.603

To accurately model delays within the simulation, it was essential to gain an understanding of how delays manifest during the execution of DARA in an experimental setting. Figure 3.2 is provided as a visual aid to illustrate the sequence of interactions among the agent, the OpenAI Gym, and the wireless device within an experimental context. Examining these exchanges enabled us to comprehend the nature of the problem and identify the points at which delays occur.

Firstly, when the chosen action is configured in the wireless device, a computational delay occurs between the configuration initiation at time t_1 and the actual implementation of the rate on the device at t_2. This delay arises due to the time required for the configuration to take effect.

Following the configuration, there is a predefined time period $[t_2, t_3]$ during which statistics are collected. This period, set to 100 ms in the simulation, allows for the gathering of pertinent data such as SNR values and successful/failed frame transmissions. These statistics represent a crucial sample as they reflect the outcomes of the newly implemented action, which will be used to determine the next action. Moreover, this time period presents a trade-off. If it is too short, the algorithm can quickly respond to abrupt changes in channel conditions, but it may result in insufficient data samples being collected to make an efficient decision. On the opposite, if the time period is too long, there will be a larger volume of quality data available, but the algorithm's responsiveness to sudden changes will be reduced.

Subsequently, the collected statistics are transmitted to the OpenAI Gym, where we compute the reward and state. This transmission occurs within the time interval $[t_3, t_4]$. The reward and state information are then relayed to the agent. Upon receiving the reward and state, the DRL

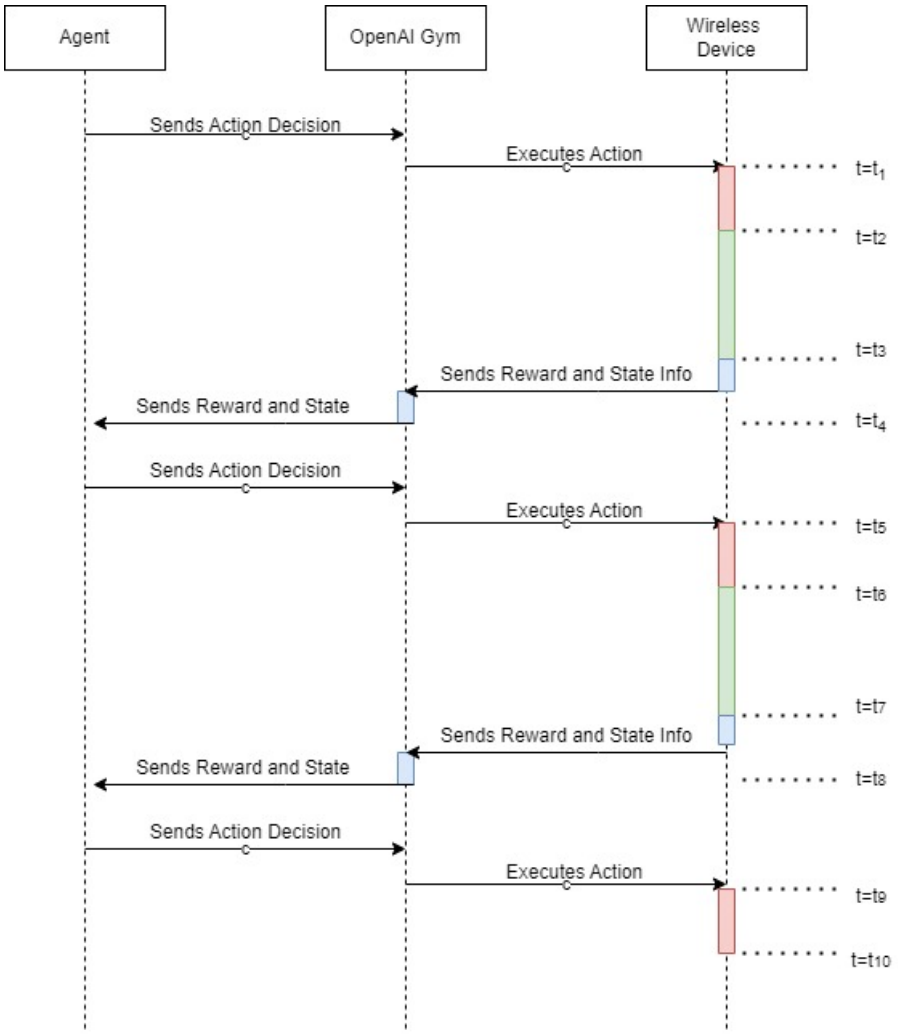


Figure 3.2: DARA Sequence Diagram.

agent proceeds with either training and decision-making or decides on an action using the current policy without training, depending on the specific session running.

Finally, the newly determined action, represented by the MCS value, is sent back to the OpenAI Gym. The Gym configures this action in the wireless device, resulting in its actual effect taking place at the designated time instant, t_{10}.

The current simulation work on DARA has been conducted under ideal conditions, where no delays are considered. In this ideal scenario, the simulator pauses, allowing the algorithm to evaluate the collected data and make a decision before resuming the simulation. Consequently, network traffic, channel conditions, node movement, and everything environment-related is temporarily halted while the algorithm performs its computations, and the newly decided action takes effect immediately after the statistics collection period t_3. However, real-life situations do not align with this ideal scenario, which means that there is transmission of outdated statistics to the agent. These outdated statistics, collected based on the preceding action, can impact the agent's

decision-making effectiveness and the overall performance of the algorithm.

To address this issue, we propose a solution that involves introducing delays in the simulator's action execution timeline. As such, we delay the execution of the action from its original time instant, right after the statistics collection period t_3, until the time when the decided rate actually takes effect in the wireless device. This is repeated every time an action is decided by the agent. By incorporating these delays, we can account for the computational delays observed in real-world scenarios and provide the agent with realistic, out-of-date statistics, thereby enhancing the simulation's realism.

The upcoming sections will provide a detailed exploration of the simulation techniques employed to implement our solution. Furthermore, they present the configuration parameters used in the simulation, offering a comprehensive understanding of the underlying methodology and settings used in our approach.

3.2 ns-3 Environment

One of the key components of this work is the ns-3 simulation environment, which is a general-purpose C++ program that implements all the simulation scenarios and configures the required parameters. The ns-3 version used was 3.38. Table 3.2 summarises the main configuration parameters for the conducted simulations.

Table 3.2: ns-3 Environment 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
Antenna Gain	0 dBi
Remote Station Manager	DARA/Ideal/Minstrel-HT
Traffic	UDP, generated above link capacity at 70 Mbit/s
Packet Size	1400 bytes
Frame Aggregation	Disabled during training

The decision to adopt the IEEE 802.11n Wi-Fi standard was made in order to align with the standard previously used in the original version of DARA. Moreover, the documentation for ns-3 is extensively consolidated for the 802.11n standard, enabling a more accurate representation of real-world scenarios. For the propagation delay model we used constant speed. As for the propagation loss model, we employed the Friis Free Space Propagation Model. This model accurately

represents the line-of-sight path loss in a free space environment without any intervening objects. Additionally, we used a Fixed RSS option, which allowed us to enforce specific SNR values or variations within the simulation to test specific network circumstances. For instance, by including abrupt SNR changes we could represent an example of temporary Line-of-Sight blockage.

User Datagram Protocol (UDP) was chosen instead of TCP for traffic generation purposes due to its simplicity and to avoid the impact of the TCP mechanisms for reliability and congestion control, that represents an additional variable that could affect the interpretation of the results. A constant data rate of 70 Mbit/s (above link capacity) is employed, along with fixed-size packets of 1400 bytes, to ensure link saturation, ensuring that packets are consistently and regularly transmitted, thereby maintaining a consistent load.

Moreover, frame aggregation was disabled during training due to ns-3 bugs that were identified in its current version, during the development of this work. As a result, the training process proceeded without frame aggregation to avoid any potential conflicts and ensure accurate and reliable results. Frame aggregation refers to the grouping of multiple smaller frames into a single larger frame for transmission over a wireless link. This technique helps to improve the efficiency of data transmission by reducing the overhead associated with transmitting and acknowledging individual smaller frames. By combining multiple frames into a single aggregated frame, the overall transmission time and overhead can be reduced, resulting in improved network performance.

This programme is also configured so that we can choose the simulation type as either a training or evaluation episode, based on its purpose. The simulation type specifies whether it is used to train the DRL agent or evaluate the performance of the selected algorithm. Additionally, we also define the duration of each episode, representing the length in simulation seconds. All the evaluation episodes were 30 seconds long.

All the simulations conducted for this work have used two network nodes, a transmitter (T_x) and a receiver (R_x). For testing and validation, three different scenarios were considered:

- **Abrupt SNR Oscillation Scenario**: this scenario forces an abrupt variation of the SNR value between 30 dB and 20 dB at regular 5 second intervals. This scenario aims to replicate real-world conditions where a physical object intermittently obstructs the transmission path between the transmitter (T_x) and receiver (R_x) nodes, resulting in a temporary sharp decrease in SNR. The objective here is to assess how the performance of the DARA algorithm is affected by computational delays, when it comes to dynamically adjusting the MCS in response to these SNR variations.

- **Moveaway Scenario**: this scenario aims to simulate the progressive movement of R_x away from the T_x node, resulting in a gradual decrease in the SNR at the R_x node. This scenario is particularly valuable for training the agent as it provides a comprehensive range of SNR values, requiring the agent to go through all the levels of MCSs available, learning the limits of each index and the correct action, that maximises the throughput for each observation.

- **Teleporting Node Scenario**: In this scenario, we explore a simulation environment where R_x teleports to a new random position every two seconds within a range of zero to 600 meters from T_x. Although teletransportation is not a realistic scenario, it serves the purpose of testing the performance of the DRL agent in the presence of abrupt SNR changes, resembling extreme conditions like in airborne or underwater networks. By inducing sudden and extreme variations in the distance between the nodes, the scenario effectively generates the desired SNR variations. Additionally, this scenario provides a valuable test case for evaluating DARA, as it requires the algorithm to adapt to a highly dynamic environment. The challenges are further intensified when computational delays are introduced, making it an ideal setting to assess the DARA system's performance under realistic conditions.

3.3 ns-3 gym

The ns-3 Gym serves as an interface between ns-3 and the OpenAI Gym framework, facilitating the integration of reinforcement learning techniques into network simulation environments. The gym program defines the observation and action spaces. Moreover, it collects the current network state, calculates observations and rewards based on the gathered information, and transmits them to the DRL agent. The agent, in turn, leverages this information to select the most appropriate action for the upcoming step, relaying it back to the ns-3 gym. The gym program then efficiently translates these chosen actions into practical implementations within the ns-3 framework.

This collaborative process ensures that the simulation progresses smoothly, incorporating delays realistically, and providing a suitable environment for comprehensive analysis. In this sense, the gym program is the most relevant part of the implementation for this work, as it is where the delays are integrated into the simulation.

The original implementation of DARA in the ns-3 simulator comprised the three main functions presented in Listing 3.3. The first function is responsible for collecting the observation value and scaling it to a range between 0 and 1. The second function determines the reward based on the achieved average data rate and the success rate of attempts. Lastly, the third function implements the chosen action by applying the determined MCS value and the corresponding data rate in the simulation.

```
1. Ptr<OpenGymDataContainer> GetObservation();
2. float GetReward();
3. bool ExecuteActions(Ptr<OpenGymDataContainer> action);
```

Listing 3.1: Gym Main Functions.

To incorporate the delays into the simulation effectively and achieve a high degree of realism, it becomes necessary to introduce a temporal latency in the execution of actions that aligns with the specific delay period. However, in the existing setup of simulation programs, we lacked control over the timing of the function *ExecuteActions(action)* called by the DRL agent. Consequently, delaying its execution within the simulation became a challenge.

To address this issue, a pragmatic approach was to create a new function called *doExecuteAction* outside the knowledge of the agent. This new function mirrors the structure and properties of the original function and is responsible for configuring the action in the simulator. Subsequently, the *executeActions* function was modified to only call the newly created function. By leveraging the ns-3 Simulator::Schedule() function, we gain control over when the created function is invoked, enabling the introduction of delay values and the faithful simulation of real-world conditions.

```
1  bool
2  MyGymEnv::ExecuteActions (Ptr<OpenGymDataContainer> action)
3  {
4    Simulator::Schedule(Seconds(delay), &MyGymEnv::doExecuteActions,this,action);
5    return true;
6  }
```

Listing 3.2: MyGymEnv::doExecuteAction - Static Delay.

As shown in Listing 3.2, the updated method calls the ns-3 *Simulator::Schedule* function, which accepts the action and delay as arguments. Here, the action parameter represents the MCS value selected by the DARA agent, while the delay denotes the duration for which we schedule the execution of *doExecuteAction*. By employing this approach, we can apply the chosen rate in the simulation and replicate real-world scenarios more accurately

Regarding the delays, we employed two distinct approaches to ensure flexibility and realism in modelling the delays within the simulation environment. The first approach was static, meaning that the input provided to the scheduler function remained constant throughout the whole simulation period, and for every considered episode. In contrast, the second approach was non-static, involving the random generation of delay values based on a normal distribution derived from a large data set of delay samples, previously collected in [7].

Figure 3.3 displays the distribution plots of delay values for various operations incorporated in the DARA implementation. The delay values are represented in blue, while the corresponding normal distribution graph is depicted in red. This non-static approach aimed to create a more realistic simulation by accounting for the inherent variability in execution times across different episodes. By incorporating this variation, the simulation can better reflect real-world scenarios where execution times may differ.

```
7  Ptr<NormalRandomVariable> rng_var = CreateObject<NormalRandomVariable> ();
8    rng_var->SetAttribute("Mean", DoubleValue(0.245));
9    rng_var->SetAttribute("Variance", DoubleValue(0.00735));
10
11   for (int i = 0; i < 300; ++i) {
12     reward_delay[i] = rng_var->GetValue();
13   }
```

Listing 3.3: Random Delay Value Generator.

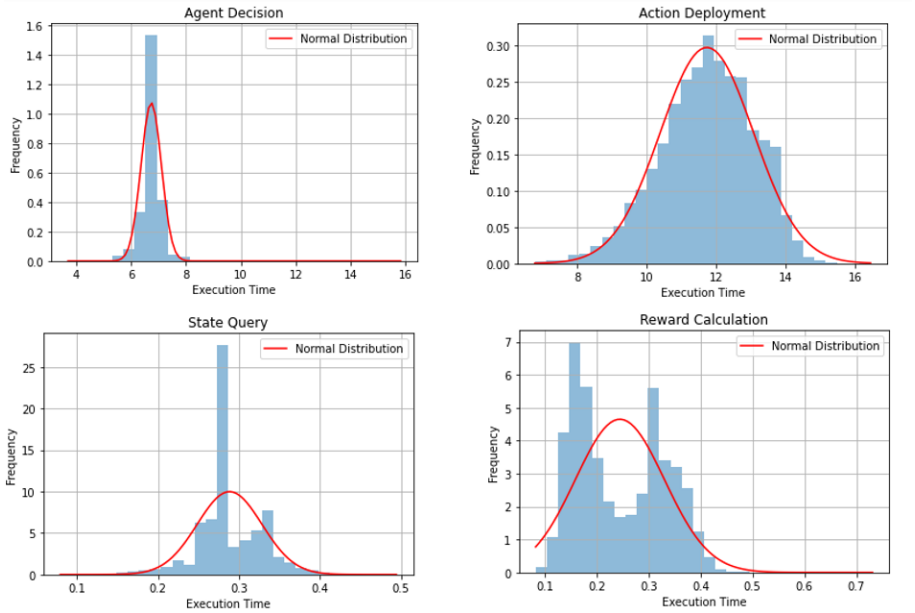


Figure 3.3: Normal Distribution of Delays.

Listing 3.3 presents the code for generating the delays, using the example of the reward computation delay. The code begins by creating as many values as the total number of steps in the simulation, with each step representing 0.1 seconds in simulation time. All these values are generated at before the beginning of the simulation and stored in an array. Then, when scheduling the execution of an action, a new value from the array is used as the delay. This ensures that each action execution has a different delay based on the corresponding value from the pre-generated array and that the generation is not affected by events that can occur during the simulation.

Both approaches were tested, with the static approach being initially employed due to the unavailability of the delay samples captured in [7] during the initial development phase. Furthermore, the static approach also served the purpose of introducing arbitrary delay values to simulate different scenarios that could be encountered on different hardware configurations.

3.4 Agent Architecture and Training

The DARA agent is implemented in Python and uses the TF-Agents library. The agent's role is to receive an observation and a reward from the environment of the previous time interval, and select the action for the next time interval. The DARA agent uses a Deep Q-Network (DQN) learning algorithm with the following parameters:

- **Observation Space**: one-dimensional float ranging between 0.0 and 1.0. The SNR value is scaled by a factor of 100, as unscaled input variables can result in a slow or unstable learning process;

- **Action Space**: one-dimensional integer (0-7), representing the eight possible MCS indexes for packet transmission;
- **Optimiser and Learning Rate** (α): Adam Optimiser [28], which is an algorithm for first-order gradient-based optimisation of stochastic objective functions, based on adaptive estimates of lower-order moments. The optimiser requires the definition of the learning rate, which aims at scaling the magnitude of weight updates in order to minimise the network loss function. The value chosen for this parameter was 10^{-2}.
- **Epsilon Greedy** (ε): plays a vital role in the decision-making process of the agent. Initially set to 100%, the ε governs the proportion of random actions taken by the agent. Throughout the epsilon decay period, this factor gradually decreases until reaching the final value of 10%. This deliberate reduction ensures comprehensive exploration of all possible actions, allowing the agent to discover optimal strategies. Simultaneously, the exploration-exploitation trade-off comes into play, with an initial setting of 100%, indicating that every decision is exploratory. As the agent gains knowledge about the environment, this value diminishes over time, enabling the agent to exploit its acquired knowledge in later stages of training.
- **Replay Buffer**: this parameter serves as a memory structure that stores the agent's experiences during its interaction with the environment. During the simulation, the replay buffer accumulates trajectories, which consist of the initial observation, the corresponding action taken, and the subsequent observation and reward. As training progresses, the agent randomly selects a batch of 64 trajectories from the replay buffer for learning purposes. This sampling allows the agent to learn from a diverse set of experiences stored in the replay buffer. The Replay Buffer size allowed the storage of up to 10^6 trajectories.
- **Neural Network Architecture**: refers to the Deep Q-Network architecture. We used two fully connected layers with 32 units each.
- **Number of Training Episodes**: training episodes refer to the count of instances where the environment is reset to its initial configuration, triggered by the game over rule. Each training episode encompasses a series of actions performed by the agent, observations obtained from the environment, and corresponding rewards or penalties based on the agent's actions.

In terms of the training methodology, we first train the agent without taking into account any delays. This initial training phase allows us to assess the performance when a correct policy is implemented, serving as a baseline for comparison. Following this, we incorporate delays into the training process to evaluate their impact on both the learning process and the resulting performance. This approach enables us to analyse the effects of delays on the agent's training dynamics and overall performance outcome.

Chapter 4

Performance Evaluation

This chapter presents the simulation results obtained from evaluating the DARA algorithm, taking into account the delays modelled within the simulation. The evaluation of DARA was conducted across three distinct scenarios, as detailed in Section 3.2. In each scenario, we analyse the performance of DARA under varying computational delays and compare it to the Ideal and Minstrel-HT algorithms. We also use the original simulated version of DARA, which assumes no delays, as a reference for comparison. The primary performance metric we evaluate is network throughput, as DARA's design was oriented towards throughput maximisation. We additionally present frame loss results to provide a broader analysis of the algorithm's performance. This chapter ends with a summary of the main conclusions drawn from the obtained results.

Throughout this chapter, we will refer to the delay values using specific terminology based on their respective experimental implementations. For instance, the term "E-DARA delays" pertains to the delay configuration in the final implementation of DARA, which incorporates improvements introduced by previous work developed in [7]. Moreover, the term "baseline DARA" refers to the original execution times without any improvements made by the author. The average delay for E-DARA when considering agent training is 37.25 ms, and 22.57 ms when the agent makes decisions based on a pre-trained policy, i.e., without training. Regarding the baseline DARA version, the average delay is 549.54 ms when considering agent training and 534.47 ms when the agent decides based on a trained policy, i.e., without training. The results regarding the E-DARA and the baseline DARA delays were obtained using the non-static implementation, i.e., the delay values were generated based on the normal distribution of their samples.

4.1 Abrupt SNR Variation Scenario

This was the scenario selected as the preliminary test case due to its simplicity, as it involves only two different MCSs and a fixed variation regarding the channel conditions. Initially, the training approach aimed to expose the agent to a wide range of SNR values by using scenarios that encompassed all available MCSs. However, the presence of ns-3 bugs regarding the frame aggregation, as discussed in the previous chapter, complicated the training process, particularly when dealing

with broader and more complex scenarios. Consequently, to address these challenges, the decision was made to simultaneously utilize this particular static scenario for both training and evaluation purposes.

Figure 4.1 displays the throughput achieved for each MCS across the entire range of SNR values. In this particular scenario, the environment's state alternates between 30 dB and 20 dB, indicating that the corresponding MCS values are 7 and 4, respectively. Initially, the training process for this scenario started without considering any computational delays, and employed 30-second episodes for both training and evaluation purposes.

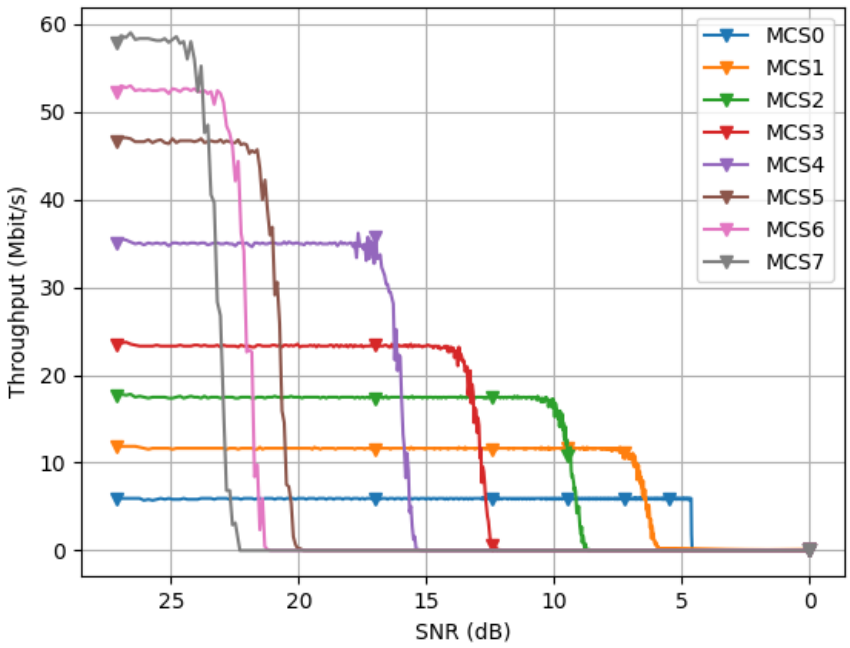


Figure 4.1: Throughput per MCS considering the whole range of available SNR values. [29]

We initially compared Ideal and Minstrel-HT against the original implementation of DARA, which did not take into account any computational delays. This allowed us to measure DARA's performance under ideal processing conditions for future comparison. The plots are shown in Figure 4.2. We observed that, while DARA matched ns-3's Ideal in terms of overall performance, it still experienced significant drops in throughput whenever the SNR decreased and a lower MCS index was configured. Despite this drawback, DARA still achieved an overall throughput that was 17.85% higher than Minstrel, and only 1.52% lower than Ideal. To note that all percentage comparisons mentioned in this document will be based on Equation 4.1. In this equation, the term 'Metric' represents either Throughput or Frame Loss, and RAA_1 and RAA_2 refer to the rate adaptation algorithms being considered.

$$\%_{difference} = \frac{Metric_{RAA_1} - Metric_{RAA_2}}{|Metric_{RAA_2}|} \times 100 \tag{4.1}$$

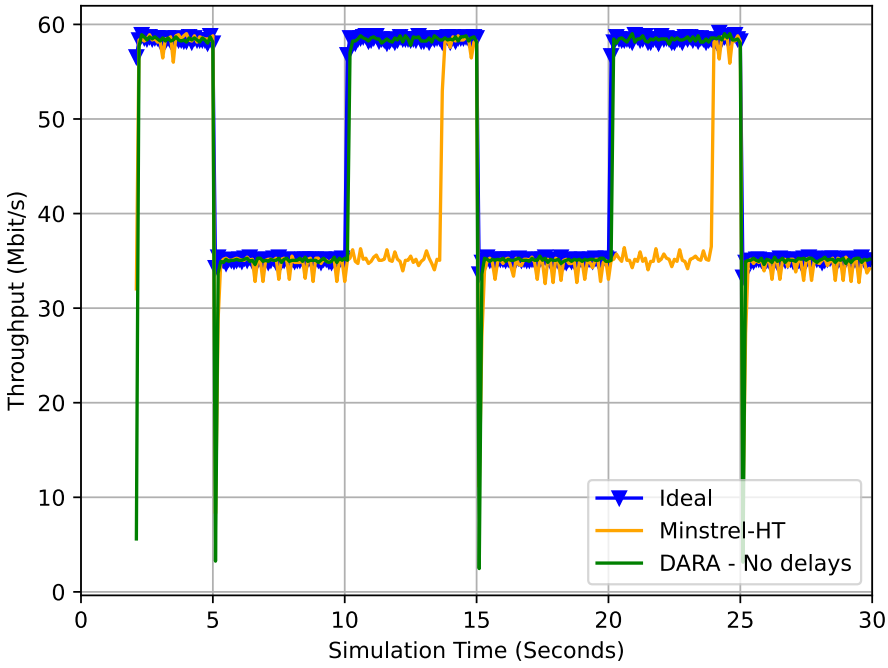


Figure 4.2: Throughput comparison of Ideal, Minstrel-HT and DARA with no delays - Abrupt SNR Variation Scenario.

The drops in throughput result from an already known desynchronisation issue in DARA. This desynchronisation occurs when the SNR changes before an action is executed, resulting in the selection of an inappropriate MCS for the current channel conditions. This is due to relying on the observed SNR from the previous interval. Additionally, when no acknowledgments (ACKs) are received, the average SNR cannot be calculated, and a value of 0 is assumed. The objective of this measure was to try to recover link information, i.e. SNR, at the cost of a minimal throughput. If the next decision has a valid SNR, the problem is corrected, but it causes suboptimal action decisions in those previous cases. In deteriorating link quality scenarios, the same desynchronisation issue occurs. However, since no packet loss is observed, the SNR is calculated based on the average of the received packets. Consequently, the agent adjusts to the correct MCS within a single decision interval, minimising the significance of this problem.

The issue becomes more pronounced when computational delays are introduced into the simulation, as shown by the plots in Figure 4.3 and the results in Table 4.1. In fact, as the total value of delays increases, the drops in throughput become more prolonged. The introduction of a temporal lag in executing the action leads to a situation where the action encounters significantly different channel conditions when it takes effect. Consequently, the selected action is no longer suitable, resulting in a visible decrease in throughput. This can be better observed in the close-up of the throughput variations detailed in the Figures 4.4a and 4.4b. It is important to note that the initial seconds of the simulation should not be considered for analysis as they represent the "Warm-up period". This period is necessary for setting up the simulation and should be excluded from obser-

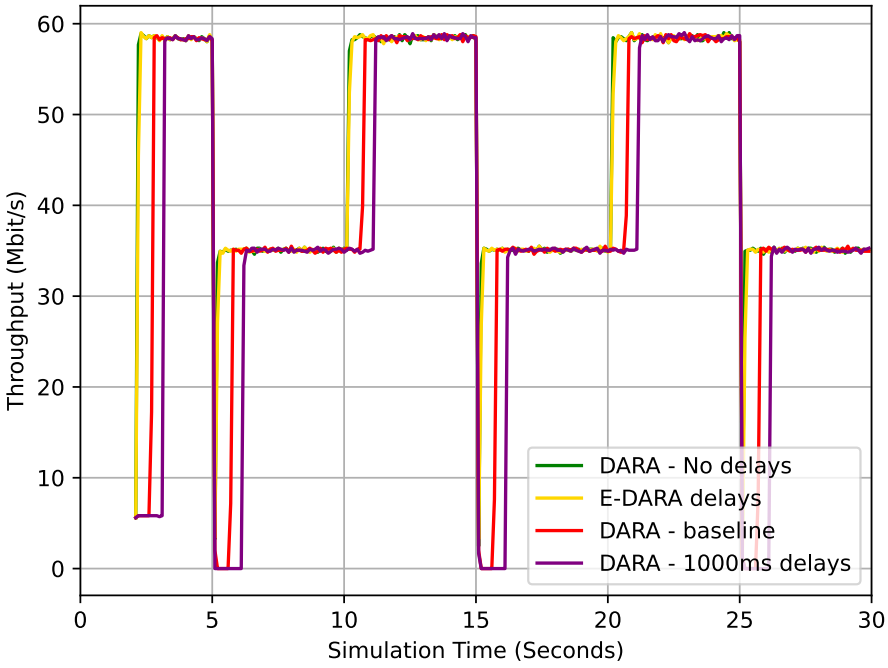


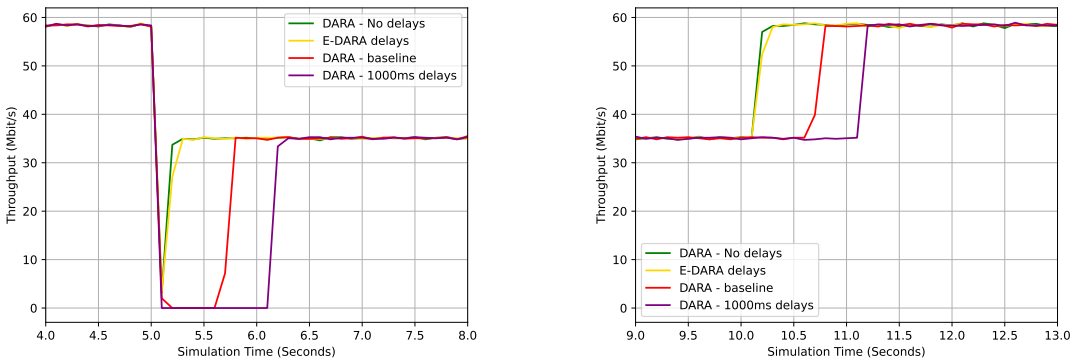
Figure 4.3: DARA under varying delays - Abrupt SNR variation Scenario.

Table 4.1: Throughput and Frame Loss Comparison - Abrupt SNR variation Scenario

	Abrupt SNR Variation Scenario	
	Throughput (Mbit/s)	Total Frame Loss
Ideal	45.93	————————
Minstrel-HT	38.97	————————
DARA - no delays	45.24	1410
E-DARA	45.07	1740
Baseline DARA	40.98	9810
DARA - 1000ms delay	37.88	16020

vations. However, it is interesting to observe that the introduction of delays within the simulation increases this initial time period. This is because the first action taken by the agent is delayed due to the delays introduced.

The impact of computational delays is less significant in the E-DARA version of the algorithm, as it was specifically designed to reduce execution times. With a total delay of approximately 22 ms, the average throughput only experiences a marginal 0.38% decrease compared to the original DARA implementation without any delays. The baseline DARA version, which introduced a total delay of approximately 530ms, shows a more substantial throughput decrease of 9.37%, when compared to DARA with no delays. Furthermore, when subjected to a longer arbitrary delay of 1000 ms, the decrease in throughput reaches 16.27% compared to the DARA's implementation with no delays. Overall, the E-DARA version exhibits the least impact in throughput variation



(a) Time instant in which the SNR decreases to 20 dB from 30 dB.

(b) Time instant in which the SNR increases to 30 dB from 20 dB.

Figure 4.4: Close-up of Throughput Variation.

due to its optimised execution times. In contrast, both the baseline DARA and the version with longer delays experience more significant decreases in throughput compared to the ideal DARA implementation.

Regarding the frame loss, our results indicate that, in the case of E-DARA, frame loss increases by a factor of 23.43%. Moreover, for the baseline version, it rises by 595%, and for the 1000 ms delay version, it surges by 1036%. In this scenario, where link quality deteriorates abruptly, an inappropriate MCS is selected for the current channel conditions, due to the desynchronisation problem explained before. This selection leads to a temporary drop in throughput to zero or near-zero. Although the throughput eventually recovers to the ideal value, there is a time interval during which packet loss occurs, as no data is flowing through the network. As a consequence, the inclusion of delays leads to a notable increase in packet loss, despite the relatively lower impact on throughput.

4.2 Moveaway Scenario

For the Moveaway scenario, we conducted 100 training episodes. Moreover, to ensure that the agent had sufficient exposure to different SNR levels and learned all the correct MCSs accurately, we selected a training duration of 60 seconds. This longer duration facilitated a slower decrease in SNR over time. During the evaluation phase, we employed 30-second-long episodes to maintain consistency with the testing duration used in the other scenarios.

Once again we do a preliminary study where we evaluate the DARA version with no delays against ns-3's Ideal and Minstrel-HT. The obtained plots are presented in 4.5 and the results regarding average throughput and total frame loss are presented in Table 4.2. The results show that DARA is able to learn all the correct MCSs and achieve a average throughput higher than the Ideal and Minstrel. DARA achieves 1.86% higher throughput than Ideal and 9.6% higher than

Minstrel-HT. The fact that Ideal is not oriented towards throughput maximisation explains why DARA, which is oriented towards throughput maximisation, performs better overall.

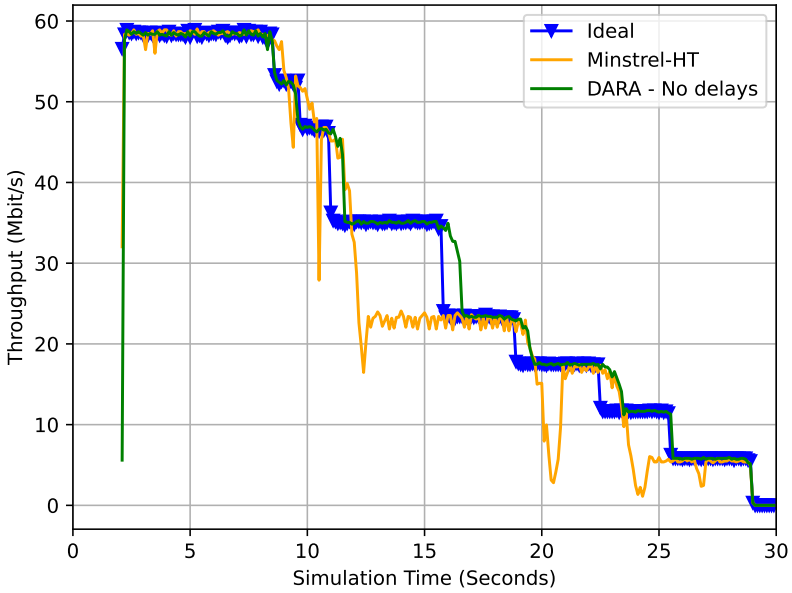


Figure 4.5: Throughput comparison of Ideal, Minstrel-HT and DARA with no delays - Moveaway Scenario.

Table 4.2: Throughput and Frame Loss Comparison - Moveaway Scenario.

	Moveaway Scenario	
	Throughput (Mbit/s)	Total Frame Loss
Ideal	30.71	————
Minstrel-HT	28.54	————
DARA - no delays	31.28	839
E-DARA	31.25	908
Baseline DARA	30.29	3285
DARA - 1000ms delay	28.55	7435

When simulating with E-DARA delays, the impact on overall average throughput is basically non-existent, with only 0.01% decrease compared with the zero delay version. As for the baseline DARA delays, throughput decreases to 30.29 Mbit/s, about 3.16%. And for the arbitrary 1000 ms total delay version, the overall throughput decreases to 28.55 Mbit/s, a 8.72% drop.

A distinct drop in throughput can be observed at MCS transition points for both the baseline and 1000 ms delay versions, a phenomenon that is not yet evident in the E-DARA version. This observation is likely attributed to the previously mentioned desynchronisation issue between action decision and execution. However, the impact of this issue remains insignificant for cases with lower computational delays. It becomes increasingly pronounced as the total delay value rises,

which may even exceed 1000 ms in scenarios involving hardware-limited experimental wireless nodes.

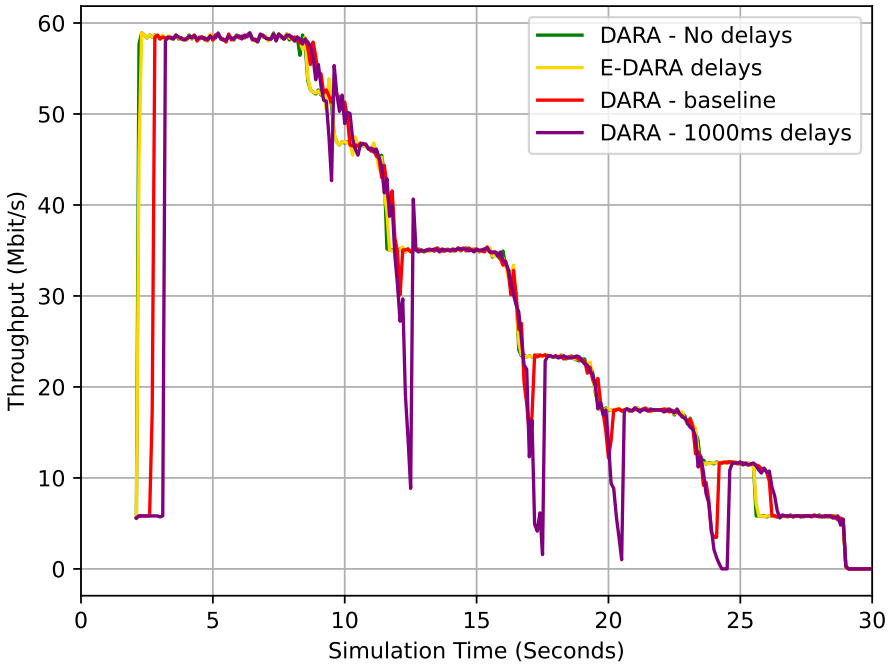


Figure 4.6: DARA under varying delays - Moveaway Scenario.

In terms of total frame loss, the results obtained demonstrate an increase of 8.22% for the E-DARA version, 291% for the baseline version, and 786% for the 1000 ms delay version compared to the original DARA version. It is important to note that this comparison is made in a scenario without abrupt changes in link quality; instead, the link quality gradually deteriorates. As a result, the observed increase in packet loss is less drastic compared to the previous scenario, mainly because the drop in throughput is not as significant or prolonged.

To assess the influence of delays on the training process, we conducted the same set of 100 episodes we did previously, this time incorporating the delays into the training process. The resulting throughput plots are depicted in Figures 4.7 and 4.8.

Table 4.3: Throughput Comparison between using Training with and without Delays.

	Throughput (Mbit/s)	
	Training without Delays	Training with Delays
E-DARA	31.25	27.02
Baseline DARA	30.29	13.49

The observed results indicate a reduced learning efficiency of the algorithm when delays are introduced, even with small delay amounts such as those in E-DARA. This is because the introduction of delays causes desynchronisation between action decision and action execution, leading

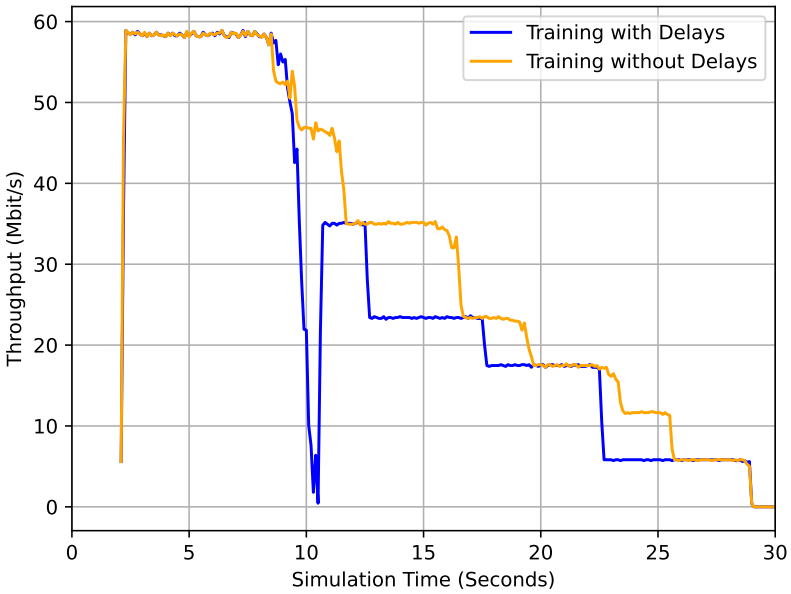


Figure 4.7: Throughput Comparison: training with and without E-DARA delays.

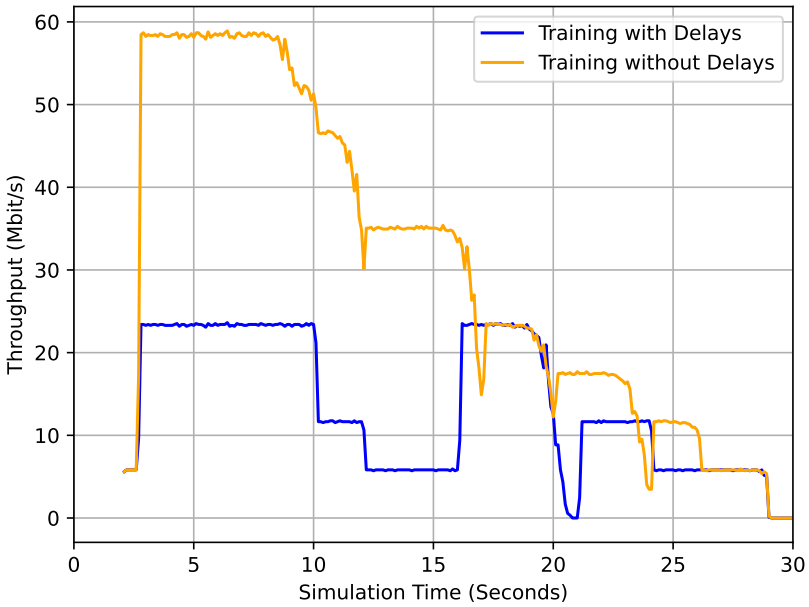


Figure 4.8: Throughput Comparison: training with and without baseline DARA delays.

to a less efficient training process. As shown in Table 4.3, when we run a evaluation session, the average throughput of the simulation decreased as inadequate MCSs were chosen by the agent. For the E-DARA version, the average throughput decrease by a factor of 13.54%. As for the baseline DARA version, throughput fell by about 55.4%. From these results, we can conclude that a higher number of training episodes might be required for the agent to effectively adapt to the

practical limitations imposed by delays. Moreover, it is possible that achieving an identical performance to the original DARA version could be unattainable when confronted with substantial delays. However, this particular aspect was not addressed in this work.

4.3 Teleporting Node Scenario

As explained in Chapter 3, in this scenario we assign a new random position to node R_x, between zero and 600 meters, every two seconds, and maintain the T_x node at a fixed position. For this scenario, no new training was necessary due to the extensive training the agent underwent in the previous moveaway scenario. During that training, the agent was exposed to a wide range of SNR values, which equipped it with valuable knowledge that can effectively be applied in all scenarios, as long as the other link condititons remain the same. Figure 4.9 contains the plots of the throughputs obtained by the Minstrel-HT, Ideal and DARA's version with no delays. These preliminary results indicate that the original version of DARA reacts similarly to Ideal and significantly better than Minstrel-HT, that fails to adapt to the optimal MCS between t $\approx$ 15s and t $\approx$ 25s, resulting in a lower throughput.

Table 4.4: Throughput and Frame Loss Comparison - Teleporting Node Scenario.

	Teleporting Node Scenario	
	Throughput (Mbit/s)	Total Frame Loss
Ideal	27.47	————————
Minstrel-HT	21.59	————————
DARA - no delays	26.82	1442
E-DARA	26.71	1570
Baseline DARA	23.94	5678
DARA - 1000ms delay	21.81	8774

When conducting a realistic simulation with delays, it is observed that E-DARA's delays do not significantly impact the throughput, resulting in only a 0.41% decrease compared to the version without delays. However, the baseline version experiences a more substantial drop of 10.71% in throughput. In the case of introducing an arbitrary delay of 1000 ms, the drop in throughput amounts to approximately 18.68%. Since this simulation represents a dynamic scenario where channel conditions undergo abrupt variations, we observe the same desynchronisation issue detailed in Section 4.1. Consequently, the results obtained highlight that computational delays exacerbate this issue, particularly for the baseline version and delays of 1000 ms, which are considerably higher than E-DARA's. It is observed that when link quality deteriorates rapidly, the observed throughput of DARA temporarily drops to zero. The amount of time the throughput remains low is also exacerbated by higher delay values.

The obtained results reveal a notable increase in total frame loss for different versions: 8.9% for the E-DARA version, 293% for the baseline version, and 508% for the 1000 ms delay version when compared to the original DARA version. This aligns with the first scenario where there is

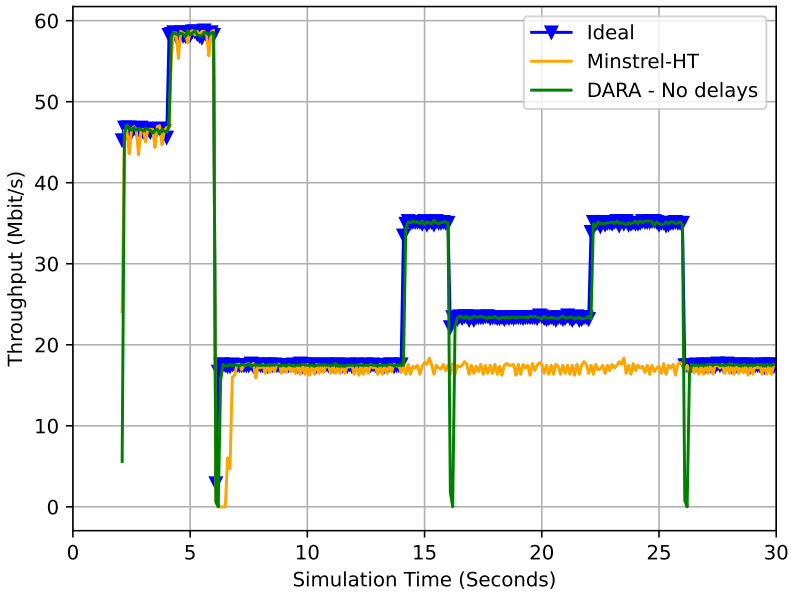


Figure 4.9: Throughput comparison of Ideal, Minstrel-HT and DARA with no delays - Teleporting Node Scenario.

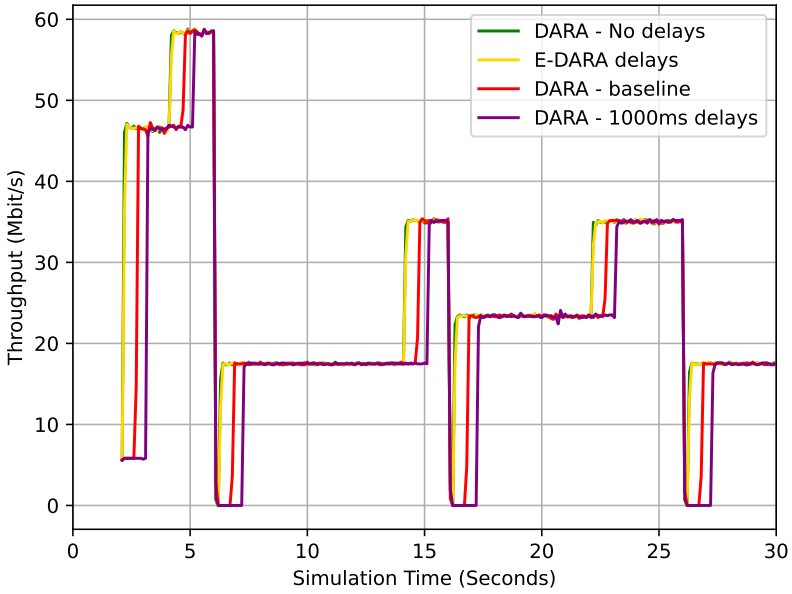


Figure 4.10: DARA under varying delays - Teleporting Node Scenario.

an abrupt drop in SNR, resulting in a higher packet loss during periods of zero or extremely low throughput.

4.4 Summary

The results presented in this chapter offer valuable insights into the impact of computational delays on the performance of RL-based rate adaptation RA algorithms. The evaluation of DARA reveals that even when simulating optimized experimental implementations, a noticeable effect on throughput and frame loss happens due to the computational delays. However, these effects are not significant enough to make the algorithm impractical when the agent already possesses a good policy, i.e., when it has previously learned the appropriate MCS for each observation interval.

Even when simulating DARA with an arbitrary and potentially unrealistic total execution time of 1000 ms, the overall throughput for all three scenarios simulated remains comparable to Minstrel's performance. Since Minstrel is the default RA algorithm used in the Linux Kernel, this indicates that even with more constrained hardware, an RL-based approach to RA can still prove highly suitable, presenting a promising solution to the ongoing challenges of RA.

Regarding the training process, the preliminary results obtained in this study demonstrate that considering computational delays affects the efficacy of the training process. The desynchronisation between action execution and decision-making, coupled with the unpredictability and variability of real-world wireless networks, poses a challenge to learning, which could potentially impact the prospects of online learning in the future.

Chapter 5

Conclusion and Future Work

RL-based algorithms have shown promising results in RA but often neglect computational delays and simplify real-world network environments, during early development stages. As such, simulated scenarios lack accuracy in capturing network instability and unpredictability. Prior research highlighted the execution time problem and the significance of computational delays in RL-based RA algorithms. However, their impact on algorithm performance has been overlooked. Therefore, the motivation for this work is addressing this gap by investigating the effects of delays on RL-based RA algorithms, while simultaneously making the simulation environment as realistic as possible.

In this context, our solution involved the replication of real-world scenarios in ns-3, where DARA would be subject to hardware limitations and dynamic network environments. To validate our approach, we conducted evaluations of DARA in three distinct simulation scenarios. These scenarios involved SNR oscillation between two constant values, a moveaway scenario, and a node teletransportation to a random position. By simulating different experimental implementations, we observed the variations in throughput and packet loss when incorporating varying computational delays in the simulator. We compared these results to Ideal and Minstrel-HT RA algorithms, and also the original version of DARA, evaluated in simulation without taking into account any sort of computational delays.

The results indicate that even with optimised experimental implementations, noticeable effects on throughput and frame loss occur due to computational delays. However, these effects do not invalidate the algorithm's experimental deployment, since that even for exaggerated delays, DARA still achieves an average throughput similar to Minstrel-HT, one of the most prominent RA algorithms used today. This suggests that RL-based RA remains a promising solution despite hardware limitations. Regarding training, preliminary results indicate that considering computational delays can affect the overall efficacy of the training process.

In this work, we assessed DARA by incorporating delays observed in real experimental scenarios with actual wireless nodes, as well as arbitrary values that could represent more limited hardware conditions. Given the significant variation in execution times based on hardware profiles, future work could focus on developing a benchmarking tool to provide delay statistics specific to

the hardware on which the algorithm is deployed. Such a tool would enhance the deployment process in ns-3 and expedite the acquisition of relevant results.

To enhance the current work, testing DARA or other state-of-the-art RL-based RA algorithms, in more complex scenarios involving a higher number of users and diverse mobility scenarios would provide a more comprehensive evaluation of its performance. Additionally, analysing additional performance metrics, such as network jitter, could offer new insights into the impact of delays on the network's packet transmission. Moreover, to better simulate real-world environments, incorporating asymmetric links, which are commonly encountered in practical scenarios, would be beneficial. Analysing the behaviour of DARA under TCP traffic can also be relevant since, because of reliability control, this could result in a more pronounced throughput drop but a less significant increase in packet loss. Furthermore, a more comprehensive analysis on the impact of delays in the training process would also be relevant as future work. By addressing these aspects, a more thorough understanding of DARA's and other RAAs capabilities and limitations can be attained.

In conclusion, this solution presents a valuable contribution by introducing a method to incorporate computational delays into ns-3, thereby enhancing the realism of simulation environments for evaluating RA algorithms. This implementation addresses the existing limitation in ns-3, which lacks mechanisms for considering processing time. As such, this positions ns-3 as a digital twin of the physical world, enabling a more precise replication of real-world scenarios and realistic prediction of experimental outcomes. Furthermore, the provided results on the performance of RL-based algorithms, accounting for computational delays, offer insights that can be valuable not only for RL-based rate adaptation algorithms, but also for other RL or ML-based algorithms, outside the scope of RA. By bridging the gap between simulated scenarios and real-world complexities, this work facilitates more accurate evaluations and fosters advancements in the field of RL for wireless networks.

References

[1] Rúben Queirós, Eduardo Nuno Almeida, Helder Fontes, José Ruela, and Rui Campos. Wi-fi Rate Adaptation using a Simple Deep Reinforcement Learning Approach. In *2022 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–3, 2022. doi:10.1109/ISCC55528.2022.9912784.

[2] Syuan-Cheng Chen, Chi-Yu Li, and Chui-Hao Chiu. An Experience Driven Design for IEEE 802.11ac Rate Adaptation based on Reinforcement Learning. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pages 1–10, 2021. doi:10.1109/INFOCOM42981.2021.9488876.

[3] Saad Biaz and Shaoen Wu. Rate adaptation algorithms for ieee 802.11 networks: A survey and comparison. In *2008 IEEE Symposium on Computers and Communications*, pages 130–136, 2008. doi:10.1109/ISCC.2008.4625680.

[4] Dong Xia, Jonathan Hart, and Qiang Fu. Evaluation of the Minstrel rate adaptation algorithm in ieee 802.11g wlans. In *2013 IEEE International Conference on Communications (ICC)*, pages 2223–2228, 2013. doi:10.1109/ICC.2013.6654858.

[5] iwlwifi [linux wireless]. URL: https://wireless.wiki.kernel.org/en/users/drivers/iwlwifi.

[6] Ricardo Trancoso, Ruben Queiros, Helder Fontes, and Rui Campos. On the analysis of computational delays in reinforcement learning-based rate adaptation algorithms, 2023. arXiv:2303.17477.

[7] Ricardo Trancoso. Analysis and Optimisation of Computational Delays in Reinforcement Learning-based Wi-Fi Rate Adaptation. Master's thesis, Faculdade de Engenharia, Universidade do Porto, 2022. URL: https://hdl.handle.net/10216/144595.

[8] Cees Links. The evolution of Wi-Fi Networks: From ieee 802.11 to wi-fi 6e. URL: https://www.wevolver.com/article/the-evolution-of-wi-fi-networks-from-ieee-80211-to-wi-fi-6e.

[9] Evgeny Khorov, Ilya Levitsky, and Ian F. Akyildiz. Current status and directions of ieee 802.11be, the future wi-fi 7. *IEEE Access*, 8:88664–88688, 2020. doi:10.1109/ACCESS.2020.2993448.

[10] Rohan Patidar, Sumit Roy, and Thomas R. Henderson. Technical report on validation of error models for 802.11n, May 2017. URL: https://depts.washington.edu/funlab/wp-content/uploads/2017/05/Technical-report-on-validation-of-error-models-for-802.11n.pdf.

[11] Haoyong Lv and Hengyao Tang. Machine learning methods and their application research. In *2011 2nd International Symposium on Intelligence Information Processing and Trusted Computing*, pages 108–110, 2011. doi:10.1109/IPTC.2011.34.

[12] Julianna Delua. Supervised vs. unsupervised learning: What's the difference? URL: https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning.

[13] Wang Qiang and Zhan Zhongli. Reinforcement learning model, algorithms and its application. In *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*, pages 1143–1146, 2011. doi:10.1109/MEC.2011.6025669.

[14] Shweta Bhatt. 5 things you need to know about reinforcement learning, Mar 2018. URL: https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html.

[15] Beakcheol Jang, Myeonghwi Kim, Gaspard Harerimana, and Jong Wook Kim. Q-learning algorithms: A comprehensive classification and applications. *IEEE Access*, 7:133653–133667, 2019. doi:10.1109/ACCESS.2019.2941229.

[16] Xu Wang, Sen Wang, Xingxing Liang, Dawei Zhao, Jincai Huang, Xin Xu, Bin Dai, and Qiguang Miao. Deep reinforcement learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–15, 2022. doi:10.1109/TNNLS.2022.3207346.

[17] Rémy Grünblatt, Isabelle Guérin-Lassous, and Olivier Simonin. Simulation and Performance Evaluation of the Intel Rate Adaptation Algorithm. MSWIM '19, New York, NY, USA, 2019. Association for Computing Machinery. URL: https://doi.org/10.1145/3345768.3355921, doi:10.1145/3345768.3355921.

[18] Raja Karmakar, Samiran Chattopadhyay, and Sandip Chakraborty. Smartla: Reinforcement learning-based link adaptation for high throughput wireless access networks. *Computer Communications*, 110:1–25, 2017. URL: https://www.sciencedirect.com/science/article/pii/S0140366417306370, doi:https://doi.org/10.1016/j.comcom.2017.05.017.

[19] Fed4fire+ web site. URL: https://www.fed4fire.eu/.

[20] Nsnam. ns-3 home page. URL: https://www.nsnam.org/.

[21] Piotr Gawłowicz and Anatolij Zubow. Ns-3 meets openai gym: The playground for machine learning in networking research. MSWIM '19, page 113–120, New York, NY, USA, 2019. Association for Computing Machinery. URL: https://doi.org/10.1145/3345768.3355908, doi:10.1145/3345768.3355908.

[22] Lelio Campanile, Marco Gribaudo, Mauro Iacono, Fiammetta Marulli, and Michele Mastroianni. Computer network simulation with ns-3: A systematic literature review. *Electronics*, 9(2), 2020. URL: https://www.mdpi.com/2079-9292/9/2/272, doi:10.3390/electronics9020272.

[23] Piotr Gawłowicz and Anatolij Zubow. ns3-gym: Extending openai gym for networking research, 2018. arXiv:1810.03943.

[24] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. arXiv:1606.01540.

[25] ns-3: ns3::idealwifimanager class reference. URL: https://www.nsnam.org/docs/release/3.19/doxygen/classns3_1_1_ideal_wifi_manager.html.

[26] Guilherme Ferreira. Modeling the Processing Delays of Internet of Things Nodes in the ns-3 Network Simulator. Master's thesis, Faculdade de Engenharia, Universidade do Porto, 2017.

[27] w-ilab.2 hardware and inventory. URL: https://doc.ilabt.imec.be/ilabt/wilab/hardware.html.

[28] Tensorflow adam optimizer. URL: https://www.tensorflow.org/api_docs/python/tf/compat/v1/train/AdamOptimizer.

[29] Ruben Queiros, Eduardo Nuno Almeida, Helder Fontes, José Ruela, and Rui Campos. SMART - Self-adaptive Machine Learning Approach for Real-time Tuning of IEEE 802.11 PHY and MAC layers, March 2022. URL: https://doi.org/10.5281/zenodo.6394799, doi:10.5281/zenodo.6394799.