FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Rate Adaptation Algorithm using Reinforcement Learning for Delay Minimisation in a Wi-Fi Link

José Manuel de Sousa Magalhães

MASTER IN ELECTRICAL AND COMPUTERS ENGINEERING

Supervisor: Hélder Martins Fontes

October 18, 2022

Abstract

Since the original IEEE 802.11 (Wi-Fi) standard was released, there have been several improvements, corrections and then addition of new features over the years. With Wi-Fi evolution, it has become possible to use Wi-Fi for different types of situations. On the one hand, we have situations where the most favourable Quality-of-Service (QoS) metric is throughput, such as downloading or uploading files. On the other hand, we have situations where the most favourable QoS metric is delay, such as controlling drones, autonomous driving and controlling industrial systems.

In the past years, several Machine Learning (ML) approaches have been developed to optimise the performance of Wi-Fi networks. In most of these approaches, the throughput is the most common performance metric used to evaluate the developed solutions, leaving open other unexplored metrics, such as the delay.

This dissertation aimed to develop a Deep Reinforcement Learning (DRL) algorithm that can optimise the packet delay of a Wi-Fi link by adapting an algorithm developed by INESC TEC. This dissertation proposes the Smart Latency Aware Rate Adaptation (SLARA) algorithm to optimise the packet delay by training the algorithm with a delay-oriented reward function developed during this work. To achieve this reward function, a preliminary study was conducted to study the delay behaviour in other algorithms and find reference delay values for each MCS to normalise delay in the reward function. To validate this solution, several simulations of this and other algorithms were performed to evaluate and validate the improvements introduced by SLARA, as well as to identify aspects with room for improvement that could be addressed in future work. SLARA overall has a similar behaviour compared to Data-driven Algorithm for Rate Adaptation (DARA) but has some improvements such as lower max delay values and delay peaks due to more conservative Modulation and Coding Scheme (MCS) changes. This improvement makes SLARA an RAA with reliable delay values, which in the future can be useful for the scenarios mentioned above where the delay is the most important QoS metric.

Resumo

Desde que a norma original IEEE 802.11 (Wi-Fi) foi lançada, ao longo dos anos houve várias melhorias, correções e também a adição de novas características. Com a evolução do Wi-Fi, tornou-se possível a sua utilização para diferentes tipos de situações. Por um lado, temos situações em que a métrica de *Quality-of-Service* (QoS) mais favorável é o *throughput*, como o *download* ou *upload* de ficheiros. Por outro lado, temos situações em que a métrica de QoS mais favorável é o atraso, tais como o controlo de drones, condução autónoma e solução de controlo de sistemas industriais.

Nos últimos anos, foram desenvolvidas várias abordagens de *Machine Learning* (ML) para otimizar o desempenho das redes Wi-Fi. Na maioria destas abordagens, o *throughput* é a métrica de desempenho mais comum utilizada para avaliar as soluções desenvolvidas, deixando em aberto outras métricas inexploradas, tais como o atraso.

Esta dissertação tinha como objetivo desenvolver um algoritmo de *Deep Reinforcement Learning* (DRL) para otimizar a o atraso de pacotes de uma ligação Wi-Fi, adaptando um algoritmo desenvolvido pelo INESC TEC. Esta dissertação propõe o algoritmo *Smart Latency Aware Rate Adaptation* (SLARA) para optimizar o atraso do pacote através do treino do algoritmo com uma função de recompensa orientada para o atraso desenvolvida durante este trabalho. Para alcançar esta função de recompensa, foi realizado um estudo preliminar para estudar o comportamento do atraso noutros algoritmos e encontrar valores de referência de atraso para cada *Modulation and Coding Scheme* (MCS) para normalizar o atraso na função de recompensa. Para validar esta solução, foram realizadas várias simulações deste e de outros algoritmos para avaliar e validar as melhorias introduzidas pelo SLARA, bem como para identificar aspetos com margem para melhorias que poderiam ser abordados em trabalhos futuros. O SLARA em geral tem um comportamento semelhante em comparação com o algoritmo *Data-driven Algorithm for Rate Adaptation* (DARA), mas apresenta alguns melhoramentos, tais como valores de atraso máximo e picos de atraso mais baixos, devido a alterações de MCS mais conservadoras. Esta melhoria faz do SLARA um RAA com valores de atraso fiáveis, que no futuro pode ser útil para os cenários acima mencionados onde o atraso é a métrica de QoS mais importante.

Acknowledgments

This work is a result of the project "DECARBONIZE – DEvelopment of strategies and policies based on energy and non-energy applications towards CARBON neutrality via digitalization for citIZEns and society" (NORTE-01-0145-FEDER-000065), supported by Norte Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, through the European Regional Development Fund (ERDF).

I want to express my gratitude to my supervisor, Hélder Fontes, who always helped and guided me throughout this dissertation. In addition and no less important, I would like to leave a special thanks to Rúben Queirós, who, despite not formally being my supervisor, was tireless during these months of work, always showing a willingness to help and guide me. Without a doubt, I would not have completed this work without his help and his contribution. I thank them both for their patience and support, and I apologize for the moments when I failed or did not put in the necessary effort. I would also like to thank Prof. Rui Lopes Campos who also guided and helped me in the first steps of this dissertation.

I also want to thank my friends, who have accompanied me during these five years, with whom I have lived remarkable moments, and with a spirit of mutual help, we have reached this point. To my family, I want to express my gratitude for all their support. I thank them for their investment in my education, affection, and trust in me. Without my parents, I would not be at this point and, to them, my eternal gratitude. Finally, I want to thank my girlfriend for all the support during my academic career and this dissertation. I also thank her for her patience in the most frustrating moments and for always trying to keep me motivated.

José Manuel de Sousa Magalhães

"Success is not final; failure is not fatal: It is the courage to continue that counts."

Winston S. Churchill

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

AI	Artificial Intelligence
ARA	Advanced Adaptation Algorithm
CDF	Cumulative Distribution Function
DARA	Data-driven Algorithm for Rate Adaptation
DQN	Deep Q-Network
DSSS	Direct Sequence Spread Spectrum
DL	Deep Learning
DRL	Deep Reinforcement Learning
FHSS	Frequency Hopping Spread Spectrum
IEEE	Institute of Electrical and Electronics Engineers
INESC TEC	Institute for Systems and Computer Engineering, Technology and Science
LAA	Link Adaptation Algorithm
MAC	Medium Access Control
MCS	Modulation and Coding Scheme
MIMO	Multiple-Input Multiple-Output
MiRA	MIMO Rate Adaptation Algorithm
ML	Machine Learning
MLRA	Machine Learning-based Rate Adaptation
MU-MIMO	Multi-User MIMO
NN	Neural Network
ns-3	Network Simulator 3
OFDMA	Orthogonal Frequency Division Multiple Access
PER	Packet Error Ratio
QoS	Quality of Service
RAA	Rate Adaption Algorithm
RL	Reinforcement Learning
RSSI	Received Signal Strength Indicator
SNR	Signal-to-Noise Ratio
SL	Supervised Learning
SLARA	Smart Latency Aware Rate Adaptation
STEM	Science, Tecnology, Engineering and Mathematics
TCP	Transmission Control Protocol
TS	Trace-based Simulation
UDP	User Datagram Protocol
UL	Unsupervised Learning
Wi-Fi	Wireless Fidelity
WLAN	Wireless Local Area Network

Chapter 1

Introduction

1.1 Context

Nowadays, Wi-Fi is the most common wireless local area network. Since the original version, based on the IEEE 802.11 standard, was first released in 1997, Wi-Fi has continually evolved with faster speeds and further coverage. Several versions of the IEEE 802.11 standard have emerged over the years. Currently, Wi-Fi is in its 6th version. On one hand, there are scenarios where throughput is the most relevant Quality-of-Service (QoS) metric, such as when we download or upload files. On the other hand, there are scenarios where the delay is the most relevant metric, such as controlling drones and industrial systems or autonomous driving.

Over the years, rate adaptation algorithms (RAAs) have been determinants in the performance of Wi-Fi networks. RAAs optimise link performance by allowing data transmission at different rates depending on network conditions, such as those associated with the scenarios mentioned above. This matter has been investigated in the past years, and as a result, several algorithms have been created. The diversity and innovation of some of these new algorithms allow a division into two groups, as the authors of [3] mentioned: heuristic-based (algorithms with pre-defined and hard-coded rules, the traditional ones) and Machine Learning (ML)-based algorithms (with the ability to learn from experience or trial and error, and gradually replacing the traditional algorithms).

1.2 Problem and Motivation

Several adaptation algorithms have been proposed in state of the art, such as in [4], [5], [6], [7], [8] and [9]. RAAs may be classified according to the metrics used to evaluate the link quality, such as packet loss, Signal-to-Noise Ratio (SNR), transmission time, throughput or combined metrics. Typically, these algorithms tend to improve throughput, but they are not validated in scenarios with low latency requirements. The existence of algorithms that aim for the best delay performance (such as the heuristic-based algorithm in [4]) is limited. It is important to note that,

out of the three types of delay (queuing, propagation or transmission), only the transmission delay is influenced by rate adaptation.

INESC TEC has been working on an ML-based RAA [10] named DARA. DARA aims to optimise throughput in a Wi-Fi link and, as a result, presented a higher throughput when compared to Minstrel High Throughput and equals the performance of Ideal Wi-Fi RA algorithm, from ns-3.

In this work, we seek to adapt DARA to explore this problem and develop a solution to minimise the Medium Access Control (MAC) layer transmission delay of a Wi-Fi link compared to existing algorithms, while maintaining the highest throughput possible. This solution could be helpful in scenarios such as real-time control of robotic systems over wireless networks and video streaming for virtual and augmented reality use cases with edge computing. Although IEEE 802.11n is the standard used in this work due to its mature implementation in ns-3, the work carried out in this dissertation is applicable to all other Wi-Fi variants in which RAA are used, including the most recent ones. Given this, it is possible to improve the performance of the existing algorithms or develop algorithms with different approaches. The innovative and noticeably better results than traditional algorithms could imply that current versions of Wi-Fi can be used more efficiently, aiming to achieve the full potential of Wi-Fi with better reliability (by focusing the algorithm on delay, it may be possible to achieve consistent values and reduce the number of retransmissions and packet loss). This leads to the conclusion that this work is innovative, given that the existence of similar work that aims to choose an optimal data rate that optimises the MAC transmission delay is scarce, especially algorithms with the same objective developed using ML.

1.3 Objectives

The main objective of this dissertation is to study, evaluate and validate an algorithm based on ML that optimises the configuration of a Wi-Fi connection to minimise the packet transmission delay in the MAC layer. To reach this, the DARA algorithm from INESC TEC will be the reference for developing this new algorithm, readapting the work to achieve the expected results. With this in mind, several objectives must be accomplished to reach this goal:

- Creation of a simulation environment to train and test the ML-based algorithm, integrating ns-3 with ns3-gym to reproduce and replicate specific scenarios where this type of algorithm fits.
- Development of the proposed solution. Adaptation of the DRL model and the respective reward function.
- Evaluation of the performance of the solution, in the simulation environment, against other algorithms available in the state of the art.

1.4 Document Structure

This document is structured in five chapters:

- Chapter 2 - describes the state of the art, background and related work;
- Chapter 3 - presents the proposed solution, describes the simulation environment and the preliminary study with the corresponding results and conclusions;
- Chapter 4 - presents the results to validate the solution, compares the solution with algorithms from the state of the art and presents a detailed analysis of the results obtained;
- Chapter 5 - exposes the main conclusions and key ideas to retain in this dissertation, what was learned in the process and addresses the future work.

Chapter 2

State of the Art

This chapter covers the topics considered relevant to understanding the problems exposed in Chapter 1. It is divided into the following sections:

- IEEE 802.11 — a brief introduction on the IEEE 802.11, how it all started, the most important improvements and functionalities throughout the standard versions and some concepts;
- Machine Learning — introduction to ML and a brief background on the usefulness and possible applications;
- Software tools and frameworks — brief description of software tools and framerworks;
- Rate adaptation algorithms — background on the RAAs related to this work;

2.1 IEEE 802.11

In 1997, the IEEE 802.11 original standard was released for the first time to the consumers. There have been several improvements, corrections and new features ever since. The most relevant are the following:

- **IEEE 802.11** - Provided a maximum of 2 Mbit/s of data rate in the 2.4 GHz band Wireless Local Area Networks (WLANs) using Frequency Hopping Spread Spectrum (FHSS) or Direct Sequence Spread Spectrum (DSSS);
- **IEEE 802.11b** - Can be called "Wi-Fi 1", extended data rate up to 11 Mbit/s using only DSSS in the 2.4GHz band;
- **IEEE 802.11a** - Also refered as "Wi-Fi 2", Orthogonal Frequency Division Multiplexing (OFDM) scheme instead of FHSS or DSSS, which allows data rates up to 54 Mbit/s in the 5 GHz band;
- **IEEE 802.11g** - Also known as "Wi-Fi 3", extended data rate up to 54 Mbit/s in the 2.4 GHz band;

- **IEEE 802.11n** - Also well known as "Wi-Fi 4", this version was released in 2009. The addition of multiple-input multiple-output antennas (MIMO) technology for separate spatial streams (four spatial streams), frame aggregation and wider channels (from 20 MHz to 40 MHz) allowed to extend the maximum data rate up to 600 Mbit/s in the 2.4 GHz and 5 GHz bands;

- **IEEE 802.11ac** - "Wi-Fi 5" adds support for wider channels (up to 160 MHz), increased number of spatial streams (up to eight) and the Multi User MIMO (MU-MIMO) for concurrent transmissions to different stations in the downlink direction. This allowed to extend the total data rate up to 1300 Mbit/s, in the 5 GHz band;

- **IEEE 802.11ax** - "Wi-Fi 6" has a denser modulation, uses up to 1024-QAM and that increases the data rates by 35%. MU-MIMO on the uplink is available, in addition to the MU-MIMO on the donwlink used in IEEE 802.11ac. There is also an Orthogonal Frequency Division Multiple Access (OFDMA)-based scheduling to reduce overhead and latency and a robust high-efficiency signaling for better functioning with the Received Signal Strenght Indicator (RSSI);

In a Wi-Fi link, there are three main types of delay: propagation, transmission and queuing. The propagation delay is the time it takes for the last bit of a packet to reach the destination after the packet is transmitted to the transmission medium. Then it has to pass through the medium to reach the destination. The transmission delay is the time it takes to transmit a packet from the host to the transmission medium. The transmission delay depends on metrics such as throughput (transmission times decrease when throughput is high), packet loss and the number of retransmissions (which increases the transmission time). The queuing delay is the time a packet is queued and depends on the number of packets in the queue and the rate at which they are being served. The authors of [4] have shown that queuing and transmission delays may have the biggest impact, given that they depend on metrics that can be used in the algorithm, indirectly influencing the delay variation. In this work, the focus will be the transmission delay. In Figure 2.1, it is possible to observe a simple diagram illustrating the types of delay exposed above. It is possible to understand where and in which order these delays occur.

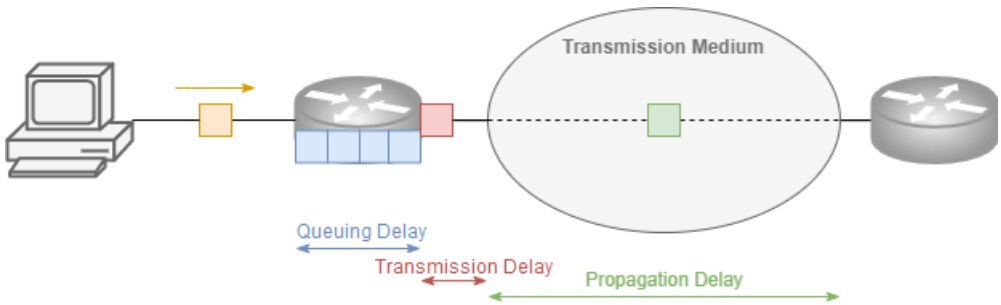


Figure 2.1: Types of delay in a Wi-Fi link.

2.2 Machine Learning

ML is a sub-field of Artificial Intelligence (AI) and aims to reproduce human intelligence by learning from experience and the surrounding environment. ML algorithms use data as input to predict output values without being programmed to do it. There are four different methods of machine learning: Reinforcement Learning (RL), Supervised Learning (SL), Unsupervised Learning (UL) and Semi-Supervised Learning (a combination of SL and UL). ML has several applications, such as prediction, computer vision or semantic analysis. These different applications are distributed among the different methods, as shown in Figure 2.2. It is possible to use ML algorithms in several research areas such as science, technology, engineering and mathematics (STEM), which makes it a versatile tool.

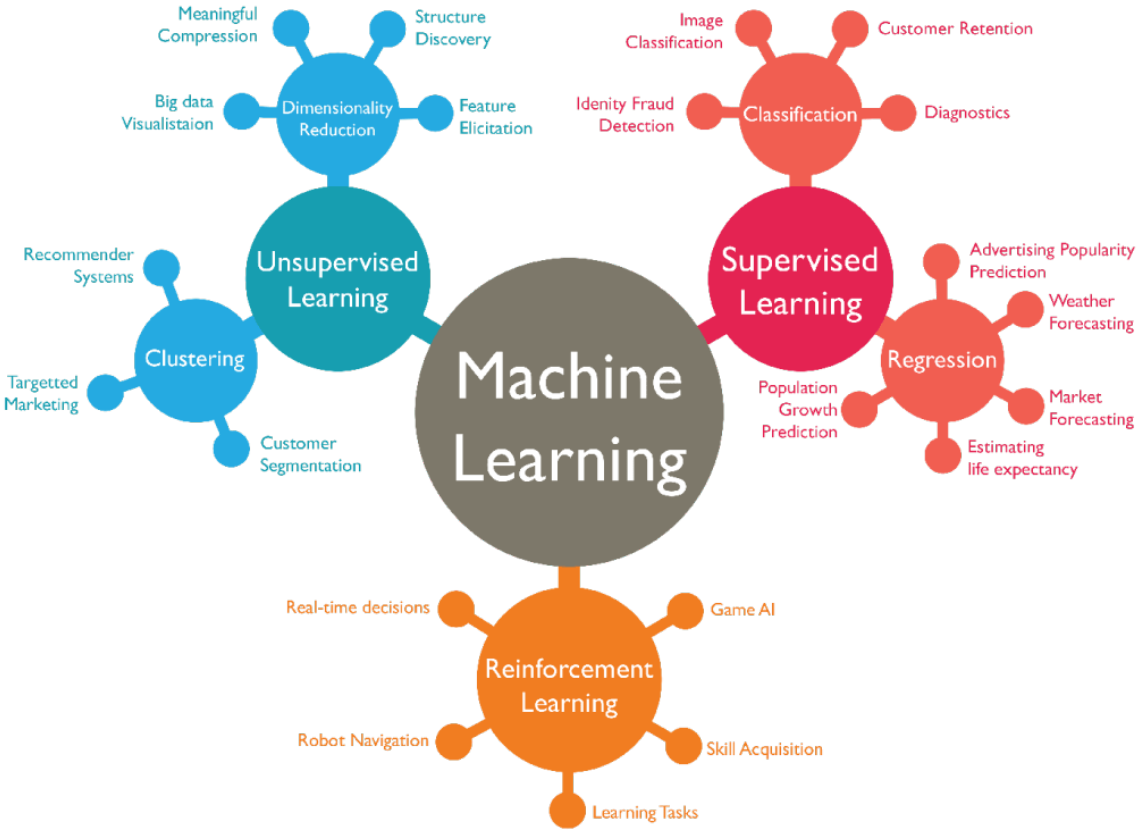


Figure 2.2: Machine Learning methods scheme from [1].

2.2.1 Deep Learning

Deep Learning (DL) is a sub-field of ML and brings an advanced approach to ML, given its incapacity to solve more complex tasks. DL uses artificial neural networks (NNs) (an algorithm inspired by the human brain) to learn from an extensive amount of data. Learning is based on experience, i.e. the algorithm learns as a task is performed repeatedly. A NN is composed of different layers of neurons, where there is the input layer (which accepts inputs), the output layer

(which returns the result) and the remaining layers are hidden layers (which process inputs). The term "deep" is due to the NNs of this method having various hidden layers (deep NNs) that increase their level of complexity and abstraction. Deep NNs have different types, such as convolution NNs, recurrent NNs, multi-layer perceptrons or Q-Learning. DL can be applied to different ML methods when they face complex problems with larger datasets or a high number of variables, and it is necessary to build more accurate models. These models can be achieved by more oversized layers (more neurons) to detect more complex patterns.

2.2.2 Reinforcement Learning

RL is an ML technique that focuses on goal-directed learning from interaction. The algorithm (*agent*) evaluates the current *state*, takes an *action* and receives a *reward* from the *environment*. The algorithm learns how to make good decisions (actions) by Trial-and-error, performing several attempts that could lead to better or worse performance. The main goal is to maximise the total *reward*. Although the *reward* policy is defined, there are no rules about how to take the *action* that results in the best *reward*. It is up to the algorithm to understand how it can perform the task to maximise the *reward*. The algorithm initially makes arbitrary decisions, and as it learns, it finds increasingly complex patterns. RL has several applications, such as real-time decisions, autonomous driving, or optimising video streaming quality. In Figure 2.3 it is possible to observe a block diagram representing a typical RL model.

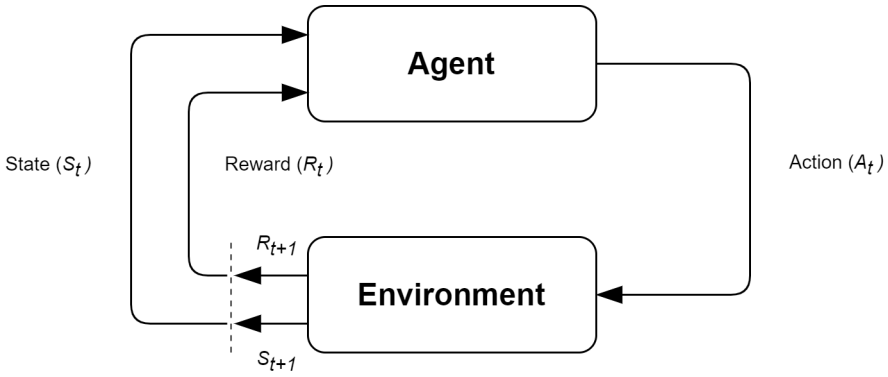


Figure 2.3: Reinforcement Learning block diagram.

Deep Reinforcement Learning (DRL) combines RL with DL to solve the limitations of RL in complex environments with large state spaces or high dimensional inputs.

2.2.3 Supervised Learning

SL is an ML technique that uses labelled datasets to train algorithms to classify data or predict an output from a given input. This model tends to be mainly applied to problems such as image classification or market forecasting, as observed in Figure 2.2.

2.2.4 Unsupervised Learning

UL is an ML technique that aims to analyze and cluster unlabeled datasets. This model discovers patterns or data grouping without external interference (such as human interventions or labelled datasets). This model can discover similarities and differences between data and can be used in image recognition or data analysis.

2.3 Software tools and frameworks

In this work, three software tools and frameworks (ns-3, ns3-gym and TF-Agents) are used, as shown in Figure 2.4.

Ns-3 [11] is a discrete-event network simulator for research and educational use. It is a highly realistic and one of the most commonly used, leading wireless network simulators. Ns-3 allows excellent control over the simulation scenario and accurately measures metrics such as the delay. Although ns-3 is a simulator, we can use Trace-Based Simulation (TS). TS focuses on capturing SNR traces of a real experiment and reproducing them by using the traces in ns-3. The TS aims to improve the simulation accuracy (as the authors mention in [12]).

The ns3-gym framework [13] is the first framework for RL research in networking. Ns3-gym integrates OpenAI Gym (a toolkit for RL commonly used in research) and ns-3 network simulator. It allows representing an ns-3 simulation as an environment in the Gym framework for the RL agent's learning purposes.

TF-Agents [14] is a RL library in TensorFlow. TF-Agents is easy to implement and deploy and provides well tested and modular components that can be modified and extended. It enables a fast code iteration, with good test integration.

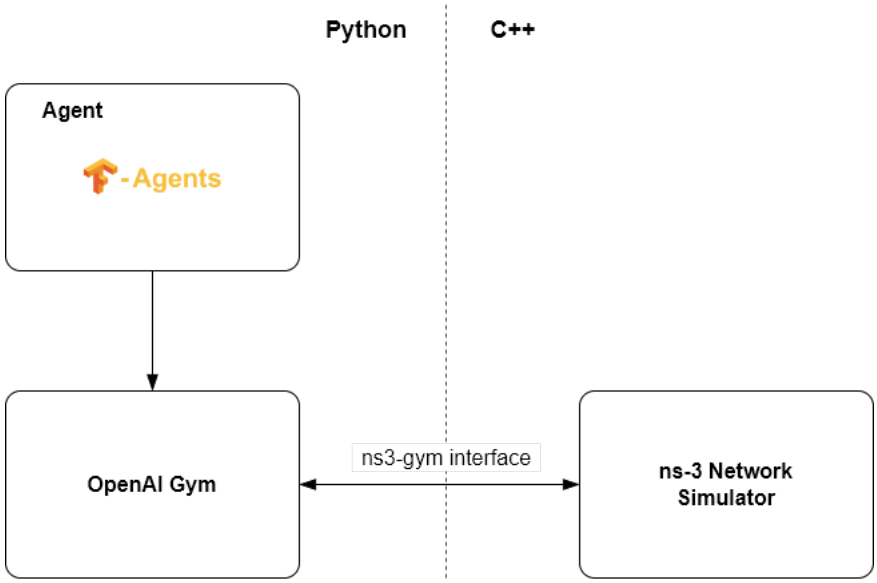


Figure 2.4: Software tools and architecture scheme.

2.4 Rate adaptation algorithms

Several RAAs have been developed. However, the number of algorithms that aim to minimize the delay or latency is limited. With this, this section exposes ML-based and heuristic-based RAAs that may be relevant not only for the development of the work but also as a comparison..

2.4.1 Heuristic-based algorithms

There is an approach where a latency-aware rate adaptation algorithm [4] is proposed. In this document, the authors show how current rate adaptation solutions work and prove that they work well for throughput but not for latency. These theorems are duly corroborated and essentially consist of making rate choices, considering link metrics such as the number of retransmissions, number of packets in the queue and Packet Error Ratio (PER). Finally, the proposed algorithm showed a performance improvement in terms of latency compared with algorithms such as Advanced Rate Adaptation Algorithm (ARA) and MIMO Rate Adaption Algorithm (MiRA) mentioned in [4], while maintaining a similar throughput. The authors also used Trace-based scenarios, which forced SNR to values captured in a real experiment.

Minstrel is a heuristic-based algorithm that aims to have the highest throughput. As mentioned in [15], this algorithm is divided into three parts: retry chain, rate decision and statistic calculation. The retry chain consists of four rate-count pairs where a packet is transmitted first with the first pair, and if the number of retries reaches the count, it moves on to the next pair and so on if the packet is not sent successfully. Next, rate selection is used to choose the retry chain rates and has two types of transmissions: normal transmission (which occurs 90% of the time) and sampling transmission (which occurs 10% of the time). In normal transmission, the first two rates are those with the highest throughput, the rate with the highest probability of success, and the lowest available rate. In sampling transmission, the data frames are sent with a random rate and the first rate is set to the higher rate out of the sample rate, the second rate is set to the lower rate out of the sample rate, and the other remains the same. In the statistic calculation, Minstrel maintains that the probability of successful transmission is based on the historical success rate at each data rate and is used to estimate the throughput of each rate and the retry chain is evaluated based on this estimate every 100 ms.

2.4.2 Machine Learning-based algorithms

The authors of [16] proposed an algorithm that aims to enhance the Robust Rate Adaptation Algorithm with a RL approach. The objective of the algorithm is to find the "best" throughput possible. Given that, it is defined as action the data rate selected and an action becomes better as the related packet loss decreases. The reward is the trade-off between the packet loss minimisation and the throughput maximisation. For each data rate a set of ten packet loss intervals were defined. Since there are 8 possible actions and each action has a set number from 0 to 7, respectively, from the lowest to the highest data rate, 80 states are identified. The reward is calculated with the packet

loss and the rate, and with reduced packet loss and high rate, the higher is the reward. The authors selected 0.45 to use as trade-off between packet loss minimisation and throughput maximisation, but do not explain how they achieved that value. The simulations results presented an improvement in the packet loss from 28.016% to 22.599% and in the mean delay from 35.255 ms to 16.122 ms, compared to Robust Rate Adaptation Algorithm.

In [17] the authors proposed a RAA to augment drive-thru internet using RL-based RA that efficiently selects the rate for every vehicle's egress frame. The authors defined the overall data amount that a vehicle can transmit to a roadside access point, and aim to maximise this metric. They defined as the state a set of the most recent SNR records. The action is the data rate to be selected and the reward depends on the transmission result (1 if it is successful or 0 if it fails) and the packet size from that interval.

The authors of [8] proposed the ML-based RA (MLRA) algorithm to identify correlations between the rate, goodput performance and link quality. The aim of this RAA is to find the rate with the highest goodput based on the link quality indicators and estimated congestion degrees. The results shown that MLRA outperforms other RAAs, such as Minstrel, MiRA and lwlwifi by around 133%-658%.

DARA is a DRL algorithm that aims to optimise throughput in a Wi-Fi link, as the authors mention in [10]. The authors describe the DRL model and the implementation used for training and evaluating the algorithm.

In the DARA DRL model, the authors' goal was to train an agent that learns to choose the optimal MCS for a fixed time interval based on the SNR of the previous time interval. The action space is a subset list of MCS values in IEEE 802.11n, considering eight different *actions* (MCS_0 to MCS_7). The *state* is the average SNR value observed in the previous time interval from the received frames. The reward is the product of the frame success ratio and the normalised MCS of the previous time interval, as shown in Eq.2.1.

$$reward = \frac{MCS_n}{MCS_7} \times FER, n \in [0,1,...,7] \tag{2.1}$$

The authors mention that the combination of success ratio and the highest MCS possible avoids scenarios where the agent chooses either the lowest MCS to preserve the highest frame success ratio or the highest MCS, even if frames are not delivered.

In the DARA implementation, the authors mention that they use ns-3 (with version 3.35) and the ns3-gym framework to develop the simulation environment and use the TF-Agents library to create the RL agent.

For the ns-3 environment, the IEEE 802.11n is the Wi-Fi standard, and the Wi-Fi MAC defined is Ad-hoc. They defined *constant speed* as the propagation delay model and Friis as the propagation loss model. The defined frequency is 5180 MHz with a channel bandwidth of 20 MHz. A user

can choose between DARA, Ideal or Minstrel for the remote station manager. The traffic protocol is UDP, and the link is saturated with UDP traffic, so there is always traffic on the link.

For the action, DARA only supports one MCS per time interval, and the new MCS chosen by the agent is selected at each time interval.

For the observation, DARA tracks the number of the received frames and their SNR. Then calculates the average SNR for the current time interval. If the selected MCS is high enough that no frames are delivered, SNR cannot be calculated and then it assumes that in that time interval, SNR is zero, and in the next time interval, chooses the lowest MCS.

For the reward, its calculation depends on whether the transmitted frame was ACKed or not. First, the frame is received, and the transmitting node receives this information through ACK / Block ACK. Next, the frame is not received, and the transmitting node receives this information through Block Ack. Finally, none of this information reaches the transmitting node during the timeout period. FER is calculated based on the scenarios mentioned above, and the MCS is obtainable through the action taken for that time interval.

DARA is implemented in Python for the agent architecture and uses the Deep Q-Network (DQN) learning algorithm and the TF-Agents library, as mentioned above. The agent receives the step of the previous time interval where it gets the observation and the reward and then chooses the action for the next interval. The DQN learning algorithm has several parameters such as the *Observation space* (a one-dimensional float value between 0.0 and 1.0, corresponding to the division of the final SNR in dB by 100, to stabilise the learning process), the *Action space* (a one-dimensional integer value which represents the MCS to select for the next time interval), the *Optimiser* (Adam algorithm with a 10^{-2} learning rate to minimise the network loss function), the *Epsilon greedy* (value between 0 and 1, determines the percentage of random actions and starts at 1 and decreases over a defined period until 0.1 where only 10% of actions are random), the *Q-Network* (has 2 layers of interconnected parameters with 32 units each) and the *Replay buffer* (has a size of 10^6 trajectories and with the simulation progresses, trajectories are added to the replay buffer and for every agent train, it randomly samples a batch of 64 trajectories).

Finally, the agent can run a training session or an evaluation session. In the training session, it starts to collect the simulation trajectories and fill up the replay buffer. A trajectory is composed of a time step (environment initial observation), an action step (the action taken considering the previous time step) and the next time step (the new observation and the reward obtained from the previous action step). The replay buffer fills up during the simulation until the *Game Over Module* informs that the simulation is over. Then, the agent trains randomly by obtaining from the replay buffer trajectories updating the weights, and increasing the time step counter. The user can adapt the number of episodes and how epsilon greedy decreases over the training process. At the end of the training session or when the user wants to pause it, it is possible to save the progress with a checkpoint, which makes it possible to use this policy later in an evaluation session. In the evaluation session, the user can load a saved and trained policy and the epsilon greedy changes to zero to avoid exploratory attempts. It is possible to obtain logs from the throughput at the application level and, in addition, obtain packet capture files for debugging purposes and track and

log the reward during the session.

Chapter 3

SLARA Model and Implementation

In this Chapter, with the knowledge obtained during the study performed in Chapter 2, it is presented detailed information about the proposed solution. First a preliminary study is described to learn more about the delay behaviour in order to help to develop a reward function for the DRL model. Given the conclusions from the preliminary study, it is presented the SLARA model which describes the DRL model and the developed reward function. Then it is described the implementation of the ns-3 simulation and the ns-3 gym. Finally the DRL agent architecture is presented with the respective details.

3.1 Preliminary study

At the beginning of this work, for developing a future reward function focused on delay, there was doubt about how to normalise it as it is done in DARA to stabilise the learning process with scaled values. This doubt arises because there is a maximum throughput value associated with each MCS, which only decreases according to SNR degradation or by switching to a lower MCS. Regarding the delay, it is implicit that each MCS has a minimum delay and that this delay can grow indefinitely due to several factors such as SNR degradation, switching to a lower MCS or retransmissions due to problems in the transmission medium or low link quality. This means that sometimes a higher MCS has higher delay than a lower MCS because it has more packet retransmissions. This preliminary study aims to improve our understanding on the delay behaviour in a Wi-Fi link. Besides that, the study intends to identify the delay limits for each MCS with the distance variation, normalise delay to formulate a reliable reward function and overcome the instability around the delay.

The simulation scenario, from Table 3.1, has been prepared to test the behaviour of some of the algorithms exposed in Section 2.4 (Minstrel and Ideal), where there are two nodes, and the transmitting node moves over 120 seconds from an initial position, 10 metres away from the other node, until the 1350 metres distance between nodes is reached. The objective is to plot all the data obtained from the simulation and obtain two different graphs. The first graph plots the relationship between the delay of all the packets in the simulation and the distance. For the second graph, the

Parameter	Description
Wi-Fi Standard	IEEE 802.11n
Propagation Delay Model	Constant Speed
Propagation Loss Model	Friis
Traffic	UDP
Traffic Data Rate	75 Mbit/s
Packet Size	Constant (1000 Bytes)
Number Of Nodes	2
Duration	120 seconds
Mobility Model	Waypoint (10 meters to 1350 meters) / Constant (variable distance)
Frame Aggregation	Disabled
Remote Station Manager	Minstrel / Ideal

Table 3.1: Preliminary Study ns-3 Environment Parameters.

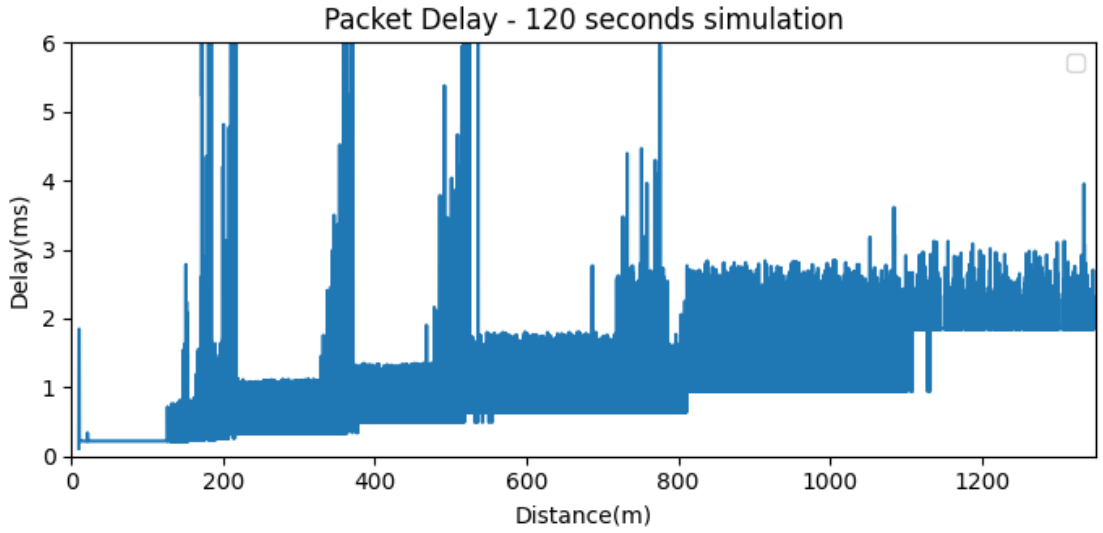
"intervalValues" list of the function "MyGymEnv::RxTime" in Listing 3.7 stores and then sorts the delays of the received packets in a 100ms interval to then obtain the minimum and maximum delay, the 10th, 50th (median) and 90th percentiles of the delay and that are plotted relative to the distance.

In Figure 3.1a it is visible that the delay in Minstrel varies a lot. It is possible to observe several delay peaks mainly caused by packet retransmission. Since Minstrel is an algorithm built always to maintain the highest throughput possible, it is expected that with its MCS choices, the values of the delay are not as optimal as those of the throughput since higher throughput does not always generate a lower delay, and these choices may be wrong from a delay point of view. Figure 3.1b shows that some of these peaks happen due to the shift to lower MCSs (each step is a different MCS) and are visible at "P90" and "Max", unlike the peaks mentioned above. This problem might indicate that the link quality is low at the instant when the MCS change occurs and again indicate that the MCS change should be sooner, from the delay point of view.

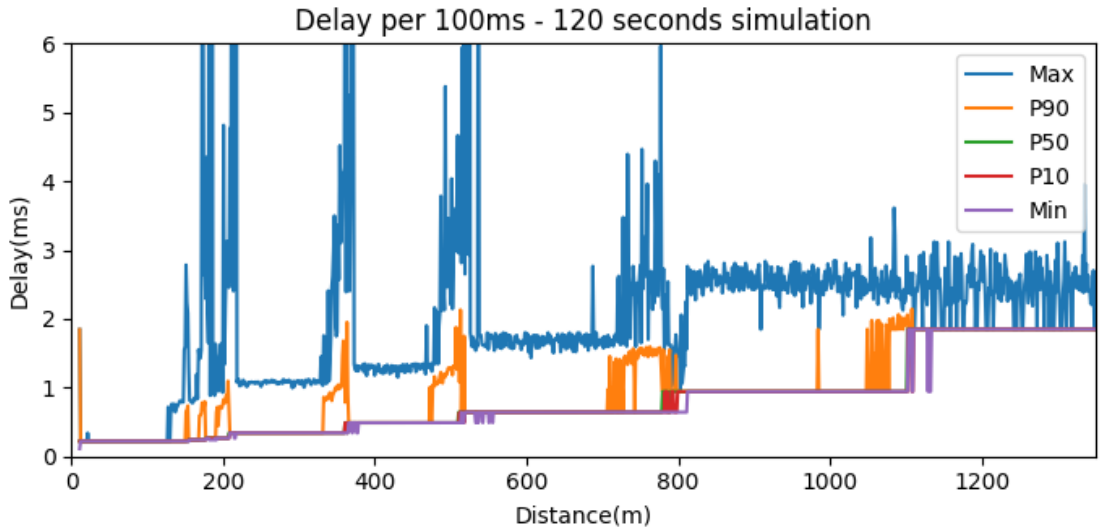
In Figure 3.2 it can be seen that the delay variation is low with only a few peaks. This behaviour is expected because Ideal is an algorithm only available in ns-3. Its function is to implement an "ideal" rate control and even with the good results. It is impossible to implement Ideal in a real context, since it knows the status of the link in the receiver and use this to adjust the data rate. This algorithm can be used as a reference to analyse the performance of future SLARA results.

The results verified that Minstrel is a greedy algorithm, while Ideal is a more conservative algorithm when switching MCS. It is possible to conclude that, for greedy algorithms, the delay analysis is more complex due to the high variation of values that results from its peaks, making it difficult to find a reference value to formulate a reward to train an algorithm. Concerning conservative algorithms, it is concluded that these have more constant delay values and with minor variations.

Finally, several simulations have been executed (one for each MCS), maintaining the same simulation scenario from the previous simulations. In Figure 3.3 there are some of these results and once again the presence of peaks in the delay is observed. These peaks represent the limits



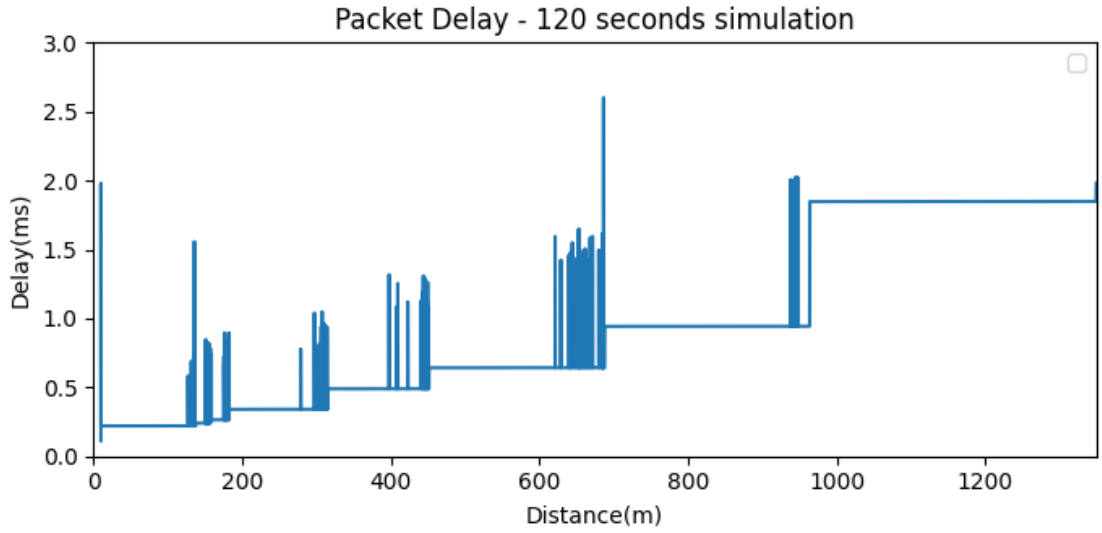
(a) Minstrel - All packets delay vs distance.



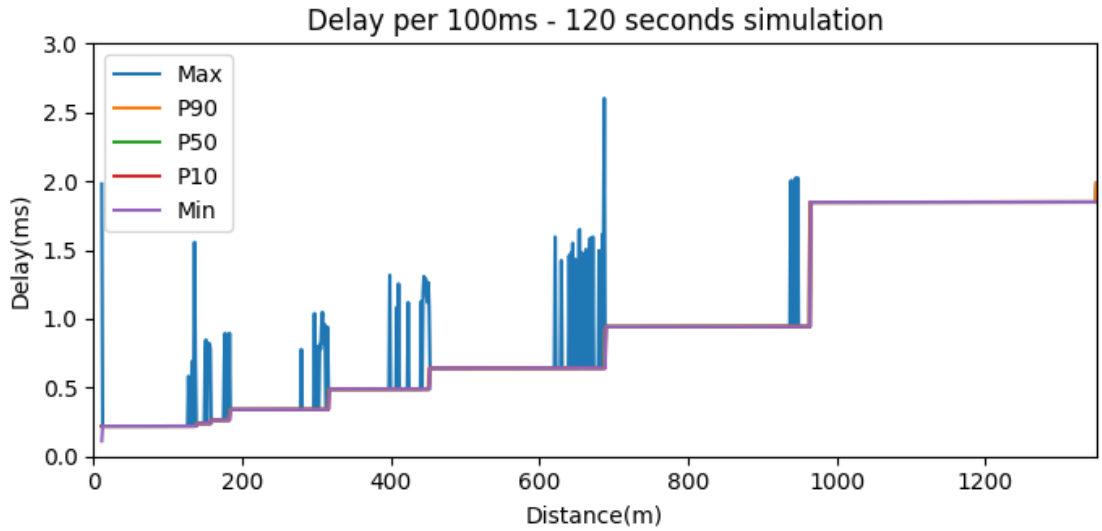
(b) Minstrel - 100ms interval delay stats vs distance.

Figure 3.1: Minstrel delay behaviour over a 120 second simulation.

of each MCS due to the SNR of these instants being lower than the SNR required to maintain the respective MCS. Until they reach these limits, each of these MCSs has linear and constant delay values. Given this, the average delay (until the limit, excluding the peak values) for each MCS was stored to reference the delay for each MCS, as it is already done with the data rate. In Table 3.2 it is possible to consult the obtained values.



(a) Ideal - All packets delay vs distance.



(b) Ideal - 100ms interval delay stats vs distance.

Figure 3.2: Ideal delay behaviour over a 120 second simulation.

3.2 SLARA Model

The objective of this work is to train a DRL agent to learn the optimal MCS that should be used for a time interval t based on the SNR of received frames from the previous time interval $t+1$, in order to minimise the delay, as observed in Figure 3.4. The action space A is a subset list of the MCS values from IEEE 802.11n standard with eight possible actions, as referenced in Section 3.3. The state is the average SNR observed during the time interval t. In the SLARA model, the preliminary reward is defined in Equation 3.1 and is the product of the FER and the normalised delay. The normalised delay is given by $\frac{x}{y}$, where x is the delay reference value and y is a delay

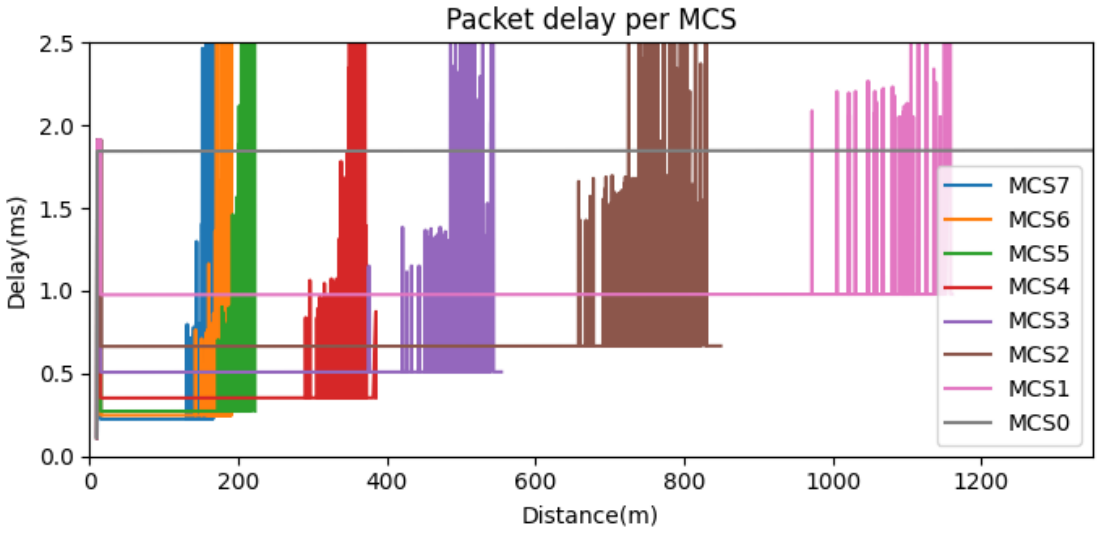


Figure 3.3: Packet delay vs Distance for each MCS.

value observed in the previous time interval. The goal is to find an optimal reference delay x to use and develop a strategy so that the y delay represents the state of the delay in the respective time interval.

$$reward = \frac{x}{y} \times FER \tag{3.1}$$

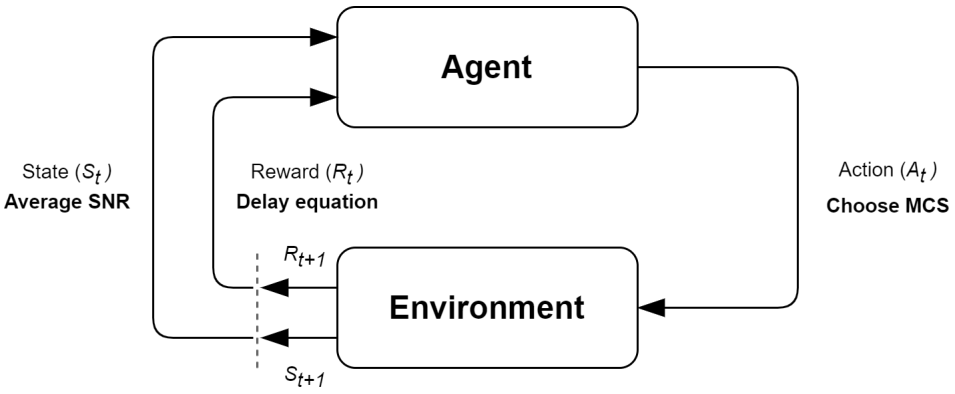


Figure 3.4: SLARA model scheme.

Given this introduction, as mentioned in Section 3.1, the development of the reward function would be challenging due to the higher complexity of normalising delay compared to throughput. The inexistence of delay values that reference each MCS, such as the data rate associated with each MCS, increases this complexity. Table 3.2 presents the values obtained in the preliminary study and can be addressed to solve the previous problem. The reward's goal is to minimise the delay, and 0.224ms is the delay reference x, corresponding to the MCS_7 delay. Equation 3.2 is the new

MCS	Packet Delay (ms)
7	0.224
6	0.248
5	0.272
4	0.352
3	0.508
2	0.664
1	0.976
0	1.912

Table 3.2: Average packet delay for each MCS.

reward function for SLARA, and it is the product between the normalised delay and the packet success ratio. The normalised delay ($\frac{0.224}{y}$) is the quotient between the minimum delay (delay associated with MCS_7 - 0.224ms) and y, a flexible input variable around the observed delays in a time interval.

$$reward = \frac{0.224}{y} \times FER \tag{3.2}$$

The y input variable can take the different percentiles, the minimum delay, maximum delay or even the average delay of the 100ms interval, as shown in Section 3.1. This reward function may be used for different purposes and be optimized relative to the version of SLARA trained with this function. In this version of SLARA, the variable y assumes the 90th percentile, due to the results in Figure 3.1b, and 85th and 95th percentiles to understand later how the algorithm behaves with different percentiles. On one hand, it is possible to conclude that for the minimum delay and the 10th and 50th percentiles, its use would not be the most optimal because as they present low values, the agents' actions would not be regularly penalized if there are several delay peaks, and they could even ignore longer delays because these correspond to higher percentiles. On the other hand, the maximum delay would also not be optimal because it mostly corresponds to delay peaks caused by retransmissions and sometimes does not summarize the quality of the action, since although the maximum value is higher than the maximum value of another action, it does not mean that globally the delay values of the second action will be better than the first action. Finally, to penalise high delay values, it was decided that the reward will be null if y is greater than or equal to 2ms (a value arbitrarily chosen to be close to the 1.912ms associated with MCS_0 in Table 3.2), as presented in Equation 3.3.

$$reward = \begin{cases} \frac{0.224}{y} \times FER, & if \quad y < 2 \\ 0, & if \quad y \geq 2 \end{cases} \tag{3.3}$$

3.3 Ns-3 simulation

First, to train and test the final solution it was necessary to develop the simulation environment with adequate characteristics for the development of the solution. As seen in Table 3.3 the Wi-Fi

standard chosen is IEEE 802.11n since currently, the ns-3 documentation is well consolidated for IEEE 802.11n, while for IEEE 802.11ac, it is still recent and may be changed, which may lead to errors or low correlation with reality. The propagation delay and loss models remain the same as in DARA. For traffic, User Datagram Protocol (UDP) is chosen because it is a simple protocol, and if packet tracing or labelling is needed, it proves to be more valuable than Transmission Control Protocol (TCP) since it splits packets into smaller packets and does not allow the mentioned tasks to be executed. The chosen UDP traffic data rate is 75 Mbit/s because it is supposed to saturate the link so that packets are always being sent regularly since the maximum data rate of MCSs is 54 Mbit/s. The packets sent in the traffic have a constant size (an arbitrary size of 1000 Bytes) to simplify the delay analysis since the delay already varies due to several factors, and the irregularity of the packet size would make its analysis even more difficult.

Parameter	**Description**
Wi-Fi Standard	IEEE 802.11n
Propagation Delay Model	Constant Speed
Propagation Loss Model	Friis
Traffic	UDP
Traffic Data Rate	75 Mbit/s
Packet Size	Constant (1000 Bytes)
Number Of Nodes	2
Mobility Model	Waypoint (10 meters to 1350 meters) / Constant (variable distance) / Random
Frame Aggregation	Disabled
Remote Station Manager	DRl (DARA or SLARA) / Minstrel / Ideal

Table 3.3: Ns-3 Environment Parameters.

The mobility model is situational and and it is possible to choose between three different models:

- The **Waypoint** model, as presented in Figure 3.5, consists of the transmitting node (designated "Node 1") starting from an initial position (position a)), with a distance D_i (10m) relative to the receiving node (designated "Node 0"), and reaching the final position (position b)) stipulated at the beginning of the simulation, with a distance D_f (1350m) relative to "Node 0", at the end of the simulation with duration of T seconds. The speed (v) associated to the movement of "Node 1" is given by $v = (D_f - D_i)/T$. This is an exploratory model, since "Node 1" during the simulation goes through every position between 10m and 1350m.

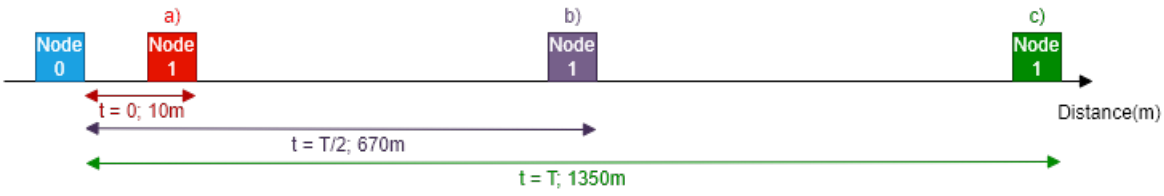


Figure 3.5: Waypoint mobility model.

- The **Constant** model, as presented in Figure 3.6, maintains "Node 1" in a fixed position, pre-defined at the start of simulation. This model is especially good for debugging and getting more detail for a specific distance.

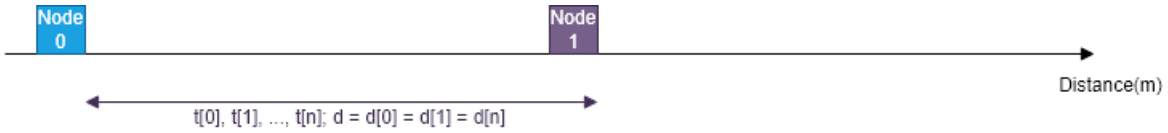


Figure 3.6: Constant mobility model.

- The **Random** model, as presented in Figure 3.7, makes "Node 1" teleport to random positions as can be seen in sequence a) (starting position), b) (position in the next time interval) and c) (position in the last time interval). This model can be used mainly to validate the solution since its teleports are random and decrease predictability.

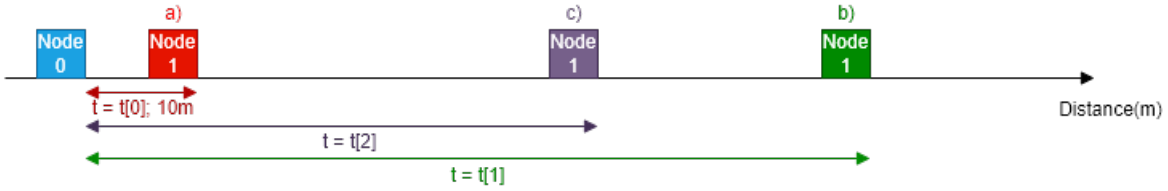


Figure 3.7: Random mobility model.

Frame aggregation is disabled for the same reasons as constant packet size because with frame aggregation, their sizes vary and, consequently, the delay, so it is disabled for simplification reasons. To the remote station manager was added a new feature has been added to the remote station manager to change the DRL reward, so it is possible to switch between DARA and SLARA rewards.

In addition, each time the simulation is launched, it is necessary to define a series of parameters that can change with each simulation. Some of these parameters are passed through the arguments of the command used to run the program (parameters that vary more regularly), and others are set manually in the algorithm code (parameters that are more regular and are changed casually). The most relevant parameters to launch a simulation are the following:

- **Port** — Designed to define the port to be used by a simulation. It is only possible to run one simulation per port. If it is intended to run several simulations in parallel, it is necessary to assign different port numbers for each simulation.

- **Mode** — Indicates the remote station manager to pick. It can be "drl" (DARA or SLARA are used), "min" (Minstrel algorithm is used) or "id" (Ideal algorithm, from ns-3, is used).

- **Simulation type** — Indicates the type of simulation. To launch an evaluation simulation of any algorithm, the simulation type "eval" is used. For a training session, either "train" or

"longtrain" are used depending on the type of training desired, and only the "drl" algorithm can be trained.

- **Simulation seed** — Indicates the seed for random generator.
- **Path** — Contains a specific file path for each simulation and it is used to save all logs generated during a simulation.
- **Duration** — Defines the simulation duration in seconds.
- **Distance** — Defines the initial position for the transmitter node. In the constant mobility model, the node maintains that distance throughout the simulation.
- **Enable Pcap** — Enables or disables the generation of pcap files used for debugging purposes.

3.4 Ns-3 gym

The ns-3 gym interface runs simultaneously with the ns-3 simulation and the DRL agent and performs an intermediary role. Initially, this interface is notified by the ns-3 simulation when it finishes executing a step (a 100ms interval). Then, ns-3 gym collects the state of the environment and calculates the **observation** and **reward** through the collected information and sends the state and reward to the DRL agent. Finally, the DRL agent chooses the **action** to execute for the next step and sends it to the ns-3 gym, which executes the action, and the ns-3 simulation starts the next step and repeats the process. In order to get the previously highlighted parameters, the following implementation was done:

- **Action** — as presented in Table 3.4, there are available eight possible actions (MCS_0 to MCS_7), corresponding for the MCS level for 800ns Guard Interval (GI), 20 MHz channel bandwidth and SISO IEEE 802.11n operation. The agent selects the new MCS for each time interval, and control and data mode are changed according to the new MCS index, as exposed in Listing 3.1.

```
bool MyGymEnv::ExecuteActions(Ptr<OpenGymDataContainer> action
    )
{
    Ptr<OpenGymDiscreteContainer> discrete = DynamicCast<
        OpenGymDiscreteContainer>(action);
    uint32_t dataRateId = discrete->GetValue();
    this->m_chosenAction = dataRateId;

    std::string dataRateStr = dataRates.at(dataRateId);
    if (DRLmode)
    {
```

```
            Config::Set("/NodeList/1/DeviceList/0/$ns3::
                WifiNetDevice/RemoteStationManager/$ns3::
                ConstantRateWifiManager/DataMode", StringValue(
                dataRateStr));
            Config::Set("/NodeList/1/DeviceList/0/$ns3::
                WifiNetDevice/RemoteStationManager/$ns3::
                ConstantRateWifiManager/ControlMode", StringValue(
                dataRateStr));
        }
        else
        {
            if (LOG_ENABLE)
                NS_LOG_UNCOND("IgnoringAction: " << action << " (
                    using Minstrel or Ideal)");
        }
        return true;
    }
```

Listing 3.1: "MyGymEnv::ExecuteActions function.

MCS Index	Modulation	Coding	20MHz			40MHz		
			Data Rate(Mbit/s)		Min. SNR	Data Rate(Mbit/s)		Min. SNR(dB)
			800ns GI	400ns GI		800ns GI	400ns GI	
0	BPSK	1/2	6.5	7.2	2	13.5	15	5
1	QPSK	1/2	13	14.4	5	27	30	8
2	QPSK	3/4	19.5	21.7	9	40.5	45	12
3	16-QAM	1/2	26	28.9	11	54	60	14
4	16-QAM	3/4	39	43.3	15	81	90	18
5	64-QAM	2/3	52	57.8	18	108	120	21
6	64-QAM	3/4	58.5	65	20	121.5	135	23
7	64-QAM	5/6	65	72.2	25	135	150	28

Table 3.4: MCS Table for SISO IEEE 802.11n, from [2].

- **Observation** — as exposed in Listing 3.2, the number of received packets and their SNR are tracked. The average SNR is calculated for current time interval. When the MCS selected for a given time interval is too high for the SNR of that interval, packets will not be sent successfully, and there is no observation to send to the agent due to the SNR calculation depending on the received packets. In order to solve this problem, it is assumed that SNR is zero for that time interval to force the agent to pick the lowest MCS for the next interval.

```
Ptr<OpenGymDataContainer> MyGymEnv::GetObservation()
{
    uint32_t nodeNum = 1;
    float avgSNR;

    std::vector<uint32_t> shape = {
```

```
        nodeNum,
    };
    Ptr<OpenGymBoxContainer<float>> box = CreateObject<
        OpenGymBoxContainer<float>>(shape);
    double snr;
    if (this->nSuccess == 0)
    {
        snr = 0;

        if (!std::isfinite(snr))
        {
            NS_LOG_UNCOND("snr: " << snr);
            snr = 0;
        }
    }
    else
    {
        avgSNR = (this->sumSNR / this->nPacketsRX);
        snr = 10 * log10(avgSNR);
        snr = round(snr) / 100;
    }
    box->AddValue(snr);
    this->sumSNR = 0;
    this->nPacketsRX = 0;

    if (LOG_ENABLE)
        NS_LOG_UNCOND("Observations: " << box);

    return box;
}
```

Listing 3.2: "MyGymEnv::GetObservation function.

- **Reward** — the reward calculation depends on whether a transmitted packet was ACKed or not. In order to verify this, two trace sources were implemented, from Listing 3.3, to trigger when a packet is successfully received ("AckedMpdu" trace source) and when a packet is not received ("NAckedMpdu" trace source) and the transmitting node has this information. When "AckedMpdu" trace source triggers, calls function "MyGymEnv::AckOk", from Listing 3.4, which updates the number of received packets and the number of transmission attempts. When "NAckedMpdu" trace source triggers, calls function "MyGymEnv::AckNeg" which updates the number of attempts. If neither of previous trace sources trigger during the timeout, here is no information and the number of attempts is updated. There is no information, and the number of attempts is updated. The FER is calculated using the values

obtained from the counting done in these functions, the reward is calculated with that result, and the normalised delay as presented in Section 3.2.

```
Config::Connect("/NodeList/1/DeviceList/0/$ns3::
    WifiNetDevice/Mac/$ns3::RegularWifiMac/$ns3::
    AdhocWifiMac/AckedMpdu", MakeCallback(&MyGymEnv::AckOk,
     this));
Config::Connect("/NodeList/1/DeviceList/0/$ns3::
    WifiNetDevice/Mac/$ns3::RegularWifiMac/$ns3::
    AdhocWifiMac/NAckedMpdu", MakeCallback(&MyGymEnv::
    AckNeg, this));
```

Listing 3.3: Acknowledgment trace sources.

```
void MyGymEnv::AckOk(std::string context, Ptr<const
    WifiMacQueueItem> mpdu)
{
    this->dataRate = getDataRate(mpdu, this->WRSM);
    this->sumDataRate += this->dataRate;

    this->nSuccess++;
    this->nAttempts++;
}

void MyGymEnv::AckNeg(std::string context, Ptr<const
    WifiMacQueueItem> mpdu)
{
    this->nAttempts++;
}

void MyGymEnv::MpduTimeout(std::string context, uint8_t reason
    ,
                           Ptr<const WifiMacQueueItem> mpdu,
                           const WifiTxVector &txVector)
{
    this->nAttempts++;
}
void MyGymEnv::PsduTimeout(std::string context, uint8_t reason
    ,
                           Ptr<const WifiPsdu> psdu,
                           const WifiTxVector &txVector)
{
    this->nAttempts++;
```

```
}
```

Listing 3.4: ACK and Block NACK functions.

In addition, it was necessary to add a mechanism to DARA, similar to the one already used for throughput, to log the delay of each packet. This mechanism was useful for the preliminary study and it was crucial for the reward calculation. To implement this mechanism, we had to consult the ns-3.35 documentation and select two trace sources (one to indicate that a packet was transmitted and the other to indicate that a packet was received) to calculate the time difference between the trigger of each trace and associate that time interval to the delay of the respective packet. In Listing 3.5, the first configuration is the **PhyTxBegin** trace from the "ns3::WifiPhy" library of ns-3 version 3.35, and its function is to trigger whenever a packet has begun transmitting to the channel medium in the transmitter node (node associated with number 1). The second configuration is the **MacRx** trace from the "ns3::WifiMac" library and triggers when a packet has been received by the receiver node (node associated with number 0), has been through the physical layers and is about to be forwarded up the local protocol stack.

```
Config::ConnectWithoutContext("/NodeList/1/DeviceList/*/$ns3::
    WifiNetDevice/Phy/PhyTxBegin", MakeCallback(&MyGymEnv::TxTime,
    this));
Config::ConnectWithoutContext("/NodeList/0/DeviceList/*/$ns3::
    WifiNetDevice/Mac/MacRx", MakeCallback(&MyGymEnv::RxTime, this)
    );
```

Listing 3.5: Trace sources configuration

Whenever a trace source triggers, it makes a callback to its respective function and executes the code inside it. In Listing 3.5, the trace source **PhyTxBegin** calls function "MyGymEnv::TxTime" and the trace source **MacRx** calls function "MyGymEnv::RxTime" when triggered. The "MyGymEnv::TxTime" function is presented in Listing 3.6 and was created mainly to store the timestamp of a packet that is transmitted to the channel medium. While developing this function, it was necessary to use the pointer to the packet being transmitted (passed as a parameter of the function and the only parameter to be used) to get its unique id and a map to store the packet timestamp. Next, it was necessary to check if the packet already existed on the map. If not, the timestamp is saved in the map with the respective unique id and the same unique id is saved in another map (to save the number of retransmissions for each packet) with zero retransmissions. If it exists, it is a retransmission and increases the number of retransmissions to the respective packet.

```
void MyGymEnv::TxTime(Ptr<const Packet> packet, double txPowerW)
{   if (this->packetMap.find (packet->GetUid())->second == Seconds
    (0)){
        this->packetMap[packet->GetUid()] = Now();
        this->packetRetrans[packet->GetUid()] = 0;
```

```
    }else if(this->packetMap.find (packet->GetUid())->second !=
        Seconds(0)){

        this->packetRetrans[packet->GetUid()]++;
    }
}
```

Listing 3.6: "MyGymEnv::TxTime" function

Listing 3.7 presents the "MyGymEnv::RxTime". This function verifies if the received packet exists in the map and if its timestamp is non-zero. Next, it calculates the delay of the packet by taking the difference between the current timestamp and the one stored in the map, stores the number of retransmissions, deletes the packet information from the timestamps and retransmissions maps, adds the obtained delay value to a list that is used later in the work for statistical purposes and updates the number of received packets. Finally, the simulation time, delay, number of retransmissions, data rate and distance associated with the packet are logged into a CSV file.

```
void MyGymEnv::RxTime(Ptr<const Packet> packet)
{
    double now = Simulator::Now().GetSeconds();
    uint64_t nRT;
    this->nodePosition = this->NodeModel->GetPosition();
    if (this->packetMap.find (packet->GetUid())->second != Seconds
        (0))
        {
            float delay = (Now() - this->packetMap.find(packet->
                GetUid ())->second).GetDouble()/1000000;
            nRT = this->packetRetrans.find(packet->GetUid ())->
                second;

            this->packetMap.erase(packet->GetUid());
            this->packetRetrans.erase(packet->GetUid());
            this->intervalValues.push_back(delay);
            this->RxCount++;

            this->oflog << now << ";" << delay << ";" << nRT << ";"
                 << this->dataRate/1000000 << ";" << this->
                nodePosition.x << std::endl;
        }
}
```

Listing 3.7: "MyGymEnv::RxTime" function

3.5 SLARA Agent Architecture

The SLARA agent, as well as the DARA agent, is implemented in Python and uses the TF-Agents library. The agent function 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 agent implementation is similar to DARA agent, it uses DQN learning algorithm with the following parameters:

- **Observation space** — float value between 0.0 and 1.0. The SNR value is scaled by dividing the SNR value in dB by 100 to improve the learning process.
- **Action space** — integer value representing the MCS index used to transmit packets for the next time interval.
- **Optimiser** — Adam optimiser [18] with a default learning rate of 10^{-3}.
- **Epsilon greedy** — value between 0 and 1 that determines the percentage of random actions the agent performs. This value starts at 1 (every action is random) and, through the established **epsilon decay period** ($10 \times T$, where T is the simulation duration), decreases over the simulation until 0.1 (10% of actions are random). The epsilon decay period value is chosen to ensure that at the end of the simulation, all options have been properly explored and that the agent, when tested, will choose the optimal actions.
- **Q-Network** — 2 layers of interconnected parameter with 32 units each.
- **Replay Buffer** — size of 10^6 trajectories. During the simulation, trajectories are appended to the replay buffer. Samples randomly a batch of 64 trajectories from the replay buffer every time the agent is trained.

Chapter 4

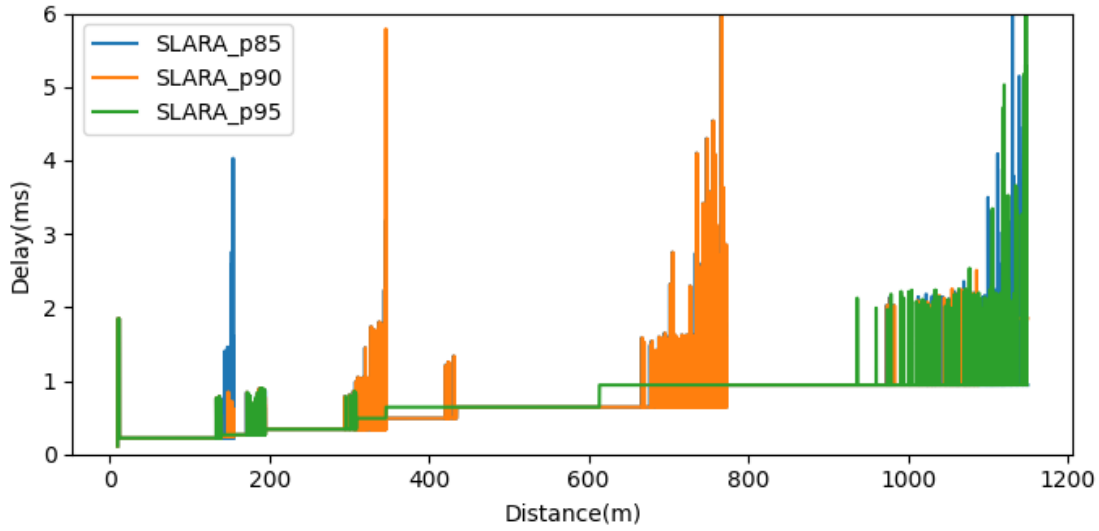
Evaluation Results

This chapter presents the results of the final solution. This version of SLARA was trained with a duration (T) of 6000 seconds with the Waypoint mobility model, as presented in Figure 3.5. A learning rate of 1×10^{-3} was chosen with an epsilon decay period of $10 \times T$. The results of this SLARA version are compared with three algorithms. First is Minstrel because it is a popular algorithm in wireless networks, and it is essential to evaluate the performance against a widely used algorithm. Next is Ideal because it is an ns-3 algorithm and tries to implement an ideal rate control. Lastly, DARA, because SLARA is an adaptation of this algorithm and it is important to evaluate the difference between the two algorithms caused by the adaptation.

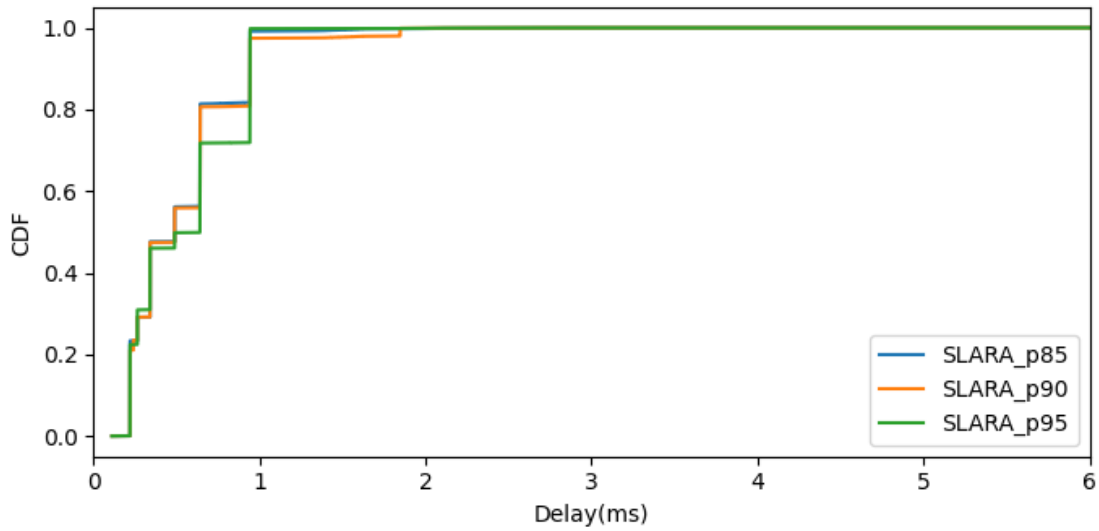
4.1 SLARA reward variation analysis

In this section, the results for the 85th, 90th and 95th percentiles are shown. In all sections, the distance range is between 10m and 1150m due to problems in training DARA for distances over 1150m. The distance limitation is done to make the evaluation of all algorithms fairer.

In Figure 4.1a is plotted the delay for every packet throughout a 60 seconds simulation from an initial distance of 10m to a distance of 1150m. It is visible that the behavior of 85th and 90th percentiles are similar and the 95th percentile tends to be more conservative when changing MCS because until around a distance of 950m the existence of delay peaks in the 95th percentile are residual due to the early MCS change. Besides that, for the 95th percentile it earlier MCS changes are observed compared with the other two percentiles, as observed at 350m and 600m. That behaviour was expected by higher percentiles due to the added functionality in Equation 3.3, which penalises drastically, assigning a null reward if the delay corresponding to the selected percentile is higher than 2ms, leading to the fact that the higher the percentile is, the higher the delay associated to it will be and the more the algorithm tends to avoid these values and be more conservative. In Figure 4.1b is plotted the Cumulative Distribution Function (CDF) plot from the previous figure results. It is observed that even with several delay peaks, the 85th and 90th percentiles in general, have lower delays than the 95th percentile and both tend to perform better than the 95th percentile.



(a) SLARA plot for all packets delay.

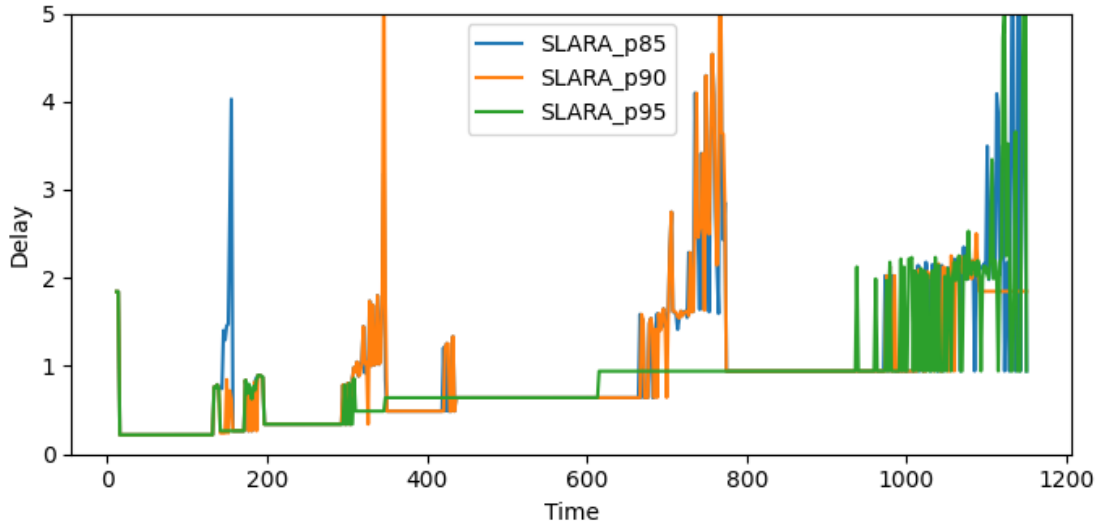


(b) SLARA CDF plot for all packets delay.

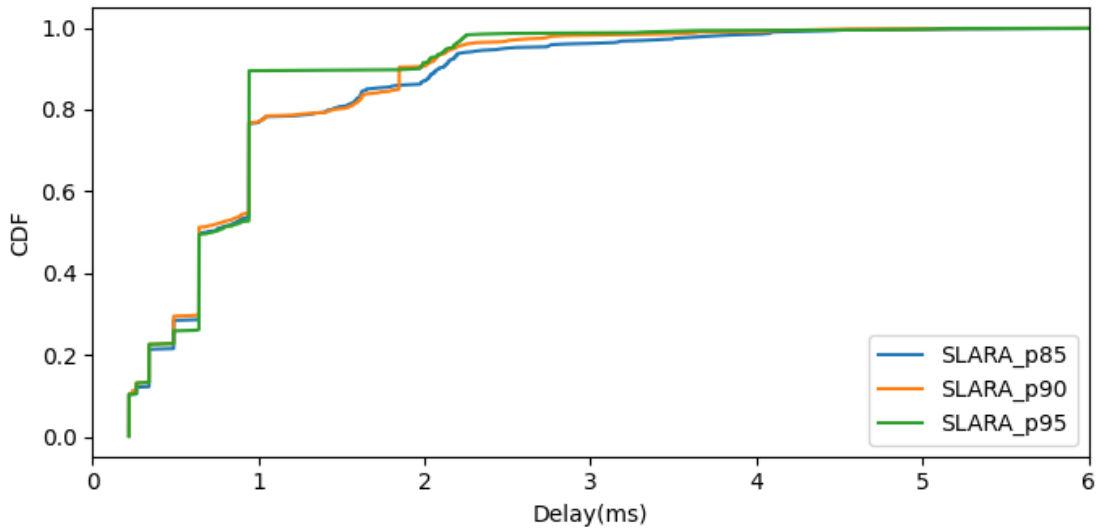
Figure 4.1: SLARA results for all packets delay.

In Figure 4.2a the max delay is plotted for every 100ms interval throughout the simulation previously mentioned. This graph is similar to the previously analysed but further reinforces each percentile's peaks and maximum delay. As observed in Figure 4.2b the 95th percentile max delay values are lower than the others percentiles. Since the designed reward tends to optimise the selected percentile, the values above that percentile are not taken into consideration when training the algorithm. Then, a higher percentile leads to better optimised max delay values with less delay peaks and more linear delay values. In contrast, lower percentiles present generally lower delays but with the cost of having high delay peaks leading to higher max values.

In Figure 4.3a the throughput is plotted for every 100ms throughout the previous simulation is.



(a) SLARA plot for max delay for every 100ms interval.

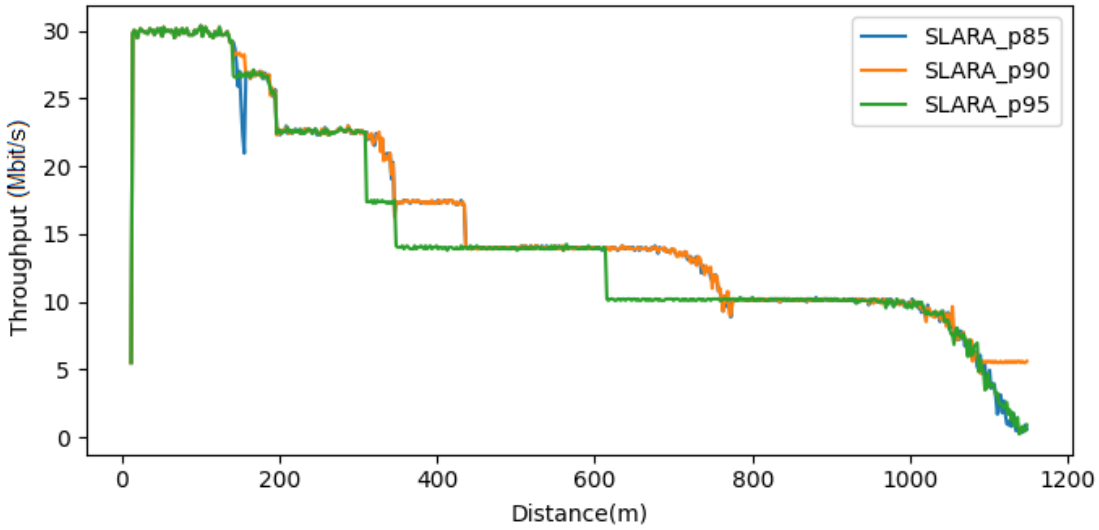


(b) SLARA CDF plot for max delay for every 100ms interval.

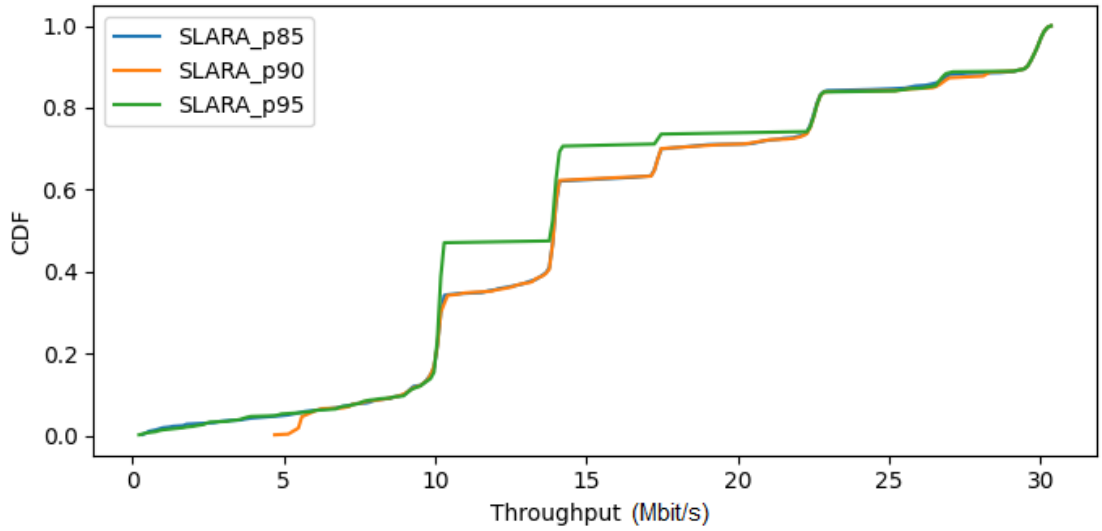
Figure 4.2: SLARA results for max delay for every 100ms interval.

As expected, the 95th percentile is conservative, it lowers the MCS earlier than other percentiles, as observed at around 150m, 300m, 350m and 600m, and falls at around 1150m distance. The 85th and 90th percentiles have similar behaviour, but the 85th percentile seems greedier than the 90th percentile since, at around 150m, the 85th percentile maintains the MCS, and throughput lowers drastically due to bad link quality, and the 90th lowers the MCS. Near the 1150m distance, the 85th percentile throughput falls to zero, and the 90th maintains the throughput at 5 Mbit/s. In Figure 4.3b the CDF plot proves the previous observations since the 85th and 90th percentiles have higher throughput than the 95th percentile most of the time, and the 90th percentile never assumes a throughput of 0 Mbit/s. That leads to the conclusion that the 90th percentile is solid

compared to other RAAs.



(a) SLARA plot for throughput for every 100ms interval.



(b) SLARA CDF plot throughput for every 100ms interval.

Figure 4.3: SLARA results for throughput for every 100ms interval.

4.2 SLARA vs RAAs

In this section, a comparison is made with some algorithms mentioned in state of the art with DARA. The purpose is to compare the results of SLARA with these algorithms and verify improvements, failures and what to improve to obtain better performance with SLARA. The simulations were performed only with the waypoint mobility model since, in the current version of SLARA, there is no functionality to obtain backup data rates. For more unpredictable mobility

scenarios (such as the random mobility model), SLARA would be at a disadvantage compared to some RAAs.

4.2.1 SLARA vs DARA

In Figure 4.4a it is verified that SLARA has lower delay peaks than DARA, especially in longer distances, and near the 400m distance SLARA lowers the MCS earlier than DARA, which result in lower packet delay than DARA until DARA lowers the MCS. This is expected because DARA tends to be greedier since it aims to maintain the higher throughput possible instead of directly being trained to maintain the lowest delay possible, but generally DARA delay for all packets is lower than SLARA. Nevertheless, the behaviour of both algorithms is similar as Figure 4.4b proves, by showing the curves closer to each other.

In Figure 4.5a, the peaks in the delay of DARA are evident when compared to the peaks of SLARA, especially after the 400m distance where SLARA has no delay peaks compared to DARA when lowering the MCS. This difference is essentially due to DARA's preference for higher data rates but also to the percentile used by SLARA, the 90th percentile, which, being high, penalises very high peaks by making MCS choices that soften these peaks, making the maximum values at each 100ms more linear. Figure 4.5b reinforces the previous conclusion since the SLARA curve is above the DARA curve, which means that generally, the max delay for every 100ms interval in SLARA is lower than the DARA max delay for every 100ms interval.

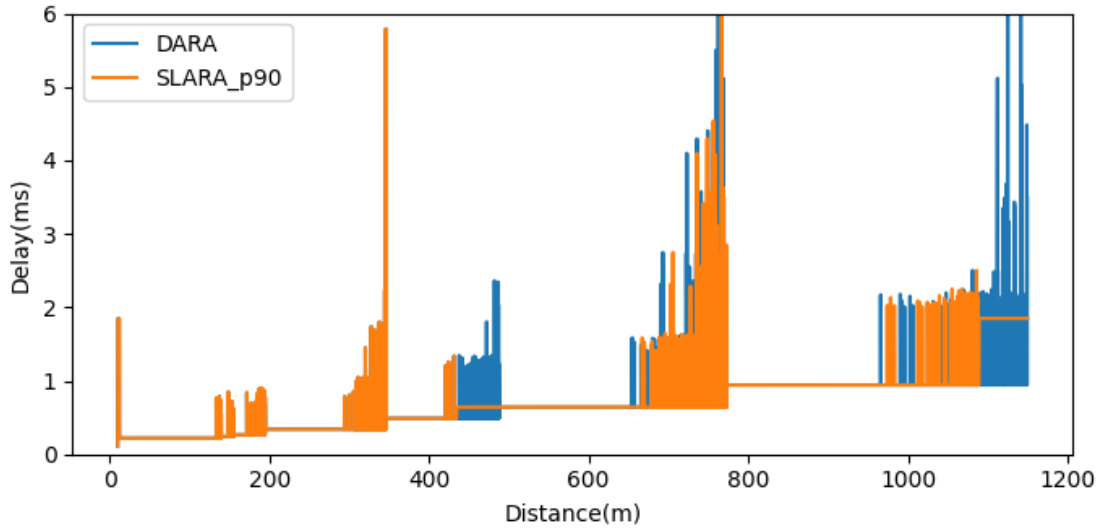
In the throughput analysis of both algorithms in Figures 4.6a and 4.6b, it is visible that DARA makes choices that benefit throughput, although the difference between the two is slight, which is caused by a more conservative choice by SLARA after the 400m distance to avoid delay peaks caused by packet retransmissions due to poor link quality.

In this comparison between DARA and SLARA, it was found that for the chosen percentile, the behaviour of SLARA is very close to DARA, optimising mainly the maximum delay for each 100ms interval at the cost of changing earlier to a lower MCS to avoid packet retransmissions to lower delay peaks.

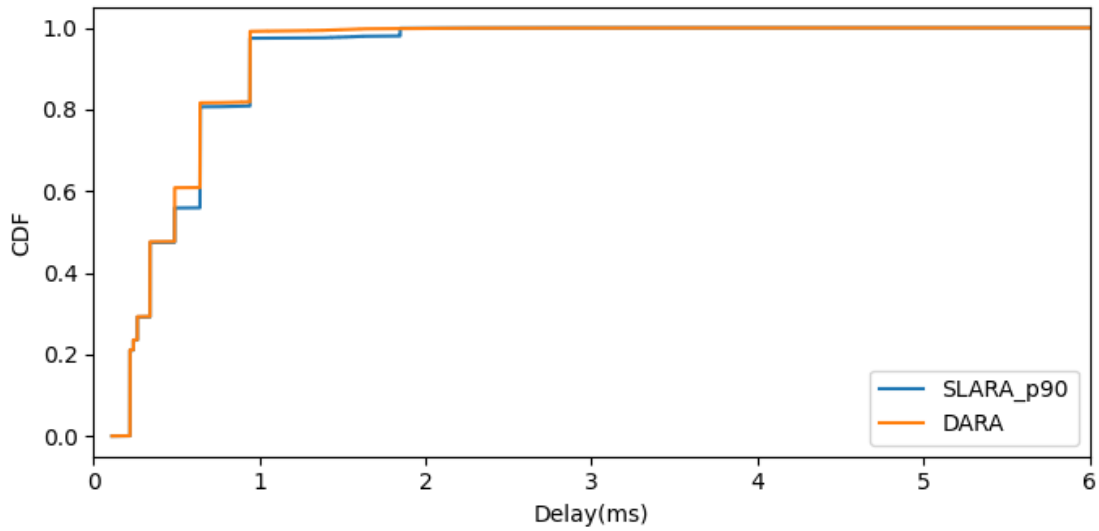
4.2.2 SLARA vs Minstrel vs Ideal

Figure 4.7a shows that Minstrel shows high delay values with many peaks. In contrast, Ideal presents more linear values with few delay peaks and changes to a lower MCS earlier than Minstrel and SLARA, being the more conservative algorithm. SLARA, even though it has more delay peaks than Ideal, changes to a lower MCS earlier than Minstrel and Ideal after the 400m distance and appears to have a lower delay at the overall level, having better performance than these two algorithms. The same is confirmed with the curves observed in Figure 4.7b, where it can be seen that SLARA's curve is above the curve of the other two algorithms.

Figure 4.8a shows the massive discrepancy between the maximum delay value at each 100ms interval of Minstrel and those of SLARA and Ideal (which present linear values with few delay peaks). There are also some points where SLARA delay is slightly higher than Ideal's, observed



(a) SLARA vs DARA plot for all packets delay.

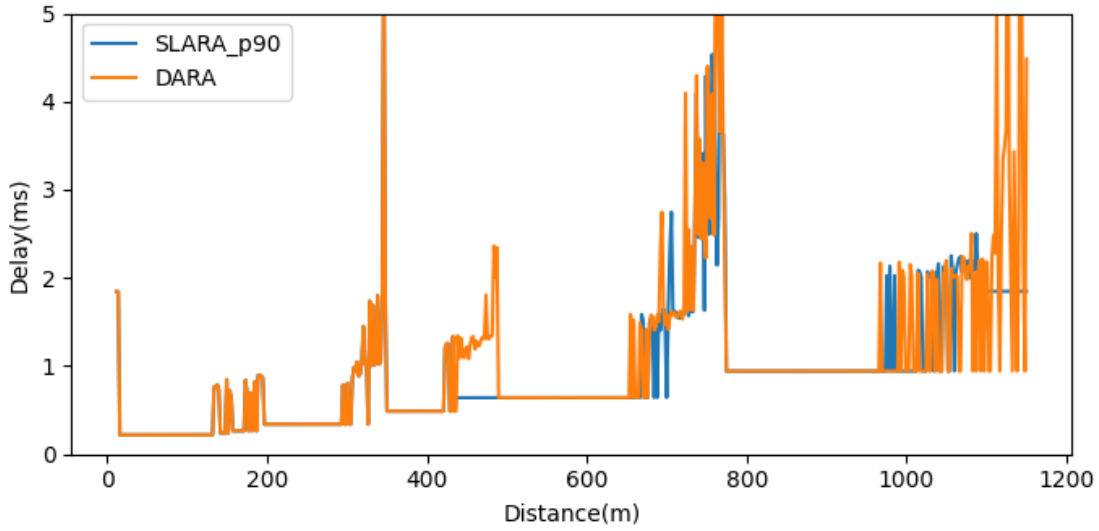


(b) SLARA vs DARA CDF plot for all packets delay.

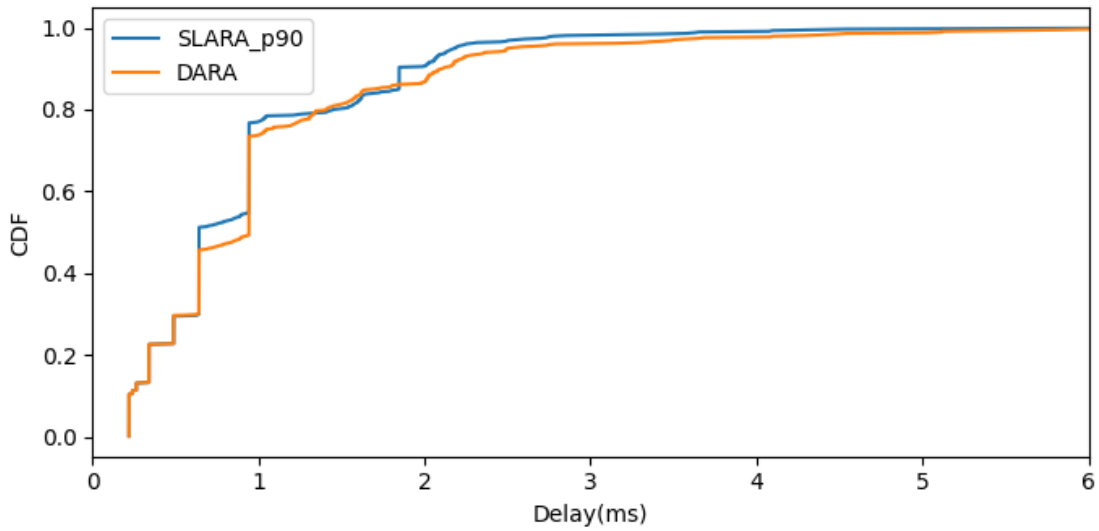
Figure 4.4: SLARA vs DARA results for all packets delay.

from 700m and 1000m, due to earlier MCS change by Ideal except after 400m distance. This proximity of SLARA to Ideal, as also observed in the curves of Figure 4.8b, adds value to the SLARA results since Ideal is an ns-3 algorithm with the purpose of reproducing an "Ideal" rate control and aims to maximise the physical rate since the frame success ratio is intended to be above a fixed threshold.

Although SLARA is not designed with the goal of throughput optimisation, in Figure 4.9a it is visible that the choices made by SLARA lead to better throughput values than Minstrel. When compared to Ideal, the throughput values are similar, and at specific distances, both Ideal and SLARA make more conservative decisions when compared to each other, such as around



(a) SLARA vs DARA plot for max delay for every 100ms interval.



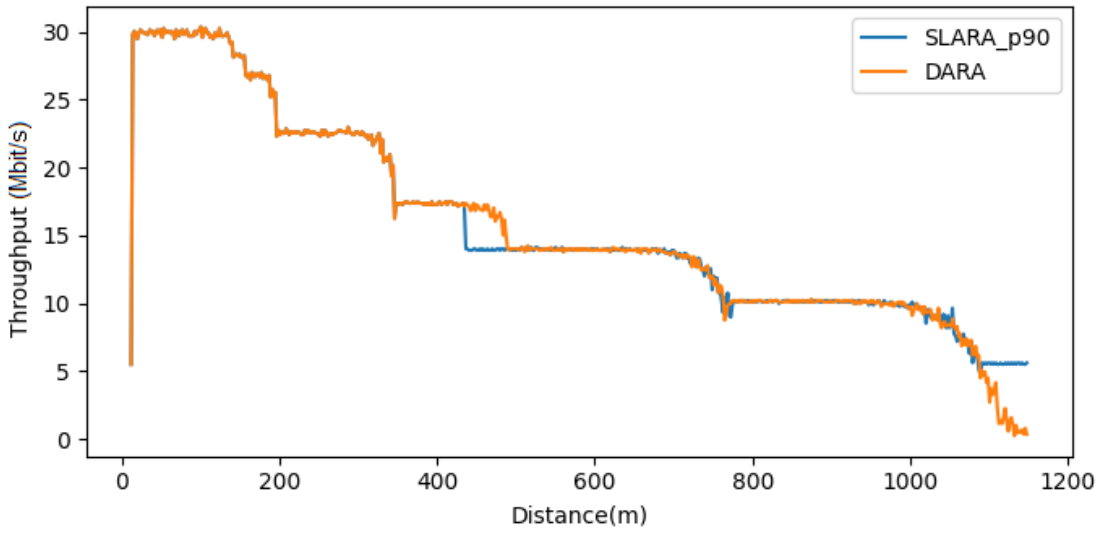
(b) SLARA vs DARA CDF plot for max delay for every 100ms interval.

Figure 4.5: SLARA vs DARA results for max delay for every 100ms interval.

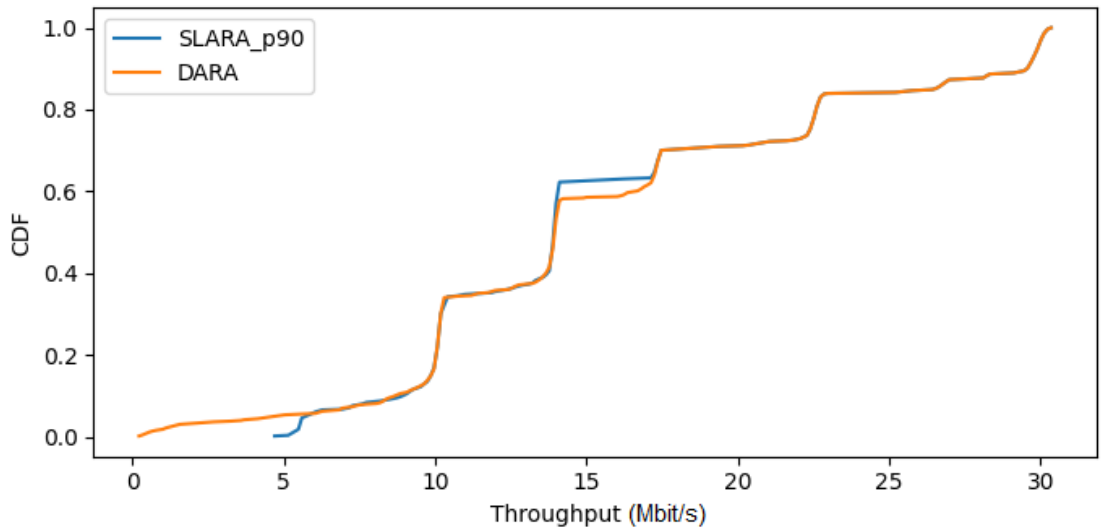
450m where SLARA switches MCS earlier and around 1000m where Ideal switches MCS before SLARA. Figure 4.9b shows that SLARA's curve is relatively lower than Ideal's, indicating that SLARA's throughput values are higher than Ideal's.

4.3 Conclusions

As observed previously, the 90th percentile reward configuration in SLARA proves to be more balanced concerning the percentiles with which it was compared. Given the complexity of identifying the optimal percentile for the reward configuration, it is possible to conclude that there may



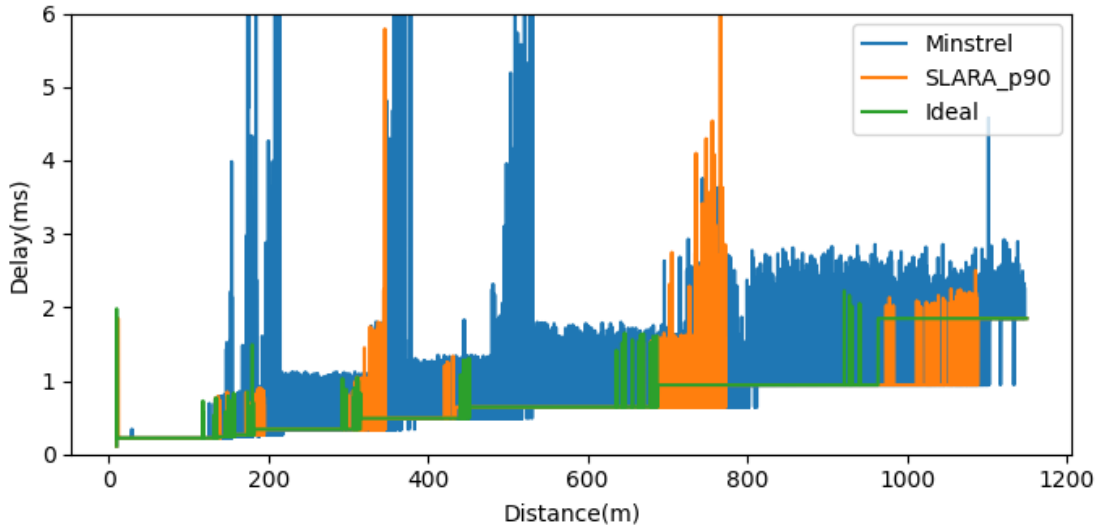
(a) SLARA vs DARA plot for throughput for every 100ms interval.



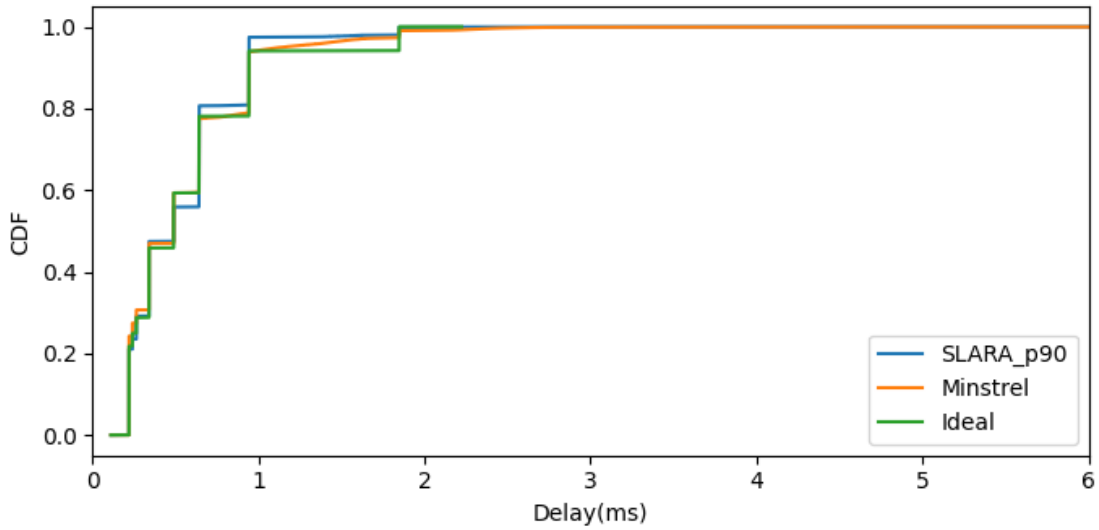
(b) SLARA vs DARA CDF plot throughput for every 100ms interval.

Figure 4.6: SLARA vs DARA results for throughput for every 100ms interval.

be a more suitable percentile that can obtain better results compared to those currently obtained. Nevertheless, with the reward configuration, it was possible to obtain improvements relative to DARA, although, in general, the behaviour of both algorithms is similar. SLARA led to the linearisation of the maximum delay and the reduction of the delay peaks, leading to greater reliability in the delay of this algorithm, as intended. Although the comparison was made through simulations, SLARA demonstrated better performance relative to Minstrel with different behaviour in MCS selection, leading to a lower delay and a higher throughput.

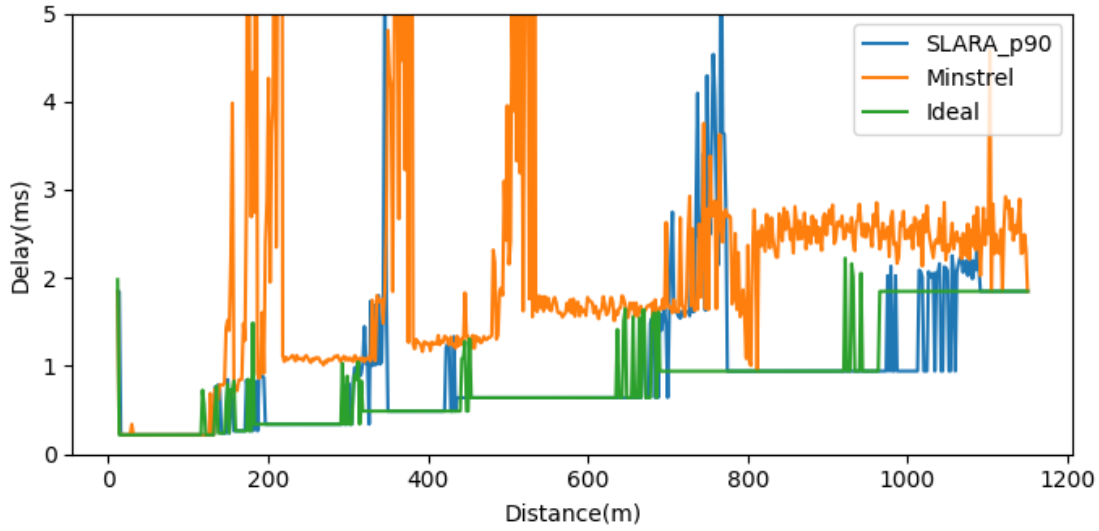


(a) RAAs plot for all packets delay.

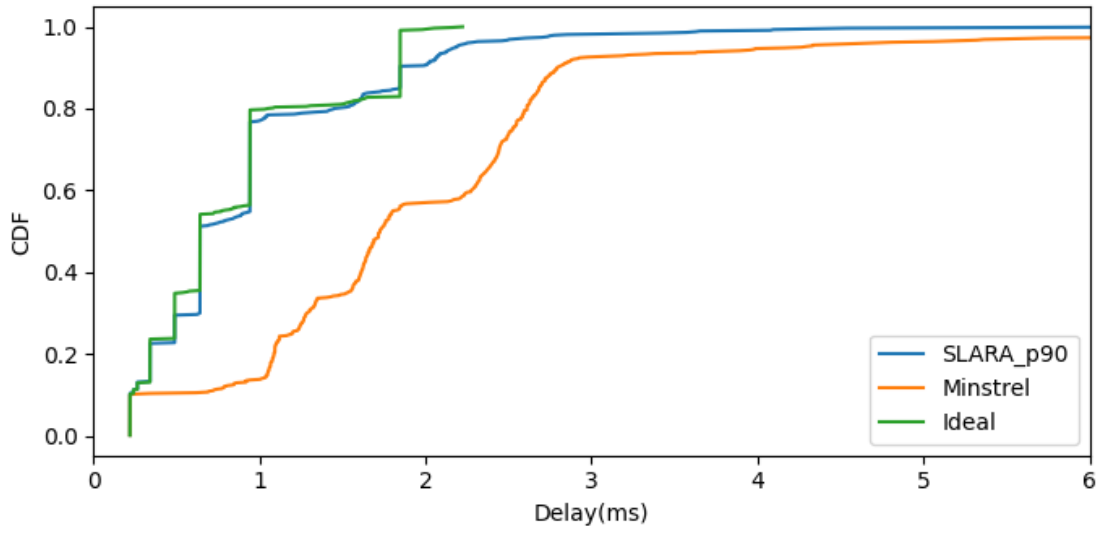


(b) RAAs CDF plot for all packets delay.

Figure 4.7: RAAs results for all packets delay.

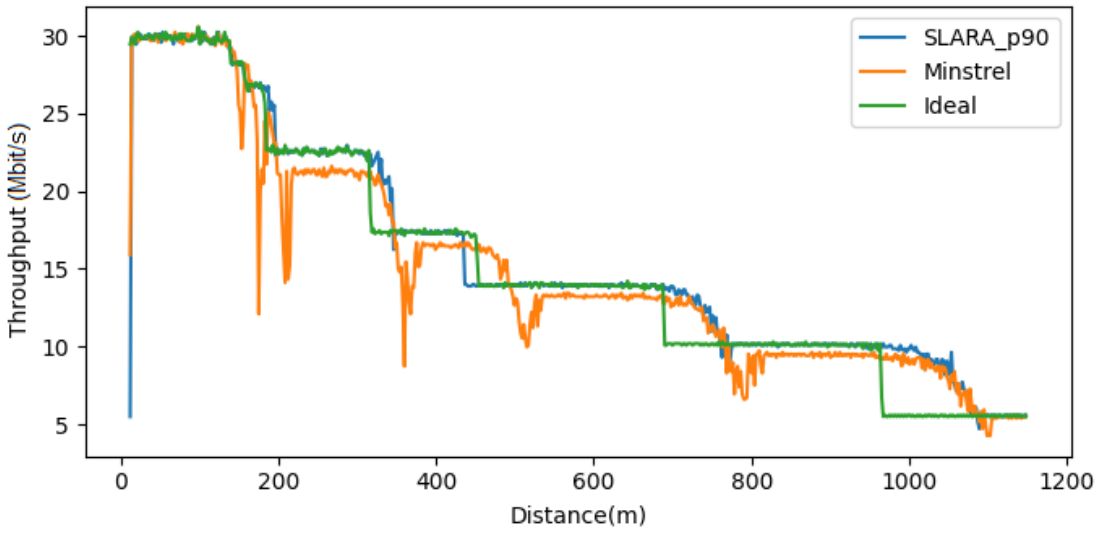


(a) RAAs plot for max delay for every 100ms interval.

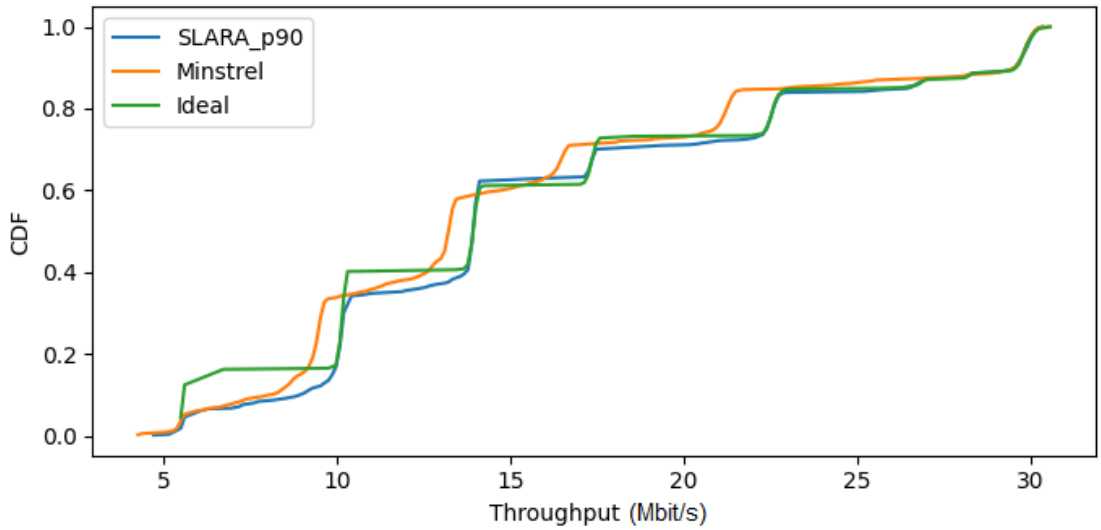


(b) RAAs CDF plot for max delay for every 100ms interval.

Figure 4.8: RAAs results for max delay for every 100ms interval.



(a) RAAs plot for throughput for every 100ms interval.



(b) RAAs CDF plot throughput for every 100ms interval.

Figure 4.9: RAAs results for throughput for every 100ms interval.

Chapter 5

Conclusion and Future Work

With the work developed throughout this dissertation, it was possible to demonstrate that MAC transmission delay is a metric that can be considered when developing RAAs. Although SLARA is a DARA adaptation and in the comparison between both, the differences are reduced with a slight advantage in the global delay for DARA, SLARA with the 90th percentile reward configuration allowed a reduction of delay peaks and the linearisation of the maximum delay, allowing greater reliability in the link, at the cost of a slight disadvantage in throughput compared to DARA. When compared to Minstrel, SLARA demonstrated great results relative to a popular RAA, although they were compared through simulation, assuming lower delay and higher throughput values than Minstrel, closely approximating the "ideal" rate control of Ideal. Even with some positive results, there is room to optimise SLARA. The 90th percentile choice was based on some tests that were not representative of all percentiles due to the time required to have that representativeness. That said, other percentiles or metrics related to the time interval delay may better suit this algorithm and obtain a lower delay than that of the algorithms compared to SLARA. Therefore, the reward function was developed to have an input parameter to make the normalisation of the representative delay of a time interval flexible.

At the beginning of this work, several doubts and difficulties arose. There was great difficulty in finding related work that optimises the delay with RAAs. It was verified that it is a gap. The existence of related algorithms could contribute to the development of new methods or even improvement of some already used, but meanwhile, it was necessary to develop a strategy from scratch. The initial goal was to normalise delay so that the reward was between 0 and 1 to increase the scalability of its training. For this, conducting a thorough preliminary study was necessary to understand the behaviour of the delay in the various algorithms and the delays associated with each MCS. These delays were useful to later use the lowest delay for delay normalisation. Although one problem was solved, another problem arose, which was to find a delay value that reflected the quality in choosing an MCS for a given time interval. As already mentioned, the temporary solution, that is subject to be improved, was to choose the 90th percentile, which represents the limit between linear and peak delay values.

In the future, there is much that can be improved and added to SLARA. As already mentioned,

it will be essential to optimise the value reflecting the choice of an MCS for a time interval. Since the conditions of the simulation scenario have been simplified for the first version by disabling the frame aggregation, the next step will be to optimise the delay with active frame aggregation, evolving the algorithm to a link adaptation algorithm. In addition, it would also be important to add a feature to the algorithm to perform micro actions within the 100ms intervals so that it is possible to switch to a backup data rate. If there is a high variation in the link quality, the algorithm could adapt quickly without waiting for the next interval to select another MCS. With this functionality, it would be interesting to use the random mobility model to test its performance of this functionality.

References

[1] Machine Learning Quantum: machine learning & analytics. https://ml2quantum.com/machine-learning/. Accessed: 10-02-2022.

[2] Mcs table and how to use it | wireless lan professionals. https://wlanprofessionals.com/mcs-table-and-how-to-use-it/. Accessed: 05-08-2022.

[3] Szymon Szott, Katarzyna Kosek-Szott, Piotr Gawłowicz, Jorge Torres Gómez, Boris Bellalta, Anatolij Zubow, and Falko Dressler. Wifi meets ml: A survey on improving ieee 802.11 performance with machine learning. *arXiv preprint arXiv:2109.04786*, 2021.

[4] Chi-Yu Li, Chunyi Peng, Songwu Lu, Xinbing Wang, and Ranveer Chandra. Latency-aware rate adaptation in 802.11n home networks. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 1293–1301, 2015. doi:10.1109/INFOCOM.2015.7218505.

[5] 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.

[6] Ibrahim Sammour and Gerard Chalhoub. Evaluation of rate adaptation algorithms in ieee 802.11 networks. *Electronics*, 9(9):1436, 2020.

[7] 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.

[8] Chi-Yu Li, Syuan-Cheng Chen, Chien-Ting Kuo, and Chui-Hao Chiu. Practical machine learning-based rate adaptation solution for wi-fi nics: Ieee 802.11ac as a case study. *IEEE Transactions on Vehicular Technology*, 69(9):10264–10277, 2020. doi:10.1109/TVT.2020.3004471.

[9] 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.

[10] Ruben Queiros, Eduardo Almeida, Helder Fontes, Jose Ruela, and Rui Campos. Wi-fi rate adaptation using a simple deep reinforcement learning approach. 2 2022. URL: http://arxiv.org/abs/2202.03997.

[11] ns-3 home page. https://www.nsnam.org/. Accessed: 02-01-2022.

[12] Vitor Lamela, Helder Fontes, Tiago Oliveira, Jose Ruela, Manuel Ricardo, and Rui Campos. Repeatable and reproducible wireless networking experimentation through trace-based simulation. In *2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 53–58. IEEE, 2019.

[13] Piotr Gawłowicz and Anatolij Zubow. Ns-3 meets openai gym: The playground for machine learning in networking research. In *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 113–120, 2019.

[14] TensorFlow Agents. https://www.tensorflow.org/agents. Accessed: 26-07-2022.

[15] 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.

[16] Giovanni Peserico, Tommaso Fedullo, Alberto Morato, Stefano Vitturi, and Federico Tramarin. Rate adaptation by reinforcement learning for wi-fi industrial networks. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 1139–1142, 2020. doi:10.1109/ETFA46521.2020.9212060.

[17] Wenchao Xu, Song Guo, Shiheng Ma, Haibo Zhou, Mingli Wu, and Weihua Zhuang. Augmenting drive-thru internet via reinforcement learning-based rate adaptation. *IEEE Internet of Things Journal*, 7(4):3114–3123, 2020. doi:10.1109/JIOT.2020.2965148.

[18] Tensorflow adam optimizer. https://www.tensorflow.org/api_docs/python/tf/compat/v1/train/AdamOptimizer. Accessed: 28-07-2022.