AUTOMATIC WLAN CONTROL USING REINFORCEMENT LEARNING FOR IMPROVED QUALITY OF EXPERIENCE

HENRIQUE DUARTE MOURA

AUTOMATIC WLAN CONTROL USING REINFORCEMENT LEARNING FOR IMPROVED QUALITY OF EXPERIENCE

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

Orientador: Daniel Fernandes Macedo
Coorientador: Marcos Augusto Menezes Vieira

Belo Horizonte - MG

Dezembro de 2019

HENRIQUE DUARTE MOURA

AUTOMATIC WLAN CONTROL USING REINFORCEMENT LEARNING FOR IMPROVED QUALITY OF EXPERIENCE

Thesis presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

Advisor: Daniel Fernandes Macedo
Co-Advisor: Marcos Augusto Menezes Vieira

Belo Horizonte - MG

December 2019

Duarte Moura, Henrique

M929a Automatic WLAN control using reinforcement learning for improved quality of experience / Henrique Duarte Moura. — Belo Horizonte - MG, 2019
xxiii, 176 f. : il. ; 29cm

Tese (doutorado) — Universidade Federal de Minas Gerais - Departamento de Ciência da Computação
Orientador: Daniel Fernandes Macedo

1. Computação - Teses. 2. Aprendizado de máquina - Teses. 3. Sistema de comunicação sem fio - Teses. 4. Redes de computador - Teses. I. Orientador. II. Coorientador. III. Título.

CDU 519.6*82.10(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

AUTOMATIC WLAN CONTROL USING REINFORCEMENT LEARNING FOR IMPROVED QUALITY OF EXPERIENCE

HENRIQUE DUARTE MOURA

Tese defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. DANIEL FERNANDES MACEDO - Orientador
Departamento de Ciência da Computação - UFMG

PROF. MARCOS AUGUSTO MENEZES VIEIRA - Coorientador
Departamento de Ciência da Computação - UFMG

PROF. ADRIANO ALONSO VELOSO
Departamento de Ciência da Computação - UFMG

PROF. STEVEN LATRÉ
IDLab - University of Antwerp

PROF. ALEX BORGES VIEIRA
Departamento de Ciência da Computação - UFJF

PROF. JOSÉ MARCOS SILVA NOGUEIRA
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 20 de Dezembro de 2019.

Acknowledgments

To my advisors Daniel and Marcos, for the invaluable assistance, guidance, and making me a researcher.
To my beloved Ana Paula and my family, for everything else.
To the WINET lab colleagues, for the support, exchange of ideas, and comradeship.
To the friends, that endured all my absence.
To the PPGCC's professors, for the knowledge, and orientation.
To the DCC's staff, for always solving my problems, and for the extra smile.
To the funding agencies of the research project - CAPES and CNPq, which enabled this study.

"In God we trust, all others must bring data."
(Brian L. Joiner, 1985)

Abstract

Real-time performance demands, such as augmented reality, and high quality video streaming brought new challenges to network control, as users expect high levels of quality to be upheld. Improving the user satisfaction and minimizing customer turnover while still maintaining their competitive advantage can be a significant challenge for service providers. quality of experience (QoE) is hard to estimate because it is perceived subjectively by users, as a result of the user's internal state (e.g., predispositions, expectations, needs, motivation, mood), the designed system characteristics (e.g., complexity, purpose, usability, functionality, relevance), and the context within which the service is experienced (e.g., organizational/social settings, meaningfulness of the activity, voluntariness of use).

In this thesis, we focus on wireless local area networks (WLANs). WLAN has become commonplace in office, and campus sites, and are the most common method of Internet access for home uses. WLANs are subject to an already crowded radio spectrum, and the communication deteriorates due to the dynamics of the transmission medium, fading multipath channels, interferences, and misconfiguration. An automatic approach to network control should be developed because: (i) WLAN users are non-technical; (ii) access providers do not perform direct or online control of the devices in the residences, for legal reasons or because of technical difficulties due to the large number of users; (iii) the increasing demand for more performance by the users; and (iv) the ISP's need to maintain its competitiveness[1]. To achieve that, the WLAN should learn from the transmission of flows to maintain a specific performance metric, and based on past experience, improve the performance. reinforcement learning (RL) guides the learning of an agent using a reward-aware approach. It is very useful in networking, because RL algorithms perform decision making under unknown or very-hard-to-model conditions. In networking, learning algorithms could vary network parameters to meet some performance criteria, such as QoE, providing a more user-centric approach to

[1]Automation can reduce OPEX, improve response time to network disturbances, can cope with service level agreement (SLA), and can focus on user's QoE.

network services. At the same time, the software defined networking (SDN) paradigm can be used to further improve the management of computer networks. SDN advocates the separation of the control and data planes, and the use of a (logically) centralized control. Those characteristics allow the network administrators to have a fuller view of the system, and thus more easily perform global optimizations on the network.

To achieve auto-configuration and auto-optimization capabilities with minimal or no human management, two prerequisites must be fulfilled: a control loop that gathers information from the environment and acts on the devices, and some QoE metric representing the user perceived satisfaction. In this thesis, we use multi-armed bandit (UCB1), Q-learning, and deep Q-learning as control loops, varying from a simple to a complex model. Further, we employ QoE estimators from the state of the art for web, and propose a new QoE estimator for video flows. We built two prototypes for the control loop, being one for web browsing and one for video traffic.To evaluate the web browsing prototype, we use a QoE metric from the literature, composed by three regressors (QoE predictors), one for each of three classes of web sites. We proposed a classification algorithm using semi-supervised learning, which correlates the example of the site classes provided by the authors of the predictor. Next, we developed a QoE predictor as well as a deep Q-learning (DQL) control loop for video traffic. Results show that the control loops improve QoE. In the web browsing scenario, our best result improved the QoE by 167%, if compared to the baselines, and improved the page load times by up to 6.6 times. In the video streaming evaluation the control loop improves the QoE by 91% compared to the baselines in the best case.

Keywords: Communication System Control, Machine Learning, Reinforcement Learning, Quality of Experience, Software Defined Networking, Wireless Networks.

List of Figures

List of Tables

Contents

Glossary

A2C advantage actor critic 39

A3C asynchronous advantage actor critic 39

ABR adaptive bitrate 62, 126

ACK acknowledgment 60

ACS automatic channel selection 93, 94, 107, 108, 134, 135

ADSL asymmetric digital subscriber line 49

ANFIS adaptive neural fuzzy inference system 48

AP access point 9, 10, 15, 16, 18–23, 39, 49, 53, 60, 61, 64, 67–69, 73, 74, 76–78, 80, 87, 89–94, 96, 98, 99, 101, 103–105, 107–111, 114–117, 120, 121, 125–128, 131, 134, 139, 140, 142, 144, 145, 149, 151

API application programming interface xvii, 21, 68, 74, 76, 126, 148

BER bit error rate 53

BSS basic service set 10, 22, 23, 91

CA centrally-controlled multi-agent 85, 93, 106–109, 111

CDF cumulative distribution function 98, 103, 106, 132, 137, 143

CI confidence interval 83–85

CMAB combinatorial multi-armed bandit 32

CNN convolutional neural network 122

CRN cognitive radio network 59

CSA channel switch announcement message 91, 93

DASH dynamic adaptive streaming for HTML 61, 64, 126

DDPG deep deterministic policy gradients 39

DDQL double deep Q-learning 30, 76, 120, 122, 123, 137

DPI deep packet inspection 50

DQL deep Q-learning xvi, 16, 17, 27, 36–40, 62, 113, 115, 142, 147, 148, 151

DRL distributed reinforcement learning 39, 62, 122

ESSID extended service set identifier 21

GD gradient descent 51, 129

GPL general public license 20

GRU gated recurrent unit 122, 123

HAS HTTP adaptive bitrate 62, 126

HetNet heterogeneous network 8

IoT Internet of Things 62, 65

ISP Internet Service Provider 9, 14

k-NN k-nearest neighbors 51, 52

LSTM long short-term memory 122, 123

LTE long term evolution 61, 64

MA multi-agent 102–104, 107–109

MAB multi-armed bandit 17, 26, 30, 31, 68, 89, 110

MAC medium access control 11

MDP markov decision process 25, 27, 51

ML machine learning 9, 13, 23, 30, 39, 52, 80

MLP multi-layer perceptron network 51

MOS mean opinion score xviii–xx, 12, 16, 48–54, 63, 65, 69, 70, 73, 75, 76, 79, 87, 89, 90, 92, 95–98, 100, 102, 103, 105–109, 111, 113, 115, 116, 118, 119, 122, 125, 128–130, 132–135, 137, 143–145, 147

MSE mean square error 138

NDA non-disclosure agreement 90

NN neural network 52

OSI open system interconnection 11

OVSDB open vSwitch database management protocol 20

PE post-encoding 61

POMDP partially observable Markov decision process 63

PSNR peak signal to noise ratio xx, 48–51, 60, 115, 118, 119, 132

Q-learning Q-learning 17, 27, 30, 33–36, 39, 60–64, 68, 77–79, 87, 88, 92, 94, 96–99, 102–104, 106–111, 147, 148

QoE quality of experience xv, xvi, xviii, 7, 8, 11, 12, 14–16, 19, 43, 44, 48–54, 59–65, 67, 68, 70–77, 79, 80, 87, 90, 109–111, 113–115, 117–120, 125, 127, 128, 132, 133, 136, 145, 147, 148, 151

QoS quality of service 11, 12, 20, 22, 23, 48, 49, 52, 53, 57, 60–62, 65, 71

RAT radio access technology 8

RBF radial basis function 129

RL reinforcement learning xv, 13, 15–17, 23–25, 27–30, 35, 37–40, 43, 51, 54, 59, 60, 64, 65, 67–74, 76, 77, 79, 94–96, 101, 103, 109–111, 113, 115, 122, 147, 148, 150

RMSE root mean square error 52, 128, 129

RNN recurrent neural network 122, 123

RSSI received signal strength indication 61

RTT round-trip time 63, 82

SA stand-alone 85, 101, 104, 109, 110

SARSA state-action-reward-state-action 61

SC spectral clustering 81, 84

SDN software defined networking xvi, 15–19, 54, 72, 76, 77, 79, 93, 103, 106, 110, 147

SINR signal-to-interference-plus-noise ratio 59

SLA service level agreement xv, 71

SNR signal-to-noise ratio 49

SSID service set identifier 22, 23

SSIM structural similarity index metric 48, 51, 52

SVM support vector machines 51, 52, 86, 120

SVR support vector regression 52, 53, 129

TCNN temporal convolutional neural network 114, 122–124, 137

TD temporal difference learning 35

TRPO trust region policy optimization 40

TSVM transductive support vector machine 78, 81–84

TSVMSC transductive support vector machine algorithm based on spectral clustering 73, 77, 80, 81, 86

UCB upper confidence bound 28

UCB1 Multi-armed bandit with upper confidence bound 27, 32, 74, 77, 79, 89, 92, 96–99, 102–104, 106–111, 147, 148

VAP virtual access point 22, 23

VoIP voice over Internet protocol 52

VQM video quality metric 51

WLAN wireless local area network xv, 7–10, 14, 15, 17, 18, 52, 53, 61, 105, 147

WMN wireless mesh network 60

Chapter 1

Introduction

Over the past 20 years, IEEE 802.11-based WLAN has become commonplace, and it is the most common method of Internet access for home stations (DiCioccio et al. [2011]). However, this transmission medium is subject to performance problems, and network operations and management still remains cumbersome. Network faults are prevalent primarily due to human error (Mahmoud [2007]). It is also important to take into account the growth of the number of users connected to the Internet using wireless devices and the numerous applications that run on it. Hence, there is a need to propose new and automatic control mechanisms, for example, using machine learning approaches, that provide an improvement in the user's quality of experience.

This chapter presents a brief introduction to wireless network control using reinforcement learning, QoE, and the paradigm of feedback control that are used as the basis of our work. We also discuss the main concepts, techniques and definitions on these subjects. Later, the motivation and this thesis' objectives are addressed. Finally we show our contribution to the field.

1.1 Contextualization

Wireless networks have been increasing their scope, and, due to the dynamics of data transmission, present several challenges not found in wired networks. In addition, home wireless networks are managed by non-technical people, and providers do not deploy full management services because of the difficulties of manually managing thousands of devices. Thus automatic management mechanisms are welcome. In this section we present an overview of the main concepts needed to understand this thesis.

1.1.1 Wireless Networks

The explosive increase in number of connected devices is rapidly affecting the telecommunication industry. Mobile subscriptions grow exponentially as shown in Figure 1.1, and are expected to reach 8.9 billion by 2023 (Ericsson AB [2018]), as shown in Figure 1.2. What's more, 73% of Internet traffic in 2016 was video, and that figure will rise up to 82% by 2021 (Cisco [2017]), due to higher video resolutions (eg, video streaming in 4k and 8k), 3D videos, and augmented reality. This would drastically increase the demand of extra capacity, with aggregate throughput anticipated to rise by a factor of 1,000 (Olsson et al. [2013]). In an already crowded radio spectrum, it becomes increasingly difficult to meet the ever growing application demands. Wireless networks are becoming Heterogeneous networks (HetNets), as they will comprise multiple Radio access technologies (RATs) and network architectures. Also, the wireless densification creates several new problems to the operators in terms of coordination, configuration and management of the network (Valente Klaine et al. [2017]).

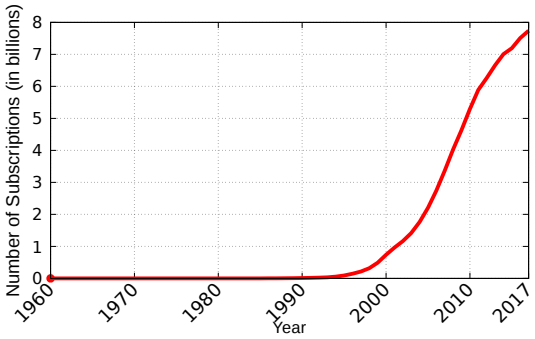

Figure 1.1: Mobile cellular subscriptions - Source: ITU [2018]; Statista [2018]

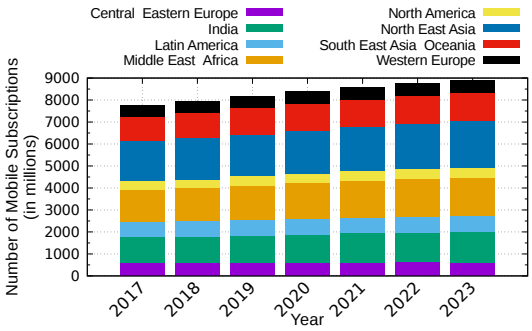

Figure 1.2: Subscriptions Forecast - Source: Ericsson AB [2018]

WLANs have become commonplace in office and campus environments, and they are the most common method of Internet access for home stations (DiCioccio et al. [2011]). In this thesis, the term WLAN refers to wireless networks that follow the IEEE 802.11 standard. Recent estimates suggest that more than 10 billion WLAN devices have been sold in total, and that more than 4.5 billion of those are in use today (Biswas et al. [2015]).

The wireless medium is subject to performance problems, such as packet loss, delay and low connection speed, often caused by interference of other wireless networks, signal propagation issues, and misconfiguration. The analysis of WLAN quality may provide some insights into the cause of poor performance, but most users do not have the required technical knowledge to do that (Alves et al. [2017]). Often, users only notice that there is a problem in the network when their QoE degrades (Hora et al.

[2016]), e.g., the user notices a slow Web browsing or the media she[1] is watching freezes. Many home WLAN Access points (APs) are provided by the Internet Service Providers (ISPs), e.g. BroadBand Now states that fixed wireless provides 51% coverage in the US (BroadBand Now [2018]). The ISPs's technicians may have the necessary expertise to analyze the data, however, most of the time they avoid monitoring home traffic because of privacy issues, and this monitoring poses a huge effort on their personnel (Baraković and Skorin-Kapov [2013]).

The complexity of managing and operating these networks is forcing operators to find new strategies to remain competitive, such as automatic management systems. Manual operation of networking services is costly. For example, Clinckx and Baffalio [2014] say that automation can provide network operation gains up to 2 % and 13 %, for ISPs and Mobile network operators, respectively. Given the important cost reductions, Joshi [2018] predicts as four (in an integer scale from one to five) the probability of ISPs adopting artificial intelligence and five to adopt Machine learning (ML) to assist telecommunication companies in improving their services. In addition to this, new telecommunications architectures such as C-RAN[2] (Chen and Duan [2011]) require very short decision times, which are achieved only through management automation.

1.1.2 Feedback control

Technically, control consists of regulating the characteristics of a subject system. A system is a group of parts and processes that we want to control. Figure 1.3 shows a schema of a feedback control system. The input of the system is influenced by the output and the environment, while the output of the system is set to a desired value by changing system parameters, i.e., the output is the expected result. The control module uses the input signals to define how the system parameters should be changed to obtain the expected result, and, in feedback control, the system response (output) is fed back to the controller, as an input, thus the controller knows how its actions affect the system. A system can also suffer disturbances (that can be modelled as uncontrolled inputs), which affect the measured output. Because of the loop provided by the feedback, the control described above is also known as closed-loop control or feedback control.

The most common objectives when using feedback control are: (1) *Regulatory control*: it ensures that the measured output is equal to (or near) the reference input;

[1] In this thesis, the pronoun *she* is used for an unknown referent as an attempt to balance out the perceived sexism of generic *he*. If there is a need to disambiguate between two participants in an action, the two pronouns will be used.

[2] Cloud Radio access network.

(2) *Disturbance rejection*: The controller ensures that disturbances do not significantly affect the measured output; (3) *Optimization*: When the controller works to obtain the "best" value of the measured output.

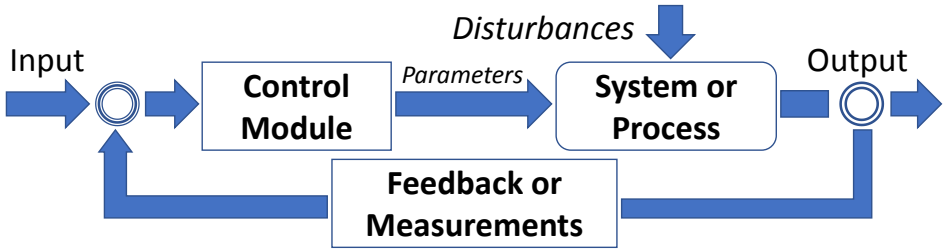

Figure 1.3: Feedback control

A person controlling a light is an example of closed loop control. When tuning a radio or turning on/off the light, the human acts as the feedback system during the system setup. Thus if, during the night, the lamp burns, the place goes dark, and only when the human returns, the system is repaired.

There is another type of control that does not use feedback, and for this reason is called open loop control. Formally, open loop control occurs when the control is determined as a function of the input parameters (Kirk [2012]). Thus, for the open loop control to be optimal, the system depends only of the initial state (Engelberg [2015]). Thus, either there is no disturbance, or it is negligible or it is predictable (in this case it is considered as another input).

A typical home wireless network is an example of an open loop control. The person buys an AP. She installs it, and turns it on. Then she sets the configuration parameters (e.g, the channel, the transmission power and other WLAN parameters), creating the Basic service set (BSS). After that, she expects that the AP works fine. There is no closed loop inside the AP that, for example, changes the transmission channel if the channel becomes too crowded.

There are pros and cons in using open or closed control loops (Dorf and Bishop [2011]; Kirk [2012]; Franklin et al. [2014]). open loop control is the best choice when the output rarely changes or does not change at all, i.e. the system always reaches a steady state given an input. In addition to that, process disturbances are extremely rare. open loop control can be implemented at a lower cost than a closed loop system. On the other hand, closed loop control requires direct or inferred system measurements, so a prerequisite for its use is the feasibility of obtaining the measurement. Besides, this method requires that the process to be controlled has a degree of predictability, i.e., an approximate response is known for the input controls. The closed control loop is useful when the output varies from a desired outcome, and cannot be dealt using the

"set and forget" approach of open loop control. Although closed-loop control systems can be expensive, they are usually cheaper than using humans.

1.1.3 Quality of Service vs Quality of Experience

Quality of service (QoS) refers to the network's ability to guarantee a certain level of performance to a flow based on network parameters. Formally, as defined in RFC 2386, QoS is a set of service requirements to be met while transmitting a flow from source to destination (Crawley et al. [1998]). According to Gomes et al. [2009], there are different mechanisms and algorithms that provide QoS, and which mainly act on the application and network layers of the Open system interconnection (OSI) model. In a wireless scenario, other layers are also involved, e.g, physical and link layers provide resource reservation signaling (Ni et al. [2004]), and specialized QoS Medium access control (MAC) (Verma and Dean [2015]; Chen and Varshney [2004]).

Different applications may have different QoS requirements (Ehsan and Hamdaoui [2012]), such as throughput, delay, bandwidth, and jitter. Table 1.1 compares the importance of each metric for the QoS of some types of applications (Guimaraes et al. [2018]). For example, in audio broadcast, a low latency is required to mitigate delayed voice. The delay can create awkward periods of silence or cause errors in receiving audio. On the other hand, in e-mail or in file downloads, throughput is the most important metric.

Table 1.1: Application Sensitivity for each QoS Metric.

Type of traffic	**Throughput**	**Loss**	**Delay**	**Jitter**
E-mail	Low	High	High	Low
Audio	Very low	Average	High	High
HTTP	Low	High	High	High
Remote connection	Low	High	Average	Low
FTP	High	Average	Low	Low
Videoconference	High	Average	High	High

The QoS refers to the performance of a network, however it does not represent the user's expectations of the quality of service. QoE approaches overcome those limitations. According to ITU-T [2007], QoE is described as the overall acceptability of an application or a service, as perceived subjectively by the end user. Thus QoE is difficult to be measured directly, and it is frequently expressed in inexact terms like 'good', 'excellent', 'poor', etc (Soldani et al. [2007]). Also, the perceived quality is influenced by the complete end-to-end system effects, as well as factors outside of network control such as user expectations, and context. Recently, the EU Qualinet Community proposed a QoE definition, based on the results of the research project

COST Action IC1003, as the degree of delight or annoyance of the user of an application or service.

Because traditional QoS metrics do not reflect the user's experience (Serral-Gracià et al. [2010]; Aguiar et al. [2011]), new concepts are needed to represent the QoE. The MOS value is one such concept ITU-T Recommendation P.10/G.100 [2016]. ITU-T G-1030 maps the MOS score in a continuous range from 1 to 5, in which the value increases with user satisfaction. Table 1.2 shows the five-point Likert scale, which maps MOS to more human understandable concepts, in the form of impairments. This scale indicates how much the network service operation affects a user due to a temporary or permanent disturbance (or interruption).

Table 1.2: MOS scale

MOS	**Quality**	**Impairment**
5	Excellent	Imperceptible
4	Good	Perceptible but not annoying
3	Fair	Slightly annoying
2	Poor	Annoying
1	Bad	Very annoying

Source: Based on Nasrabadi et al. [2014]

It is difficult to request humans to rate a service every time that a network parameter changes. Thus, an automatic QoE assessment tool is needed for real-time usage. Using a model, one can approximate the QoE of the clients out of relevant QoS parameters associated with QoE Kim et al. [2010]. In this way, delivering high QoE requires understanding the factors contributing to the user's perception of the services, and applying that knowledge to define the operating requirements (Soldani et al. [2007]). It is still a challenge to manage the QoE influencing factors because (Wang et al. [2017]): (a) the need to quantify subjective factors, (b) contextual factors are difficult to obtain, and (c) it is hard to obtain an accurate model correlating the important factors and QoE. Although several initiatives attempted to evaluate QoE in networks, this is still an open field. Each application should be evaluated so as to obtain its own QoE metric.

1.1.4 Machine learning and Reinforcement learning

There are several wireless network control parameters. According to Morocho-Cayamcela et al. [2019], several of these parameters are configured using some heuristics, many of which do not use a closed loop control mechanism to adjust them. In traditional automatic control systems, the designer creates a system model that relates

the inputs to the desired output. From that model, she or he derives an algorithm that solves (sometimes by approximation) the system of equations. However, we aim to implement a control algorithm that does not know the operating model of the system *a priori*, thus it should derive it by itself. One such approach is said to perform artificial learning.

Learning, like intelligence, covers such a broad range of processes that it is difficult to define precisely. ML usually refers to changes in systems that perform tasks associated with artificial intelligence. These tasks involve recognition, diagnosis, planning, control, forecasting, prediction, modeling, classification, etc (Nilsson [1996]; Alpaydin [2009]; Kelleher et al. [2015]). Kelleher et al. [2015] define machine learning as "an automated process that extracts patterns from data", while for Nilsson [1996], a machine learns based on its inputs or responses to external information, whenever its structure, program or data are altered in order to improve the expected future performance. A typical artificial learning agent perceives and models its environment, and calculates appropriate actions in order to anticipate their effects considering its objectives. Changes made to any of the agent's components can count as learning.

In this thesis, the learning agent uses RL. We have selected RL among many other control methods due to: (1) we wanted the system to be able to learn from a few examples without a model of the environment, and for the learning to take place while the task is being performed online; and (2) we did not make any assumption about the transition, and the reward probability distributions, then Bayesian methods are not a good option. Because the system runs without previous knowledge of the environment, we do not need a large amount of labeled data to train a model in advance.

There are two drawbacks in using RL: (i) since learning is predominantly online, the agent might have to run many trials in order to produce an effective result; and (ii) not many scenarios allow the agent to explore without serious consequence. However, in our setup the task seems simpler enough, the actions are discrete, and the information from the medium is readily available, which makes the real-time performance constraints lighter, and the agent can learn good configurations fast. Also, our scenarios are not very critical because they deal with a type of application that will only annoy the user if the quality is degraded by an exploration step. Moreover, the same cannot be told if the application is critical, for example, the transmission of life support equipment may incur in life loss if the transmission is degraded or loss. In these cases, the learner should care about making safer configurations. Such applications can be addressed in future work.

RL models how stimuli (reward) influence behavior in the pursuit of goals. In this way, the agent "perceives" the environment and when it executes an action, it oc-

casionally receives a reward. The agent is encouraged to choose actions that maximize rewards either on the short or on the long run. It must therefore predict how actions lead to rewards.

1.2 Motivation

WLAN is the most common method of Internet access for home stations (DiCioccio et al. [2011]), but its transmission medium is subject to performance problems. Vendors sell commercial wireless controllers that provide centralised WLAN configuration, but the platforms are closed. Thus, clients rely on the vendor's initiative to provide new features (McKeown et al. [2008]; Opara-Martins et al. [2016]). However, despite these approaches, network operations and management still remain cumbersome, e.g., Clinckx and Baffalio [2014] highlight that mobile network operators can save up to 13% in OPEX with better network operations. An study in a wireless service provider in China showed that the use of Self-Organizing Networks can improve in 58% their net profit and reduce in 55% the interconnection expenses (Kamboh et al. [2017]). Hence, the network can profit with the use of an automatic control mechanisms, which improves the user's quality of experience in wireless networks. Such mechanism would indirectly generate more profit to the ISPs or less cost to the network administrator, due to the higher user satisfaction and a reduced need of human intervention.

The motivation of this thesis lies in the fact that progress in the area of automatic wireless network control brings benefits to the general public, due to the ubiquity of wireless devices. Besides that, the research and development of automatic control projects for QoE improvement is of interest to technology companies and network services. The premise is that the use of QoE metrics for the automated management of wireless networks generates performance improvements for the users.

1.3 Problem Definition and Objectives

We formulated the following research question: *How to improve the QoE in wireless networks with little to no human intervention?* Since this topic is very broad, we narrowed it down to the following objective:

> Given the cumbersome nature of the network management and the difficulty of providing acceptable QoE in wireless network, the objective of this thesis is to improve the QoE by automatically reconfiguring the wireless network using reinforcement learning.

Since QoE is dependent on the application, we proposed and evaluated closed-loop controls for two separate applications: web browsing and video streaming. The general objective was broken down into the following specific objectives:

1. Propose an architecture for wireless network control using reinforcement learning and QoE;
2. Evaluate online machine learning techniques that learn the best actions for wireless network, without the use of previous network behavior models;
3. Develop and evaluate an automatic control algorithm for wireless networks that improves user QoE for a Web application.
4. Develop and evaluate an automatic control algorithm for wireless networks that improves the user's QoE of a Video application by changing physical and link layer parameters of the wireless network.

1.4 Contributions

From the study of RL and SDN in WLANs, we achieved the following contributions:

- An extensive research on the state of the art of reinforcement learning in wireless networks. This study is summarized in Chapter 3;
- An architecture based on RL to control wireless networks improving the QoE, as shown in Chapter 4;
- New model-free online control algorithms for wireless network using RL;
- A semi-supervised learning method to classify web sites based on traffic patterns from wireless stations, which is detailed in Section 5.3;
- A module for the *Ethanol* SDN architecture to control the AP using RL, which is available in GitHub.
- A southbound interface using python (available in GitHub) to control the APs using linux commands.

- We developed a framework to control APs using DQL, which is also available in GitHub;
- A new MOS predictor for video transmissions;
- A reward function that considers both the QoE and the fairness in the distribution of QoE among the wireless stations.

1.5 Organization

This work is structured into seven chapters, including this introduction. The next chapter introduces some background concepts, covering wireless networks, and the RL methods used in this thesis. Chapter 3 presents related work about the application of RL in wireless networks. Chapter 4 describes the proposed architecture to control wireless networks using RL. After that, Chapter 5 uses RL and SDN to control link layer parameters to improve user's QoE. Chapter 6 proposes a wireless control loop that optimizes video QoE. Finally, Chapter 7 presents the final considerations, the conclusions, future work, and a list of the publications during the Ph.D.

Chapter 2

Background

This chapter discusses concepts related to wireless networks, the measurement of user satisfaction in networks, the SDN paradigm, and the reinforcement learning methods used in this thesis. This chapter is organized as follows. Section 2.1 discusses wireless local area networks. The next section explains *Ethanol*, the wireless SDN platform used in the experiments. Section 2.3 describes what is reinforcement learning, and highlights how RL can be used in wireless networks. The next three sections show in more detail the RL techniques used in this thesis: multi-armed bandit (MAB) in Section 2.4, Q-learning (Q-learning) in Section 2.5, and DQL in Section 2.6. Finally in Section 2.7, some frameworks for RL are shown, and we discuss some pros and cons in using them.

2.1 Wireless local area networks using IEEE 802.11

This section focuses on IEEE 802.11-based wireless technology, which we will call WLAN in the rest of this text. It has been widely deployed in places such as homes, coffee shops, public hot-spots, universities, airports, large organizations, etc, mainly due to its unlicensed frequency band and low-cost equipment.

WLAN forms a local area network within a limited area. The subset that we are studying is defined by the 802.11 family of IEEE standards. These standards form a set of media access control and physical layer specifications for WLAN communication in the 900 MHz and some gigahertz frequency bands – 2.4 GHz, 3.6 GHz, 5 GHz, and 60 GHz. There is a main standard called 802.11, and many amendments. When a new main standard is released, some amendments are incorporated into the latest version. The last release was in 2016 (http://standards.ieee.org/findstds/standard/802.11-2016.html). The corporate world, however, tends to market using the amendments, because

they denote the capabilities of the products. That is why we hear, for example, that a device runs in a 802.11a or in a 802.11g band.

The 802.11 standard defines different types of network arrangements as shown below. The experiments in this thesis use the infrastructured type.

1. Infrastructured wireless network: in this type of network there are base stations, and client nodes (called wireless stations) communicate via the base stations. The APs usually have a cabled network connection, and may have permanent wireless connections to other nodes.

2. Wi-Fi Direct: Allows devices to easily connect with another node in range, without requiring an AP. It is a single hop communication. Wi-Fi Direct creates groups of nodes, using AP-like functionalities, and the devices negotiate who will have the role of an AP (Camps-Mur et al. [2013]).

3. Peer-to-Peer: There is no base station. Each node talks to other neighboring nodes. Routing and forwarding are provided in a multi-hop mesh fashion. Any station might be source, sink, or propagator of traffic, and the stations use mesh coordination function to access the channel.

4. Bridge mode: the bridge connects a cluster of wired users (that is, a wired network) to an access point via a wireless link. In other words, a wireless link from a bridge to an AP is established to interconnect two wired networks.

A Wi-Fi deployment is also classified based on its management (Mishra et al. [2006]): 1) centrally managed or 2) uncoordinated. Centrally managed deployments are usual in places such as university campi, offices or airports, where a wireless controller manages the APs. On the other hand, uncoordinated WLANs have no central control, and are typical in places such as residences or private hot-spots in commerce (e.g., restaurants, coffee shops, etc.). Successful deployment in either case requires efficient mechanisms to address performance issues, such as excessive interference, which usually translates into low throughput.

2.2 Software Defined Network – SDN

SDN is a network paradigm based on the separation of the control plane and the data plane (Macedo et al. [2015]). The control manages traffic, leaving the responsibility of forwarding packets to the data plane. Figure 2.1 shows the SDN architecture.

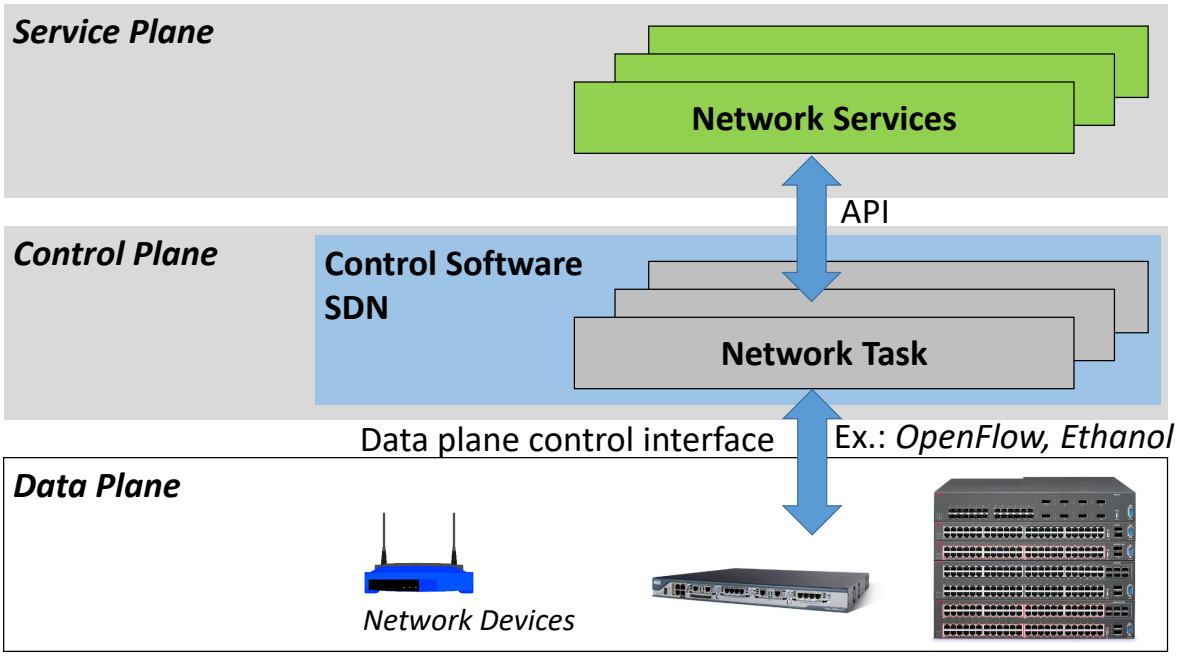

Figure 2.1: Software Defined Networks Architecture.

The control plane controls the devices using a network controller. This controller has a global view of the network, i.e. it knows the network state. Usually it is a single computer or it can be distributed across multiple computers, acts as a unique logical entity. The controller has more storage and processing power than the network devices and is, thus, able to perform more complex tasks. The data plane consists of network traffic devices, such as switches, APs, and routers, as well as other manageable devices, such as *middleboxes*.

An extension to the SDN architecture, proposed by Clark et al. [2003], incorporates the "knowledge plane". This plane would contain cognitive algorithms, which reconfigure the networks to diverse situations, aiming to optimize the performance and resiliency. In other words, SDN should provide auto-configuration and auto-optimization capabilities (Barco et al. [2012]) with minimal or no human administration (Marchetti et al. [2010]). There are two prerequisites for this: a control loop algorithm that reads the environment and acts on the devices, and metrics representative of the QoE experienced by the user.

2.2.1 Ethanol

Ethanol is an SDN architecture for IEEE 802.11 networks that configures and manages wireless resources. It provides high level calls to control the underlying IEEE 802.11 infrastructure. It requirements are (a) flexibility in management and configuration, (b)

adaptability and (c) independence of wireless suppliers (Moura et al. [2019a]). With *Ethanol*, the controller uses a a southbound interface to request network usage metrics to the APs it manages as well as to the client nodes. The control algorithm uses these metrics as input, adapting network parameters to improve the operation of the wireless network.

Ethanol is available as open source under general public license (GPL) version 2. The source code can be found in GitHub at https://github.com/h3dema/ethanol_hostapd for the AP, and https://github.com/h3dema/ethanol_controller for the controller. *Ethanol* is also available as a docker container.

The *Ethanol* architecture has three types of devices: the controller, the OpenFlow-enabled switches, and the *Ethanol*-enabled APs, as shown in Figure 2.2. The controller runs on a computer connected to the infrastructure, i.e, in the wired network or virtualized in the cloud. The *Ethanol* AP is an AP running *Ethanol* code. The wired devices are OpenFlow-enabled switching elements. Since OpenFlow does not provide a control interface for QoS, *Ethanol* adds this functionality using the open vSwitch database management protocol (OVSDB) defined in RFC 7047. The architecture can be separated into wireless and wired control protocols, and acts independently or in a coordinated fashion on both networks.

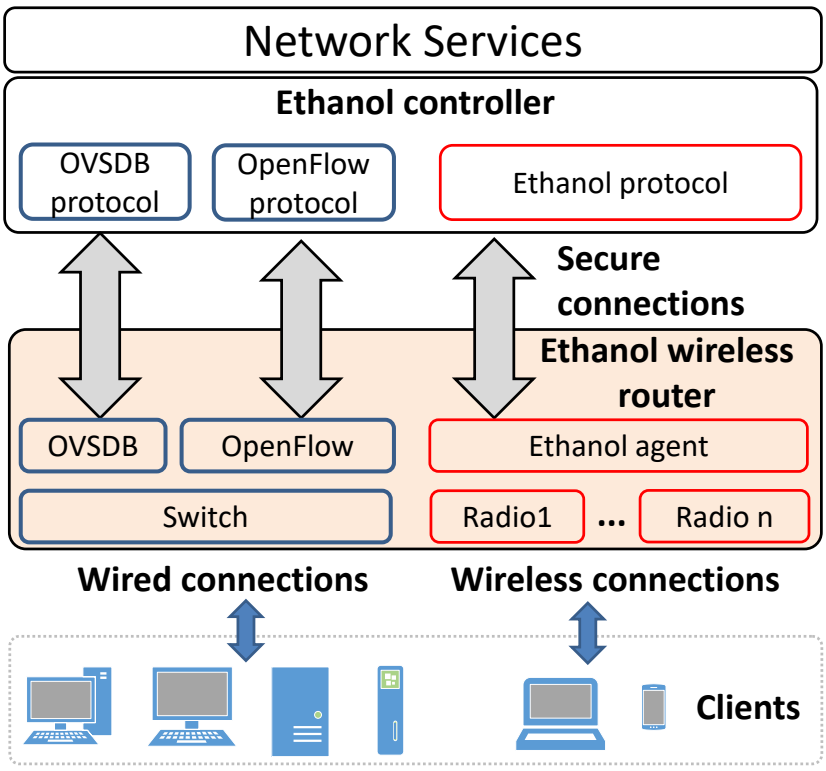

Figure 2.2: Ethanol AP implementation.

The *Ethanol* controller API is designed upon an object-oriented approach, where entities have properties, methods, and events. An *Ethanol* AP and a VirtualAccessPoint are examples of a physical and a virtual entity, respectively. Those entities have observable and/or configurable *properties*, such as the number of available channels or extended service set identifier (ESSID). The properties are accessed via *get*/*set* methods or publish/subscribe methods. Finally, entities may have *events*. One example is a wireless client requesting an association. The controller registers event notifications with the AP, and if applicable a threshold for triggering the event.

Figure 2.3 shows the entities, properties and events of *Ethanol*. To improve readability we have omitted all getter and setter methods. Read only properties are marked with a minus ('-') sign. A filled diamond shape indicates containment, a stronger form of aggregation where the contained objects only exist within their container (the class touched by the diamond). Cardinality is represented using Crow's foot notation. All methods with the "ev" prefix correspond to events.

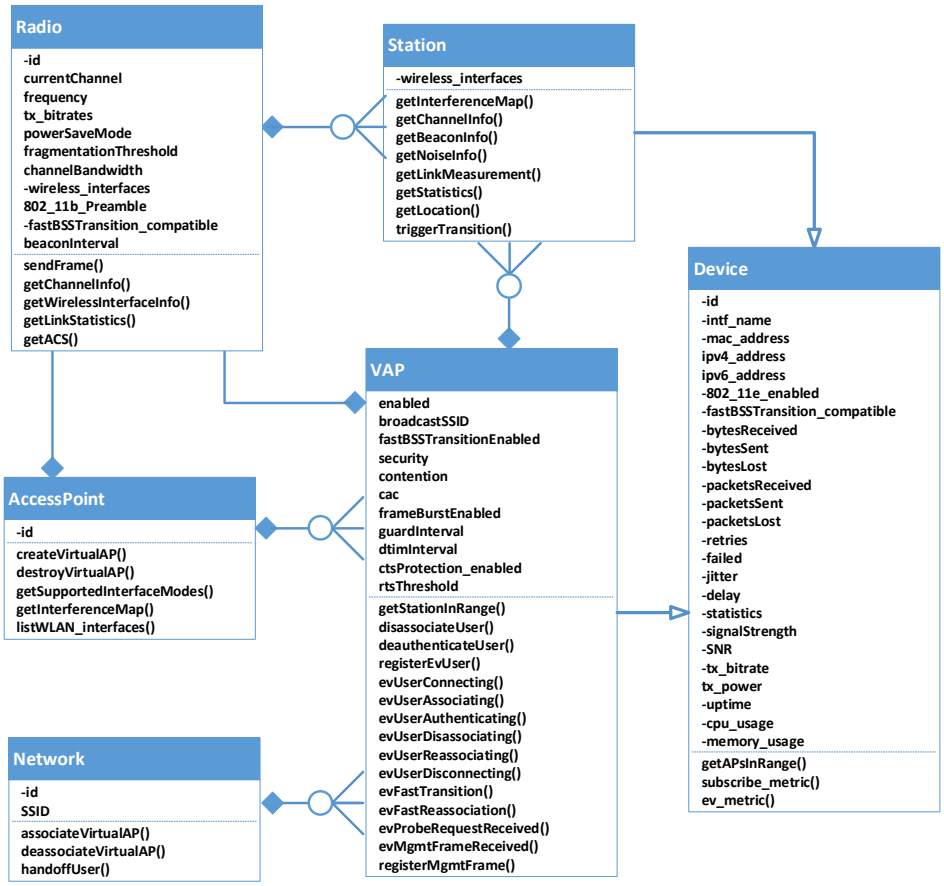

Figure 2.3: *Ethanol* control API model.

The following subsections present the *Ethanol*'s entities of the model shown in

Figure 2.3. More information can be found in the documentation at the Github repository (https://github.com/h3dema/ethanol_controller).

AccessPoint entity

This entity represents physical devices. An AccessPoint can have one or more physical radios, represented by the Radio entity, and one or more virtual access points (VAPs) running on the AP (VirtualAccessPoint class). This entity has three main attributes: *beaconInterval* (the frequency of beacons), *fastBSSTransition_compatible* (if the access point is compatible with fast BSS transition) and *802.11b_preamble* (if the preamble is long or short). This entity also creates and destroys virtual access points, as well as determines the state of the device. It has a list of network cards, the modes they support (e.g. ad hoc, infrastructure), and a method to request an interference map.

Radio entity

This entity configures the physical wireless interface of the AP. It has attributes such as the current channel, the supported bit rates, the current transmit power, and it enables or disables the power saving mode. This entity also gathers link statistics and other information from the wireless radio.

Device entity

This is a super-class of VAP and Station entities that configures and collects information from the client device. This entity contains information such as the MAC and IP addresses, and if the station supports 802.11 QoS modes. The entity also collects information about the link between the station and the AP, such as number of bytes/packets received and sent, signal strength, SNR, bitrate, number of retries, and packet loss.

VAP entity

VAP represents a network inside a physical device, so several users can connect to one VAP running in one physical device. A physical device can have zero or more VAPs (so that we can configure a VAP but keep it disabled for future use or fast startup). Stations connect to a VAP, and a group of VAPs form a Network. VAP inherits the properties of the Device superclass.

A VAP does not broadcast its service set identifier (SSID) if *broadcastSSID* is disabled. Also VAP controls MAC transmission parameters such as guard and DTIM intervals, RTS threshold, and link capabilities (e.g. if frame burst is enabled). It also

exposes the contention parameters of 802.11 QoS BSS (e.g. maximum and minimum contention window values, AIFS values) and admission control parameters. Each VAP also has its own security parameters. User association and authentication requests can be allowed or denied using events in this entity. The entity also has events for fast transition and fast reassociation, as defined in IEEE 802.11/2012 BSS Transition Management, and for probe requests.

Network entity

A network may contain several VAPs. This entity represents the network and its SSID. It provides methods for the association and dissociation of APs into the network, as well as a method to request a user handoff with 802.11 fast transition.

Station entity

This entity represents a station, and inherits properties and methods from the Device superclass. Wireless metrics, like bytes received or sent, are collected using messages from existing IEEE 802.11 standards. This entity also returns measurements using IEEE 802.11 radio resource management. Examples are channel reports ($getLoadInfo$, $getNoiseInfo$, $getInterferenceMap$), a list of APs in range ($getAPsInRange$ – useful for pre-handoff optimizations), counter group values (e.g. transmitted fragment counts, multicast transmitted frame counts, and failed/retry counts) using the $getStatistics$ method, among others.

2.3 Reinforcement Learning

Machine Learning was defined in 1959 by Arthur Samuel as "the field of study that gives computers the ability to learn without being explicitly programmed" (Boutaba et al. [2018]). The learner is the agent (program) receives information and, through learning, increases its knowledge. ML can be broadly divided into four learning paradigms: supervised learning, unsupervised learning, semi-supervised learning, and RL (Alpaydin [2009]; Mohri et al. [2012]):

1. **Supervised learning**: the agent receives a set of labeled examples as training data and makes predictions for all unseen points (situations). The labeled data acts as a supervisor, which guides the learner to the right answers.

2. **Unsupervised learning**: The training data is unlabeled. The agent searches for patterns in the data, thus using methods such as grouping or dimensionality reduction.

3. **Semi-supervised learning**: Part of the training data is labeled, and the rest (usually most of the data) is not labeled. It is used when unlabelled data is easily accessible, but labeling is expensive. The reasoning is that unlabeled data can improve the performance of the learning with regards to a purely supervised approach.

4. **Reinforcement learning – RL**: In this method, the agent actively interacts with the environment and, by perceiving a reward, it learns with each interaction. This method is capable of searching for optimal control solutions with nonlinear, possibly stochastic dynamics that are unknown or highly uncertain (Buşoniu et al. [2018]). RL is situated between supervised learning and unsupervised learning (Kaelbling et al. [1996]; Alsheikh et al. [2014]) because: (i) there is no presentation of input/output pairs to the algorithm, as in supervised learning; (ii) the agent is not told which is the best action in the long-term, as in a supervised learning; (iii) the agent aims to select better actions in the long run, and is not interested only in discovering a pattern, as in unsupervised learning; (iv) the agent must gather useful experience, explicitly exploring its environment through trial-and-error interactions; and (v) the on-line performance is important, because the system is often operational concurrently with learning.

This work develops a closed loop control mechanism, as described in Section 1.1.2. This control uses a feedback measure that allows the controller to define the action to take, so RL is the most suitable approach. Henceforth, from now on we will focus on reinforcement learning.

In RL an agent decides which action to take in order to maximize a cumulative (frequently discounted) reward by interacting with its environment (Sutton and Barto [1998]). RL solves nonlinear stochastic control problems without using a model (Kazemi et al. [2012]). Learning works in a closed loop control, where the action selected by the controller, the agent in RL, is dependent on feedback from the process in the form of the value of the process variable (the reward in RL). After each action, the agent receives an immediate reward, so the objective of the agent is to maximize his reward over a course of interactions with the environment.

The closed loop in RL (shown in Figure 2.4) can be summarized as follows. After choosing an action in the current state, the agent changes to another state, and it receives a scalar reward (feedback) from the environment, which indicates the quality of its action. A policy is a function that defines which action to be taken in a certain state. The goal of the agent is to find a policy π that maximizes the return R. The return can be the total accumulated reward, the immediate reward, a discounted accumulated reward, or a combination of these. The agent builds a function Q that estimates the return when following a given policy.

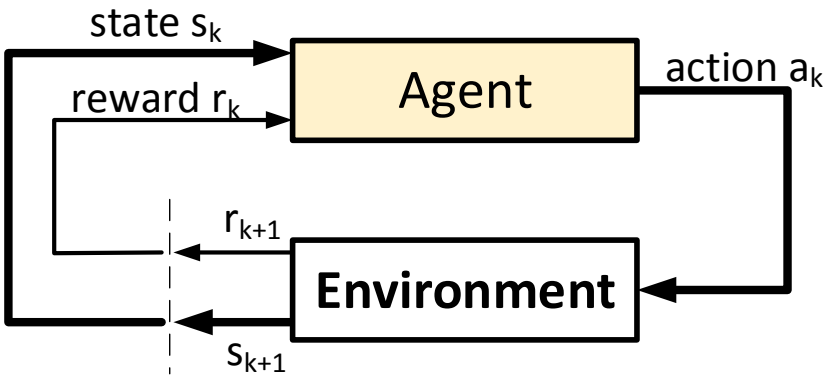

Figure 2.4: The agent-environment interaction in reinforcement learning. Based on Sutton and Barto [1998]

Formally, the RL problem in discrete time is a markov decision process (MDP) characterized by a tuple $< S, A, P, R >$, where S is a finite set of states, A is a finite set of actions, P is a transition function defined as $P : S \times A \times S \rightarrow [0, 1]$ and R is a reward function defined as $R : S \times A \times S \rightarrow \mathbb{R}$ (Sugiyama [2015]; Sutton and Barto [1998]). At each time t, the agent observes a state $s_t \in S$, takes an action $a_t \in A$, and then transitions to state s_{t+1} with probability $P(s_{t+1}|s_t, a_t)$, receiving a reward r_t.

A deterministic policy $\pi : S \rightarrow A$ is a function that outputs, for each state $s \in S$, an action $a \in A(s)$, where $A(s)$ is the subset the valid functions in state s. $V^{\pi}(s)$ is the expected return, under policy π, when the agent starts in state s, and follows the policy π thereafter. The return is written as Equation 2.1, using the infinite-horizon discounted model, where $0 \leq \gamma \leq 1$ is the discount factor, r_t is the reward at time step t, and s_t is the state at t. Table 2.1 shows the symbols used in this section with a brief description in order to provided the reader an easy way to cross-reference the symbols.

$$V^{\pi}(s) = \mathbf{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s\right] \tag{2.1}$$

Table 2.1: Symbols used in Section 2.3

Symbol	Description
α	the value of the learning rate
a, a_t	the action selected in t
D	the set of devices
$\mathcal{D}$	the replay memory buffer
ϵ_0	initial value of epsilon in the epsilon-greedy strategy
γ	the value of the discount factor
K	number of arms in MAB
K	number of samples in the minibatch to train the network
k	number of steps taken by the agent in order to updated the target network
$I_j(t)$	the index in MAB for arm j
$\mathcal{L}()$	a loss function
n_j	number of times the arm j was pulled
N	maximum size of the replay memory buffer
π	a policy
π^*	an optimal policy
$\mathbf{P}(a, \pi(s), s')$	the transition probability function
$Q(s, a)$	the action-value at state S with action a
$Q(s, a, \theta^-)$	the Q-network with parameters θ_t^-
$\hat{Q}(s, a, \theta)$	the Target-network with parameters θ_t
r, r_t	the value of the reward when the action a is taken in state s
$R(s, a, s')$	the reward function
s, s_t	the current state (in timestep t)
s', $s_t^{'}$	the next state when in time t the agent is in state s and selects action a
$\mathcal{T}$	the size of the episode (period)
$V(s)$	the state-value at state S

A similar state-action value function $Q : S \times A \rightarrow \mathbb{R}$ when starting in state s, taking action a, and following the policy π thereafter is defined as follows:

$$Q^{\pi}(s, a) = \mathbf{E}_{\pi}\left[\sum_{l=0}^{\infty} \gamma^l r_{t+l} | s_t = s, a_t = a\right] \tag{2.2}$$

Both equations 2.1 and 2.2 can be written recursively in terms of a Bellman Equation (Sutton and Barto [1998]). For example, the value function V can be rewritten as

$$V^{\pi}(s) = \sum_{s' \in S} \mathbf{P}(a, \pi(s), s') \left[R(s, a, s') + \gamma V^{\pi}(s')\right] \tag{2.3}$$

Since the goal of RL is to find the best policy, the equation below maximizes $V^{\pi}(s)$:

$$\pi^*(s) = \arg\max_a \sum_{s' \in S} \mathbf{P}(a, \pi(s), s') \left[R(s, a, s') + \gamma V^{\pi}(s')\right] \tag{2.4}$$

Based on the result above, one can devise two dynamic programming methods to solve the MDP problem. They are called policy iteration and value iteration (Sutton and Barto [2018, 1998]). In policy iteration, the agent follows a policy, *i.e.*, it defines each action to take for each state. In value iteration, however, the agent does not have an explicit policy; it decides the action based on the value function (which measures how good each state and/or action is). In this thesis we use value iteration methods called Multi-armed bandit with upper confidence bound (UCB1), Q-learning and DQL, which are described in Section 2.4, Section 2.5, and Section 2.6, respectively.

The RL methods listed above rely on value-iteration policy. According to Sutton and Barto [2018], using policy iteration can result in great increase in the speed of convergence of the policy evaluation. However each of its iterations involves the policy evaluation of full episodes, which require intense computation. Thus in our case, we selected methods that use value-iteration because they require less computer power, and they work well in non-stochastic environment as in our case.

There are many new (and promising) approaches to solving problems using RL. For example, in actor-critic methods, instead of waiting until the end of the episode to update the gradient, they make an update at each step, using a critic model that approximates the value function. The update can be asynchronous, such as in Mnih et al. [2016], or synchronous, as in Espeholt et al. [2018]. The policy can be accelerated as in Stooke and Abbeel [2018]; Schulman et al. [2017]. Such approaches showed good results in RL benchmarks, *e.g.* Atari games, 2D physics and robotics simulation. However these methods are complex to implement, and they demand several rounds to uncover a good policy. Thus, we decided to keep it simple, and focus on the network aspect of the problem. Therefore, we selected for our work the most common RL methods discovered in the related work (Section 3.3).

No long-term reward feedback is provided by the environment, thus an RL agent is faced with the exploration versus exploitation dilemma (Yogeswaran and Ponnambalam [2012]; Berger-Tal et al. [2014]). This means that it must choose between exploring unknown actions to gain more information versus exploiting the information already collected to increase the accumulated reward. The following subsection elaborates more on this dilemma.

2.3.1 Exploration and Exploitation

In RL there is a trade-off between acquiring new knowledge and using the current knowledge to reap the reward (Berger-Tal et al. [2014]). This trade-off is called the Exploration-Exploitation Dilemma (Sutton and Barto [1998]).

The exploration-exploitation dilemma is an important issue in RL (Zhang and Pan [2006]; Yogeswaran and Ponnambalam [2012]). Agents should explore to improve future rewards, yet they also need to exploit the current knowledge in order to improve the long-term reward. Pure exploration degrades the accumulated (discounted) reward in the long run, but increases the agent's flexibility to adapt in a dynamic environment. On the other hand, pure exploitation leaves the agent "stuck" into local optimal solutions. An effective exploration should probe only promising regions of the search space. The challenge is how to limit the region size, and how to estimate the expected benefits of exploring that region.

There are a number of exploration methods in the literature, such as ϵ-greedy (Sutton and Barto [1998]; Kuleshov and Precup [2014]), Boltzmann or Softmax exploration (Sutton and Barto [1998]; Cesa-Bianchi et al. [2017]), value-difference based exploration (Tokic and Palm [2011]) and upper confidence bounds (Auer et al. [2002]; Auer and Ortner [2010]). In the remaining of this section we will describe two methods, since they are used in this thesis. They were selected because they are simple to implement and configure, and Kuleshov and Precup [2014] highlight that simple heuristics such as the ones below outperform more complex ones.

The ϵ-greedy algorithm is widely used because it is very simple, and, in general, it performs well in a variety of applications with a finite-time horizon (Sutton and Barto [1998]; Vermorel and Mohri [2005]; Maes et al. [2012]; Kuleshov and Precup [2014]). The horizon corresponds to the time the learning comes to an end, either because the agents succeeded or failed. ϵ-greedy selects random actions from a set of available actions following a uniform distribution. It selects a random action with probability equal to ϵ (exploration), and follows the current best policy with probability $(1 - \epsilon)$ (exploitaton). One disadvantage of the ϵ-greedy algorithm is, when exploring, it chooses equally among all the actions. There is no explicit preference for unexplored actions, and thus the worst criterion and a criterion close to optimal are equally likely to be explored. However, it has been proved that if $\epsilon \to 0$ as $t \to \infty$, the value function converges to the optimal value (Sutton and Barto [1998]).

Other techniques were introduced in Auer et al. [2002] to guarantee optimal convergence. The upper confidence bound (UCB) class of algorithms implement the idea of optimism in the face of uncertainty, as proposed by Lai and Robbins [1985]. This class of algorithm assumes that the average returns for actions are as large as possible based on the observed data. By acting optimistically, one of two things happens:

1. the optimism is justified, and in this case the agent is acting optimally or;

2. the optimism was not justified, and, thus, the agent performs an action that he

believes would give a good reward when, in fact, it does not. If this happens often enough, the agent will learn the true reward of this action and not choose it in the future.

The action selection is based on two factors: the first indicates the current expectation of the average return, while the second represents the confidence that this expectation must be correct. As the agent exploits actions with little confidence, their confidence interval decreases, and the agent will only explore the action again if the return is high.

Many problems are modeled in RL as episodes: they have an initial state and an end state that is always reachable. In other words, the algorithm will always end. However, in other problems there is no end state. In those situations it is usual to employ a subterfuge known as "doubling trick", popularized by Cesa-Bianchi and Lugosi [2006] but that can be traced back to Auer et al. [1995]. Time is partitioned into periods of exponentially increasing lengths. The algorithm starts with a period equal to $\mathcal{T}$. When the period $\mathcal{T}$ ends, the length of $\mathcal{T}$ is doubled. Besson and Kaufmann [2018] showed that a geometric doubling trick conserves (minimax) bounds of the regret to $O\left(\sqrt{\mathcal{T}}\right)$. There are other forms of the "doubling trick", however the geometric doubling described before is the most popular.

2.3.2 Concept Drift

Classic RL algorithms learn optimal decisions in the presence of a stationary environment. However sometimes it is restrictive to consider the environment as stationary. In such situations, RL produces suboptimal decisions. Several authors have proposed changes to RL algorithms to make optimal decisions in a non-stationary environment (Sugiyama and Kawanabe [2012]; Sayed-Mouchaweh and Lughofer [2012]; Gama et al. [2014]; Harel et al. [2014]), and maximize the long-term discounted reward achieved when the underlying environment model changes over time.

One way to cope with non-stationary environments is by adding a concept drift detector. Concept drift refers to the change in relationship between input and output over time. RL learns the distribution of the reward or the transition probabilities, so a concept drift changes the distribution. The system should change its behavior whenever a concept drift is detected.

The system can use the existing distribution as the starting point, and update the model using the most recent historical data, for example, by triggering a new exploration phase. When the system knows the model for each scenario, it can act based on a threshold on the change of distribution behavior. The system checks if the confidence interval of the mean increases above x percent and the noise is above

or below some threshold, and changes among known decision models, e.g. one that applies to a high or low noise environment.

Morocho-Cayamcela et al. [2019] stated that no ML algorithm is universally better than any other based on *No Free Lunch Theorem*, which establishes that "if we average all possible data-generated distributions, every ML algorithm will have the same performance when inferring unobserved data" (Wolpert et al. [1997]). We have selected three RL algorithms to be used in the control loop. The first of these is one of the simplest algorithms called MAB that learns the distribution of rewards for each action (see Section 2.4). The second is an algorithm called Q-learning (Section 2.5) that uses value iteration to learn the distribution of rewards for each possible state and action (which is stored as a matrix called Q-matrix). The latter (explained in Section 2.6) is a variation of the second named double deep Q-learning (DDQL) that uses an approximation function for Q-matrix. This approach allows us to treat states with a continuous space representation as well as extrapolate reward values for states and actions not yet explored. In this work, these learning methods are applied without modification, since we want them to be treated by architecture as a plug-and-play module. Thus our work consisted in modeling of the states, actions, and reward functions applied to each method, as well as defining the hyperparameters.

2.4 Multi-armed bandit – MAB

Multi-armed bandit, introduced by Lai and Robbins [1985], is a special class of the more general sequential optimization problem (Sutton and Barto [1998]). MAB assumes that the feedback is limited, thus the learning process is performed as a trial and error strategy (Kaelbling et al. [1996]; Sutton and Barto [1998]). Historically, the name "bandit" comes from a gambler who plays a slot machine in a casino. The casino does not want to loose money, hence the analogy of the slot machine being a bandit who gets the gambler's money. The problem considers that the player can bet on K machines, so she, with each move (step), pulls the arm of a slot machine. Each time an arm is pulled, a random reward is returned. The K arms are also considered independent of each other. The player's objective is to accumulate the maximum reward in the long run.

Formally, MAB is described as: Let K be the number of arms the player can pull. At each time step t, also called stage, the player (agent) pulls an arm, and a scalar

reward associated with this arm is returned. Let the state of arm i at step t be $x_i(t)$, then, if the agent selects arm $j = m(t)$ at time t, the states are updated as follows:

$$x_i(t+1) = \begin{cases} x_i(t) & i \neq j \\ T_i(x_i(t), \omega) & i = j \end{cases} \tag{2.5}$$

where $T_i(x_i(t), \omega)$ is the transition probability of the $i - th$ arm, based on the state of the i-th arm and a random disturbance ω. The reward received by the player at time t is a function of the current state and a random element: $r_i(x_i(t), \omega)$. The process repeats over a $\mathcal{T}$ steps horizon. In MAB, the agent's goal is to maximize the cumulative discounted reward, as shown below:

$$V = \mathbf{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^t r(x_i(t), \omega) \right] \tag{2.6}$$

where $\mathbf{E}_\pi$ is the expectation obtained when following policy π and γ is a discount factor ($0 < \gamma < 1$). A policy is a decision rule for selected arms as a function of the state of the arms. Gittins showed that there exists an optimal index policy. Thus there is a function that maps the state of each arm to a real number (the "index"), such that the optimal policy is to choose the arm with the highest index at any given time (Gittins et al. [2011]).

Traditionally, MAB uses context-free algorithms that model situations in which no additional information is considered, and the reward depends only on the action taken. In contrast, contextual MAB choose an action to maximize the total rewards considering additional information about the system. In this thesis the problem is approached as a context-free model[1], *i.e.*, we are dealing with stochastic problems where taking an action leads to a reward.

2.4.1 Regret

A common performance measure for bandit algorithms is the total expected regret. Regret measures the absolute difference between the sum of the rewards obtained by the strategy adopted by the agent, and the optimal strategy: regret measures how far the agent is from the optimal strategy. Formally, regret is defined for a fixed round of $\mathcal{T}$ steps as the difference between the reward that would be obtained by using an

[1] A context-free algorithm simplifies the learning, but the exploration phase should be carefully designed to cope with concept drift (Gama et al. [2014]; Harel et al. [2014]). If concept drift is rare, *i.e.* we have a quasi-stationary environment (Nishida [2008]) , then controlling the length of the episodes will suffice.

optimal policy π^*, and the sum of the estimated average reward values $\hat{r}_j(t)$ actually obtained in each step by selecting arm j in step t:

$$R_T = V^{\pi^*} - \sum_{t=1}^{T} \hat{r}_j(t) = r^*T - \sum_{j \in [1,K]}^{T} \hat{r}_j \mathbf{E}[N_j] \tag{2.7}$$

where $\mathbf{E}[N_j]$ is the expected number of times arm j will be selected. In most cases, we can rewrite this as $R_T = T \times \hat{r}^* - \sum_{i=1}^{T} r_j(t)$, where $\hat{r}^* = \max_{i=1,\ldots,K} \hat{r}_j(t)$ (Kuleshov and Precup [2014]). We will use this form when we cannot identify the value of r^*.

2.4.2 MAB algorithm

Algorithm 1 shows the pseudo-code for UCB1 that uses the UCB1 algorithm to select a deterministic policy at each step t proposed by Auer et al. [2002]. We chose UCB1 because Kuleshov and Precup [2014] states that simple heuristics outperform theoretically complex algorithms on most settings, and this search policy does not have many hyperparameters to control the search.

The estimated average value $\hat{r}_j$ of each arm is initialized in line 2. It can be initialized to a constant value (often it is set to zero), or the algorithm uses the average of k pulls of each arm j as the first estimate. In the initialization, the number of times the arm j is pulled ($n_j(t)$) is often set to one for all arms. In time step t, the algorithm selects and plays the arm j that maximizes the index I_j. The environment returns a scalar reward based on the unknown reward distribution associated with the selected arm j[2]. The algorithm uses the "doubling trick" (see Section 2.3.1), since the horizon is not known.

In UCB1, after an arm j is selected and the reward r_t is received, the UCB1 index, the average reward and number of selections of arm j are updated using the following equations:

$$I_j(t) = \hat{r}_j(t) + \sqrt{\frac{2\log(t)}{n_j(t)}} \tag{2.8}$$

$$\hat{r}_j(t) = \frac{n_j(t-1) \times \hat{r}_j(t-1) + r_t}{n_j(t)} \tag{2.9}$$

[2]In this thesis a single action is performed at each step, and the reward is associated only with the action taken (arm). In combinatorial multi-armed bandit (CMAB), several actions can be performed and, likewise, the rewards can be obtained from the arms thrown. We will not address CMAB problems in this thesis.

Algorithm 1 UCB algorithm

1: **function** UCB(K)
2: Initialize $\hat{r}_j, n_j, \forall j \in [1, K]$ ▷ K: number of arms; n_i: number of times the arm i was pulled
3: Set the episode horizon $\mathcal{T} \leftarrow 1$
4: **while** $True$ **do**
5: **for** $t \in [1, \mathcal{T}]$ **do**:
6: $j = \max_j I_j(t-1)$ ▷ Select the arm j that maximizes the index
7: Take action j ▷ Play arm j
8: Observe the reward $r = r_j(t)$ from arm j
9: update n_j, $\hat{r}_j$, and I_j
10: $\mathcal{T} \leftarrow 2 \times \mathcal{T}$
11: $n_j \leftarrow 1, \forall j \in [1, K]$

$$n_j(t) = n_j(t-1) + 1 \tag{2.10}$$

$I_i(t)$, $n_i(t)$ and $\hat{r}_i(t)$ retain the previous values, for $i \neq j$. $\hat{r}_j$ is the average reward obtained from arm j, n_j is the number of times arm j has been pulled so far, and t is the overall number of plays so far.

The index provided by Equation 2.8 is used in line 6. The equation is composed of two terms. The first term records the average reward obtained by selecting arm j, because the algorithm selects the arm that provides largest rewards. The second term indicates the confidence in the average value obtained so far. Note that the confidence grows with the total number of actions taken, but shrinks with the number of times this particular action is tried. In this way, it balances exploitation and exploration.

Equations 2.9 and 2.10 perform the updates shown in line 9. Equation 2.9 updates the average reward of arm j[3] , and Equation 2.10 updates the total number of times the arm j was pulled so far.

2.5 Q-Learning

In Q-learning, the agent attempts to learn the optimal policy from its history of interactions with the environment. A history of an agent is a sequence of state-action-rewards: $< s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, a_3, r_4, s_4 \cdots >$. This means that when the agent was in state s_t, it performed action a_t, which resulted in it receiving a reward

[3]Notice that the equation is only an expansion of the definition of mean –
$\overline{x_n} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{1}{n}\left(x_n + \sum_{i=1}^{n-1} x_i\right) = \frac{1}{n}\left(x_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} x_i\right) = \frac{x_n + (n-1)\overline{x_{n-1}}}{n}$

r_t and transitioning to state s_{t+1}. This history of interactions is treated as a sequence of experiences, where an experience (or episode) is a tuple $\{s_t, a_t, r_t, s_{t+1}\}$, as shown in Figure 2.5. The agent's goal is to maximize the value of the experiences, which is usually the discounted reward.

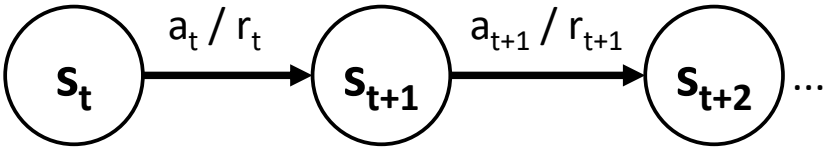

Figure 2.5: Q-Learning experience.

Q-learning learns an optimal policy no matter which policy the agent is actually following, *i.e.*, which action the agent selects for any state s, as long as there is no bound on the number of times it tries an action in any state, *i.e.*, it does not always perform the same subset of actions in a state.

If the action and state spaces are finite, the value of the action learned by Q-learning, named Q-value, is updated using immediate reward and discounted reward. The agent maintains a look-up table of $Q(S, A)$, where S is the set of states, and A is the set of actions. Q-learning uses temporal differences to estimate the value of the optimal state value $Q^*(s, a)$. Denoting a state by $s \in S$, an action by $a \in A$, a reward by $r \in \mathbb{R}$, the learning rate by α and the discount factor by γ, at time $t+1$, the Q-value of a chosen action in a state–action pair at time t is updated as follows in Equation 2.11. A negative reward represents a cost; thus if the reward is maximized, then cost is reduced. We refer to cost as a negative reward henceforth.

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a' \in A} Q_t(s_{t+1}, a') - Q_t(s_t, a_t) \right] \tag{2.11}$$

The learning rate α can assume values between zero and one ($0 \leq \alpha \leq 1$), and it places a limit on how quickly learning occurs (Szepesvári [2010]). If this parameter is set too low, the system will take a long time to learn, while if it is set too high, it will cause the Q-values to never converge to optimal values (Gummeson et al. [2010]). If $\alpha = 1$, the agent forgets its Q-value, replacing it with the most recent reward. γ is a discount factor, which assumes values $0 \leq \gamma \leq 1$ (Szepesvári [2010]; Sugiyama [2015]). The discount factor determines the emphasis over past rewards. Setting this parameter low will optimize for immediate rewards, while setting this parameter high will place more importance on past rewards.

The term inside the brackets in Equation 2.11 is called the temporal difference learning (TD) update. It allows the algorithm to learn directly from raw experience without a model of the environment's dynamics (Sutton and Barto [1998]). It is called temporal-difference because it estimates based on past learned estimates and on the immediate reward received. Notice that the term $Q_t(s_{t+1}, a_{t+1})$ is an estimation from previous iterations on the expected value of the station-action pair in $t+1$, *i.e.*, it bootstraps using the current estimate Q_t instead of the true Q^*.

To simplify our notation, we drop the t subscript in all equations henceforth, and $t+1$ is represented by an apostrophe. $Q(s, a)$ is called an action-value function that is a function of the state and the action taken by the agent. The RL algorithm approximates an optimal policy π^*. It chooses the action in Equation 2.11 that maximizes the Q-value that approximates $Q^*(s, a)$.

This procedure works even if the initial Q-values are unknown (Sutton and Barto [1998]). Q-learning uses an incremental dynamic programming approach (Watkins [1989]), since the optimal policy is learned step by step. Therefore, Q-learning selects a greedy action a_t at time t when in state s, using Equation 2.12, and after that uses 2.11 to update the agent's knowledge about the action taken.

$$a_t = \arg\max_{a \in A(s)} Q_t(s, a) \tag{2.12}$$

Algorithm 2 shows the pseudo-code for Q-learning using the ϵ-greedy policy and the doubling trick (see Section 2.3.1). The ϵ–greedy strategy is shown in line 6, where χ is a random variable. Line 9 shows that the reward used by the agent is some function $f : \mathbb{R}^{|D|} \to \mathbb{R}$, computed over the reward r_t^i of the devices controlled by it. Line 10 updates the estimated value of action-value function $Q(s, a)$.

Algorithm 2 Q-learning algorithm

1: **function** Q-LEARNING(α, γ, ϵ)
2: Initialize arbitrarily $Q(s = \{s^i\}, a), \forall s^i, i \in D$ (e.g., zeros) ▷ D set of devices
3: Initialize $s \leftarrow \{s_{t=0}^i\}$ ▷ initial state s^i for each device
4: **for each** step of episode **do**:
5: ▷ ϵ–greedy policy
6: $\pi(s) = \begin{cases} \arg\max_{a \in A(s)} Q(s, a) & P(\chi) > \epsilon \\ select\ random\ a \in A(s) & otherwise \end{cases}$
7: Take action $a = \pi(s)$
8: Observe the rewards $r_t = \{r_t^i\}$ and $s' = \{s'^i_t\}$ from environment
9: Calculate $r = f(r_t)$
10: update $Q(s, a)$ using Equation 2.11
11: $s \leftarrow s'$

2.6 Deep Q-Learning (DQL)

DQL is an enhanced version of Q-learning. It uses a function to approximate the Q-value, thus better dealing with large numbers of states and actions. Recall that Q-learning uses a matrix (Q-matrix) to find the maximum expected future return of an action given a current state. The Q-matrix has two problems: (1) it is not scalable, since its dimensions increase with the number of states and actions of the system; (2) this matrix becomes quite sparse, since a position is filled only when the agent visits the corresponding state and performs the corresponding action; and (3) it only works for discrete state-action spaces. To solve the problems above, one can use a more appropriate representation, such as an approximation function.

DQL uses a neural network as an approximation function for the Q-matrix. The neural network requires less memory than the Q–matrix in large models, and also can provide insights about non-visited state-action positions by extrapolating from known positions. To understand how DQL works, we recall the Q-value function, which is shown below in Equation 2.13 with some terms highlighted. Table 2.1 shows the symbols used in this section, and a brief description to clarify them.

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha[\underbrace{r_t + \gamma \max_{a' \in A} Q_t(s_{t+1}, a_{t+1})}_{\text{TD-term}} - \underbrace{Q_t(s_t, a_t)}_{\text{Q-term}}] \tag{2.13}$$

The above equation in DQL is replaced by Equation 2.14. The Q-term and the TD-term are estimated separately using two different neural networks, often called the Target-network and Q-Network (Mnih et al. [2015]; He et al. [2017]; Atallah et al. [2017]). Each neural network has its own weights and biases, called θ_t and θ_t^-, respectively.

$$Q_{t+1}(s_t, a_t, \theta_t) \leftarrow Q_t(s_t, a_t, \theta_t) + \alpha \left[TD_t(s_{t+1}, a_{t+1}, r_t, \theta_t^-) - Q_t(s_t, a_t, \theta_t)\right] \tag{2.14}$$

To improve the stability of the model, the weights and biases θ^- should be obtained many iterations before the calculation of the new θ. Therefore the Target-network parameters are frozen for a period of time, and are updated after n iterations with the parameters of the Q-Network. Thus both networks share the same model, but with a temporal displacement of the weights' values. The DQL algorithm convergence was empirically proven in Mnih et al. [2013].

Another technique to improve the stability of both networks is experience replay

(Mnih et al. [2015]; He et al. [2017]; Atallah et al. [2017]). According to Mnih et al. [2015], experience replay eliminates the problem of forgetting previous experiences. The algorithm is trained using a set of experiences (the size of this set is defined by the user), which is called *replay memory*. The replay memory is a circular memory that stores the tuples $\langle\ s, a, r, s'\ \rangle$, where s is the state of the agent, a is the action that was taken in the state s by the agent, r is the immediate reward received in state s for action a, and s' is the next state of the agent after the action is taken. Each time the agent trains the Q-network, it selects a number of experiences from the replay memory. Random mini-batches of the experience replay's tuples are used to calculate $Q(s, a, \theta)$. This procedure improves variance because it increases the number of examples in training. Also, it reduces the correlation between experiences, since the batch is random. Because of that, the selection may not be biased towards recent values, therefore reducing the variance of the updates.

Algorithm 3 shows the pseudo-code for DQL with experience replay, where N is the size of the experience replay buffer and K is the mini-batch size. The algorithm implements an ϵ–greedy strategy in the same way as shown in Algorithm 2. The values of ϵ are updated when the episode finishes. If the system is stationary, then decreasing ϵ is a good strategy, because as times passes, it is better to exploit more to collect better rewards. However, because the wireless environment is non-stationary, we did not decay the ϵ. This is a good line of research for a future work: find a way to decay the ϵ in order to improve the accumulated reward, but detect when to increase the exploration due to an environmental change. Notice that the Q-network is updated in each step in the algorithm's for loop, while the Target-network is updated only after k steps.

In DQL, the Target-network $\hat{Q}$ selects each action, however this may result in an overestimation of the Q-value in line 16. To solve this problem, Hasselt [2010] uses the value of Q-network to choose the action a' in line 16, and use the Target-network $\hat{Q}$ to obtain the Q-value estimation. Thus the value of TD_t is obtained using $r + \gamma\hat{Q}\left(s', \arg\max_{a'} Q(s', a', \theta), \theta^{-}\right)$. Because of this double estimator, this method is called double deep Q-learning.

2.7 Frameworks for Reinforcement Learning

This section describes some of the RL frameworks that implement classical and deep RL methods. An RL framework offers building blocks for designing, training and validating RL implementations, through a high level programming interface. Each

Algorithm 3 ϵ-greedy DQL with experience replay and Target-network

function DQL($N, \epsilon_0, \alpha, \gamma, K, k$)
  Initialize replay memory $\mathcal{D}$ to capacity N
  Initialize state s
  Initialize the size of the episode $\mathcal{T} = 2$
  Initialize Q-Network $Q(s, a, \theta)$ with weights θ
  Initialize Target-network $\hat{Q}(s, a, \theta^-)$ with weights $\theta^- = \theta$
  $\epsilon = \epsilon_0$
  while True **do**
    for each step t of episode $\mathcal{T}$ **do**:
      $$\pi(s) = \begin{cases} \arg\max_{a \in A(s)} Q(s, a, \theta) & P(\chi) > \epsilon \\ select\ random\ a \in A(s) & otherwise \end{cases}$$
      Take action $a = \pi(s)$
      Observe the reward $r = r_t$ and the next state $s' = s_{t+1}$ from environment
      Store $\langle s, a, r, s' \rangle$ in $\mathcal{D}$
      Update the parameters in Q-network:
        Sample random mini-batch of K transitions $\langle\ s, a, r, s'\ \rangle$ from $\mathcal{D}$
        $y_t = TD(s', a', r, \theta^-) = r + \gamma \max_{a'} \hat{Q}(s', a', \theta^-)$ for each sample
        Perform a gradient descent step on $\mathcal{L}(\theta)$ to update θ using α
      Set $s = s'$
      Every k steps reset $\hat{Q}$ making $\theta^- = \theta$
    $\mathcal{T} \leftarrow 2 \times \mathcal{T}$
    Decay $\epsilon = f(\epsilon_0, \mathcal{T})$

framework is built in a different manner for different purposes. There are only a few frameworks that implement basic RL, i.e, non-deep learning implementations. Table 2.2 summarizes the frameworks we evaluated, and the rest of the section contains a brief description of those.

Table 2.2 shows the name of the framework, its status, and programming languages it supports. The table also shows the if the framework implements the RL methods we are using in the thesis. The last column shows if the framework uses the de facto environment, called Gym (https://gym.openai.com/). Gym is a toolkit deployed by OpenAI for developing and comparing RL algorithms. It provides a simulated environment for control theory problems from the classic RL literature (e.g. Cart pole, Pendulum etc), continuous control tasks (e.g. Lunar lander, bipedal walker, MuJoCo, etc), robotics simulations, and games (e.g. Atari 2600 games).

RL Toolbox is a C++ based, open-source framework for all kinds of RL algorithms. This toolbox should be available in http://www.igi.tu-graz.ac.at/gerhard/ril-toolbox/general/overview.html, but the page is not active any more (visited in Au-

Table 2.2: Summary of RL frameworks

Framework	Status	Language	Reinforcement Learning Support				Environment
			MAB	Q-Learning	Other policy-based	DQL	
RL Toolbox	Deprecated	C++	✓	✓	✓		Own environment
RL Toolbox	Active	Python				✓	Gym
S-RL Toolbox	Active	Python				✓	Gym
RL Toolbox	Active	Matlab		✓	✓	✓	Matlab and Simulink
PyBrain	Active	Python		✓	✓		Flexcube and ODE Environments
RLLAB	Active	Python			✓		Own environment
GARAGE	Active	Python 3.5+			✓		Gym
KeRLym	not maintened	Python			✓	✓	Gym

gust 2019). We tested it using an archived version. Because it is deprecated, we refrained to use it in our experiments, and also because we would have to refactor the communication with the environment in order to access real APs.

There is another framework called RL Toolbox. Its GitHub repository is available at https://github.com/jjkke88/RL_toolbox. This framework communicates with Gym, so it is easily deployed in simulated environments, including ATARI video games, 2D and 3D robots. However, it lacks communication support with real life sensors and actuators.

S-RL Toolbox (https://s-rl-toolbox.readthedocs.io/en/latest/index.html) implements 10 distributed reinforcement learning (DRL) algorithms, including newer ones such as asynchronous advantage actor critic (A3C). It works with customizable Gym environments for working with robot simulation and also with real robots (Baxter Robot, and Robobo). The communication with other types of devices is not implemented in this framework.

Matlab also has a commercial propriatary RL Toolbox (https://www.mathworks.com/products/reinforcement-learning.html). RL algorithms in this toolbox interact with environments modeled in MATLAB or Simulink. The toolbox comes with common algorithms, such as Q-learning, SARSA, DQL, deep deterministic policy gradients (DDPG), and advantage actor critic (A2C). We did not had access to this module and it is not free, thus it was not used.

PyBrain[4] (Python-Based Reinforcement Learning) is an open-source python library for ML algorithms. It provides value-based (e.g., Q-learning and SARSA) and policy gradient-based (e.g. REINFORCE) RL algorithms. This framework has its own

[4]http://pybrain.org/

environments, but they simulate games and other toy examples used in RL benchmarks. It lacks access to real devices.

RLLAB (Duan et al. [2016]) is a Python RL framework that includes a wide range of continuous control tasks plus implementations of REINFORCE, Truncated Natural Policy Gradient, Reward-Weighted Regression, Relative Entropy Policy Search, trust region policy optimization (TRPO), Cross Entropy Method, Covariance Matrix Adaption Evolution Strategy, and Deep Deterministic Policy Gradient. This framework is deprecated, but was adopted by the research community and now runs as GARAGE (https://github.com/rlworkgroup/garage). The environment is built using Gym, and does not provide access to the real devices needed for our experiments.

KeRLym (KEras Reinforcement Learning gYM agents – https://github.com/osh/kerlym) is a framework using Keras and Gym. It provides two agents: (a) a policy gradient method with a neural policy network and a DQL agent where the Q-function is approximated using a neural network. It is not maintained since 2017. However, its implementation provided us with several insights.

All the frameworks shown in this section were created for simulation environments, mainly for classic examples of RL such as car pole or Atari games. However, these frameworks need deep adaptations to be used in a production network environment. Those concern mainly the operation in a distributed environment, using message passing via networks instead of message passing via shared memory in a single machine.

The main issue is that the states, actions and rewards require data from devices in the networks (e.g. APs and stations). Hence, the framework must cope with issues such as unbounded delays for receiving a message. This usually requires an event-based programming strategy. On the other hand, frameworks for simulated environments assume that the requests will be answered in short times, following an imperative programming model. For that reason, the existing frameworks would require a major rework to operate in a networked environment.

Hence, we developed from scratch the learning framework used in this thesis, called DeepWiFi. We programmed them using well established math and machine learning libraries such as Numpy (https://www.numpy.org/), Pandas (https://pandas.pydata.org/), Tensorflow (https://www.tensorflow.org/) and Keras (https://keras.io/). Our code is available at GitHub (https://github.com/h3dema/deepwifi). The framework has four basic components: the controller, the agent, the model (which is used to implement the value function), and the environment. The controller loads the experiment, creating the agent, the environment, and activating the required devices via commands sent with SSH. The agent implements an interface that executes the learning loop, reads the state of the devices, determines the action according to the implemented

policy, sends to the environment which action to take, receiving the reward and the new state, and based on this updates a value function. The environment is based on a messaging system called *command_ ap*, also developed for this thesis and available at GitHub (https://github.com/h3dema/command_ap). This messaging system handles sending synchronous or asynchronous requests to devices, requesting status information, taking actions, and evaluating the reward.

Chapter 3

Related Work

This chapter shows work related to the two main themes of this thesis: the use of QoE metrics in computer networks, and the use of RL to improve performance in wireless networks. The first section of this chapter shows how QoE is evaluated in QoE-based solutions in wireless networks. The next section describes in more depth how RL is used in wireless networks. Tables classify the application of RL in several fields of wireless network control. Section 3.5 presents a summary of this chapter contents.

3.1 Methodology

We surveyed the state of the art using Google Scholar (https://scholar.google.com.br/). This search engine returns publications of several scientific editors, indexed mostly on two digital libraries: ACM Digital Library (https://dl.acm.org/) and IEEE Xplore Digital Library (https://ieeexplore.ieee.org). Only articles written in English and published in leading peer-reviewed journals were chosen.

The search for related work started on the IEEE Xplore Digital Library. The results were classified by year and a higher priority was assigned to the most recent works. Next, the results were filtered by the number of citations. The search was repeated using the same criteria in the ACM Digital Library platform, restricted to "The ACM Guide to Computing Literature" collection. If the number of articles returned was considered small (subjective criterion), a new search was performed using the same search string in Google Scholar. The search strings used for the two themes are shown below.

1. **QoE**: The search used the following search string:

   (Qoe and measurement) and "computer networks".

This query was refined by selecting the papers with ten or more citations in both publisher platforms, in the last 10 years. The papers that are referenced in more than three papers of the previous searches are also included.

2. **Reinforcement learning using QoE**: This search employed the following keywords:

   "reinforcement learning" and qoe and wireless.

We restricted the search to proposals applied to the wireless medium, because: (1) our research addresses control over wireless networks; and (2) wireless networks have different requirements than cabled networks, due to the medium dynamics, coverage, energy requirements, interference, and mobility.

3.2 Quality of Experience – QoE

This section discusses the application of QoE to improve network performance. We summarize the approaches presented in this section in Tables 3.1, 3.2, and 3.3. The first column classifies the presented proposals, and the last column provides the reference for each proposal. The second column indicates the type of network where the proposal was tested (we did not indicate if it was a simulation or a real experiment). Proposals applied to an IP-based network are marked in the column "IP Network". The application (video, web browsing or other) is indicated in the next column, which is followed by the parameters used in the estimator, and by which metric is being estimated. The column **Data from layer** indicates from which layers of the protocol stack the parameters are obtained. The **No reference model** column, if checked, indicates that the method infers the QoE without any reference to the original data. For example, a "no reference" video QoE metric is calculated without comparing the original video content or its characteristics, e.g. by looking at the throughput only. On the other hand, a referenced method could calculate the similarity between the source and the received streams, e.g., in a video transmission, it could consider the differences from the source video to the displayed video.

In order to make the text more readable, from now on we will employ the following terms: **feature** is an input value for the QoE model, usually some characteristic of the network, user, or flow; **metric** is a combination of features, leading to a QoE estimation; and **hyperparameters** are tunable variables within the model that estimates the QoE.

The following subsections are divided according to the approach taken to obtain the QoE value. The following approaches were found: linear regression, models ob-

Table 3.1: A summary of QoE approaches.

Proposal	Type of Network	Method	Features	Predicts	Data from layer			No ref. model	Ref.
					Link	Network	Application		
Video streaming	CRN	ANFIS	bit error rate, delay and jitter	MOS					Zineb et al. [2015]
	IP network	cross session stateful predictor	throughput and session features	bitrate[1]		✓	✓		Sun et al. [2016]
	INRS audiovisual quality dataset	decision tree ensemble methods, deep learning, and genetic programming	up to 34 features	MOS		✓	✓	✓	Demirbilek and Grégoire [2017]
	IP network	restricted Boltzmann machine	image information	SSIM			✓		Testolin et al. [2014]
	WLAN	random neural network	bit error rate, aggregation size, number of stations, queue length, and maximum number of retransmissions	MOS	✓			✓	Paudel et al. [2014]
	IP network	fuzzy expert system	packet loss, jitter, and packet loss burstiness	MOS			✓		Pokhrel et al. [2013]
	IP network	naïve Bayes, SVM, k-NN, decison tree, random forest, and neural network	viewer gender, frequency of viewing, interest, delay, jitter, loss, conditional loss, motion complexity, and resolution	MOS		✓	✓		Mushtaq et al. [2012]
	IP network	regression	packet loss	MOS[2]		✓			Khorsandroo et al. [2012]
	IPTV network	regression	codec distortion and packet loss degradation	MOS			✓		Yamagishi and Hayashi [2008]

Notes:
[1] The authors consider bitrate as an estimate of the user's QoE. The lower the bitrate, the lower the QoE.
[2] The models return an unbounded and positive MOS.

Table 3.2: A summary of QoE approaches – Part II.

Proposal	Type of Network	Method	Features	Predicts	Data from layer			No ref. model	Ref.
					Link	Network	Application		
Video streaming	IP network	regression	quantization parameter, content type and packet loss rate	MOS		✓	✓	✓	Anegekuh et al. [2015]
	cellular network (3G trace)	interpolation	quality level, frequency, and duration of video freezes	MOS			✓		Claeys et al. [2014a]
	wireless networks	MDP	startup latency, playback fluency, average playback quality, playback smoothness and wireless service cost	MOS			✓		Xing et al. [2014]
		regression	packet error rate, frame rate, and send bitrate	MOS		✓	✓		Khan et al. [2009b, 2010]
	OpenFlow network	lagrangian relaxation-based aggregated cost algorithm	packet loss and delay variation	a cost function based on the bandwidth, the packet loss, and the delay variation of the link		✓			Egilmez et al. [2013]
	IP network	nearest-neighbor heuristic and dynamic time warping as its distance measure	bandwidth, frame rate, and various combinations of stream state measurements (buffer count, number of times application enters in buffer starvation state, etc)	modified and normalized MOS, rating from 0 to 100%		✓	✓	✓	Dalal [2011]; Dalal et al. [2012]
	IPTV network	regression	PSNR	MOS			✓		Issa et al. [2012]
	WiMAX	multilayer NN	delay, jitter, total loss of the video, and the losses referring to each frame type of the MPEG codec	PSNR, MOS, SSIM, VQM		✓	✓		Machado et al. [2011]

Table 3.3: A summary of QoE approaches – Part III.

Proposal	Type of Network	Method	Features	Predicts	Data from layer			No ref. model	Ref.
					Link	Network	Application		
Video streaming	IP network	regression	packet loss rate, send bit rate, and frame rate	MOS		✓	✓		He et al. [2018]
	IP network	gradient boosting	throughput, video frame loss, and video bitrate	MOS		✓		✓	Amour et al. [2017]
	IP network	mapping considering ranges	delay, jitter, and loss	classifies into good, average, and poor quality		✓		✓	Baltoglou et al. [2012]
	UMTS network	ANFIS, and regression	packet error rate, frame rate, and send bitrate	MOS		✓	✓		Khan et al. [2009a]
Web browsing	WLAN	SVR	medium busy, and bitrate	MOS	✓			✓	Hora et al. [2016]
	IP network	regression	packet loss rate, page load time, and throughput	opinion score: [0,10]		✓	✓		Shaikh et al. [2010]
Other (Audio)	wireless networks	NN, SVR, decision trees, and gaussian naïve Bayes	packet loss, delay, and packet interarrival	MOS	✓	✓		✓	Papadopouli et al. [2015]
	IP network	regression	network bandwidth, and packet loss rate	MOS[1]		✓		✓	Reichl et al. [2010]
	UMTS network	e-model (Bergstra and Middelburg [2003])	QoS-based features	MOS	✓	✓		✓	Hoßfeld and Binzenhöfer [2008]
Other (Remote Desktop Access)	IP network	polynomial regression, and bootstrap sampling	QoS features (available bandwidth and packet loss), and QoE features (the number of video impairment and control lag events, and the selected encoding 's bandwidth consumption for the user task)	quality of application		✓	✓	✓	Calyam et al. [2009]
Other (audio and file transfer)	IP network	regression	transmission rate	MOS		✓		✓	He et al. [2018]
	wireless networks	based on multi-stimuli IQX model (Shaikh et al. [2010])	packet error rate, PHY layer data rate, and the effective data rate seen by the user	MOS		✓	✓	✓	Chenji et al. [2015]

Notes:
[1] The model returns an unbounded and positive MOS.

tained from studies on user behavior (which we call model-based), fuzzy logic, machine learning, or approaches that validated the QoE with users.

Linear regression-based approaches

Many authors predict the QoE by performing a linear regression over features collected at the user's device, and also in the core network, using the least squares method. These parameters are mainly related to the physical, network, and application layers.

Calyam et al. [2009] use QoS and quality of application features to infer the QoE in remote desktop applications that employ encoding adaptation. Thus, a better quality of experience implies selecting the encoding that utilizes reduces bandwidth consumption, and that produces the least number of video impairment, and control lag events. The authors use a polynomial regression to predict the user MOS, and validated their result with actual users. Their proposal outperforms in most cases a network adaptation method that uses only network features (such as available bandwidth).

There are several approaches to classify the video based on its content. In Khan et al. [2009b], the authors use the awareness of the video content to improve the MOS estimation for MPEG-4 streams in wireless networks. The content is divided into three types based on the amount of change in the image: slight movement, gentle walking, and rapid movement. The authors analyze the influence of network-level data (packet error rate), and application-level data (frame rate and send bitrate) on MOS over the three groups. They proposed one regression for each group, with the worst determination coefficient equal to 85.72%. In a later work, the authors improve the regression equations, with the lowest coefficient of determination of 90.27% (Khan et al. [2010]). They employed adaptive neural fuzzy inference system (ANFIS), which combines the advantages of a neural network and those of fuzzy systems. Results show good performance, but the choice of parameters is crucial, and ANFIS provides worse accuracy than regression. Zineb et al. [2015] also used ANFIS considering bit error rate, delay, and jitter to predict the MOS value in cognitive radios, obtaining a correlation of 0.93 in the test phase. Khan et al. [2009a,b, 2010] and Zineb et al. [2015] use a MOS from a PSNR conversion table provided in EvalVid (Klaue et al. [2003]). Evalvid is a framework that evaluates the quality of video transmitted over a simulated communication network by measuring QoS features, like loss rates, delays, and jitter, and standard video quality metrics, like PSNR and structural similarity index metric (SSIM). All of these approaches need the source video to be classified into one of the three groups mentioned before, and no online method is provided.

Shaikh et al. [2010] correlate application-level (page download time) and network-

level QoS (throughput perceived by the user, and packet loss) with QoE. The authors used two web browsing traces: one obtained in a testbed, and another trace captured by a french asymmetric digital subscriber line (ADSL) provider. Web browsing was evaluated by a panel of users that provided an *opinion score* (OS) in the range of 0 to 10, where 0 is the worst result. Four types of regressions were evaluated: linear, logarithmic, exponential and power law. The linear regression achieved a better correlation between the QoE and the packet loss rate, whereas, with the page load time, the exponential regression was the best, and the logarithmic regression presented the best correlation with the throughput.

Other works evaluate QoE directly from application level features. For example, in Issa et al. [2012], the authors use a metric named PSNR that compares the original image with the received image, generating a value representing the noise level inserted in the received image. The authors evaluated a link between this metric and the user's subjective judgment. However, this work does not perform online ranking or attempts to improve network performance based on its results. Claeys et al. [2014a] consider the influence of the frequency and duration of video freezes on the MOS, and propose a interpolation that output the MOS for a playout of a video with K segments, N quality levels and played with a given quality level. Those approaches, however, require the cooperation of the client application.

Model-based approaches

Some researchers use psychological and behavioral models to predict the value of QoE. These models consider linear, exponential or logarithmic relationships of the features to what is perceived by the user.

Chenji et al. [2015] analyze the tradeoff between complexity and optimality when modelling QoE using an approximation function, and propose a bandwidth allocation scheme. The authors provided two equations to predict the MOS based on the multi-stimuli IQX model[1] (Wang et al. [2017]) for file download, and audio streaming. The results are extrapolations of Shaikh et al. [2010] and Khan et al. [2007], respectively. Both equations are based on the packet error rate, PHY layer data rate, and the effective data rate seen by the user. In their proposal, the AP receives the QoE calculated by each user. The authors divide the QoE optimization problem into two subproblems, one for rate and one for bandwidth, which use the QoE, the bit error rate, the users' signal-to-noise ratio (SNR), and the number of bits per symbol on

[1]The IQX hypothesis considers an exponential interdependency of QoE and QoS (Fiedler et al. [2010]).

the link as restrictions. Both problems are linked using a single feasible region, which considers limitations on application-measured QoE given channel conditions. Thus the subproblems are solved in sequence, obtaining a suboptimal solution, but reducing the complexity of the computation.

Some authors based their approach in psychological studies. The Weber-Fechner's Law was tested by Reichl et al. [2010] using regression. This law is a principle of psychology of perception, which states that the relationship between the magnitude of a physical stimulus and its perceived intensity is logarithmic. The authors evaluated web browsing over cellular networks, using a panel of users of both sexes. They obtained a root mean square error of only 0.063 in a logarithmic regression between the network bandwidth and QoE. Khorsandroo et al. [2012] compared Weber-Fechner's Law with Stevens' Power Law in order to obtain a mapping between MOS and packet loss in a video application. The results show that the power regression outperforms the correlation obtained by a logarithmic equation. However, the evaluations considered only single variable regression. Using only one feature may not capture well the dynamics of the network, therefore newer approaches have used multiparametric methods.

Egilmez et al. [2013] proposed a cost function for QoE-aware routing based on the bandwidth, the packet loss, and the delay variation of the link to improve the performance of a video service. Their proposal relies on the OpenFlow PortStatistics primitive. Their proposal poses some difficulties in a real network implementation because it requires up-to-date and fine grained information about these features. Further, they used a PSNR model (Mohamed and Rubino [2014]) to obtain the MOS, but this approach requires comparing the original video to the received one. This is not feasible in an online environment. Despite the fact that OpenFlow provides flow statistics, it is necessary to cope with the delays caused by PortStatistics, and this mechanism is not reliable because the information is not provided with timestamps. Further, the controller's request can be discarded without notification (Floodlight-dev [2016]). Besides that, their proposal has many hyperparameters that should be manually adjusted, and that does not comply with our premise of automatic control.

Fuzzy logic-based approaches

Three features were used in Pokhrel et al. [2013] – packet loss rate, packet loss and jitter – to infer QoE. The pertinence functions estimate the normalized probability distributions that correlate these features with QoE. The authors noticed that it was also necessary to access the application data using deep packet inspection (DPI) in

order to obtain the QoE. Further, the *fuzzy* system might not be generalized to another configuration, because the membership functions are specified for a given network.

Machine learning-based approaches

Many proposals use machine learning to derive a relation between the features obtained at the client or in the network with the QoE. Several methods are applied such as multi-layer perceptron network (MLP), k-nearest neighbors (k-NN), gradient descent (GD), restricted Boltzmann machine, MDP, naïve Bayes, support vector machines (SVM), random forest, etc.

Dalal [2011] and Dalal et al. [2012] evaluated the QoE of video streams using UDP and TCP protocols. Their proposal can accurately assign user quality ratings between 75% and 87% of the time, using k-NN, and dynamic time warping as its distance measure. The hyperparameters (number of neighbors and the period in the distance metric) are obtained during the training phase using leave-one-out cross-validation. The algorithm evaluates the rating in periods of 15 seconds. The ratings used in the training phase are provided by 22 users. Their predictor is also a "no reference" model.

Machado et al. [2011] trained a MLP to predict the value of four QoE metrics in a video stream: PSNR, MOS, SSIM, and video quality metric (VQM). These MOS metrics are estimated in simulations using EvalVid. They use some features from the network layer, but the neural network was trained using loss information for each MPEG frame type (intra-coded – I, predicted – P, and bi-directional – B) on the application layer. The authors did not discuss how this neural network could be used for network control.

Amour et al. [2017] predict the user QoE using gradient descent boosting considering three network and application features (bandwidth, dropped video frame and video quality). The authors employed a crowdsourcing testbed (a lab-testbed where real users evaluated the videos) to train the predictor. Their proposal also uses RL to select what is the optimal quality of the video segment to download, based on the difference of MOS values. RL is active only when the QoE threshold is reached. This threshold is determined using a cross-entropy method proposed by Priyadarshana and Sofronov [2015].

Some research use a QoE estimators calculated only from network features. For example, Yamagishi and Hayashi [2008] enhance the prediction quality of IPTV using the packet loss rate. Their approach was extended by You et al. [2009] to estimate the video quality using also the burst density, the burst loss ratio, and the number of burst periods. In these approaches, the algorithm employs features such as the

media codec, video distortion, distortion concealment, and transmission impairments. However, the authors did not explain how to obtain this information in production networks. Also, they assume the video uses a fixed codec, which is not commonplace today. Testolin et al. [2014] perform admission control, and resource allocation based on QoE. They created an n-degree polynomial estimation of the SSIM using a restricted Boltzmann machine model, where SSIM is mapped to a MOS. In simulations, their proposal outperforms the rate fairness technique (resources are distributed proportional to the full quality rate of the video) considering the relation between the QoE delivered and the computational costs.

Demirbilek and Grégoire [2017] predict QoE using decision tree ensemble methods, deep learning, and genetic programming. The predictors are trained and validated on the INRS dataset (Demirbilek and Grégoire [2016]). This dataset includes user-generated MOS on video and audio streams, which are transmitted with varying video frame rates, quantization parameters, filters, and network packet loss rates, for up to 160 distinct configurations. The proposed QoE models considered up to 34 application, and network-level features. All machine learning methods perform better with the complete set of features. The random forest model provided the lowest root mean square error (RMSE) (0.340), and highest Pearson correlation coefficient (0.930).

Mushtaq et al. [2012] proposed six classifiers – naïve Bayes, SVM, k-NN, decision tree, random forrest and neural network (NN) – to model the correlation between QoE and QoS. They trained the classifiers using 4-fold validation over a synthetic dataset with varying levels of delay, jitter, and packet loss ratio values. The traffic is scored by a panel of viewers. They used nine features: viewer gender, frequency of viewing, interest, delay, jitter, loss, conditional loss, motion complexity and resolution. Decision tree and random forest performed slightly better than the other models, with a mean absolute error of 0.126 and 0.136 respectively.

Some proposals select the best algorithm and its hyperparameters automatically. Papadopouli et al. [2015] tested some ML regressors: support vector regression (SVR), NN, decision trees, and Gaussian naïve Bayes. Their proposal selects the best ML algorithm using nested cross-validation (Tsamardinos et al. [2015]) and the performance metric, e.g. the average QoE. They applied this method to voice over Internet protocol (VoIP) and one mobile video datasets, but they claim the method can be used with other types of traffic. The proposal outperforms other models like E-model (Bergstra and Middelburg [2003]), PESQ (Recommendation [2001]), WFL (Reichl et al. [2010]), and IQX.

Hora et al. [2016] proposed a QoE metric for web browsing over WiFi. To infer the relationship between WLAN features and QoE, the authors used three sources

of data. First, they used a WLAN testbed with instrumented commodity APs. They passively monitored WLAN features, and evaluated the correlation of the QoE with the link quality, and the medium availability. Then they collected data using the selected features in a bigger testbed (their office), which was used to train the predictor. Finally, they validated the predictor using data from real users. They used measurements on APs deployed in 4,880 residential customers of a large Asian-Pacific ISP, reporting over 23,000 devices to a backend server, collecting a total of 180 million samples. To account for page complexity, Hora et al. ranked sites into three categories - *light*, *average*, and *heavy*. They used only two input features: average PHY rate, and the fraction of busy times. The first one represents the link quality, and the second, the medium availability. The regressor's output is MOS, which ranges from 1 to 5. Their predictors (one SVR was trained for different types of sites) adhere to the MOS provided by a panel of users up to 93% in their validation set.

Human-validated approaches

Some authors evaluated the influence of network features on the perceived value of QoE using volunteers, who indicated on a scale how satisfied they were with the service provided.

Paudel et al. [2014] evaluated the influence of WLAN features on video QoE. The model employs a random neural network. Using simulations, they varied MAC layer features, and features related to signal interference, and used the classifier to verify the influence of such features on the average QoE evaluation. In addition to the QoE, the authors also evaluated the influence of different MAC-level features (bit error rate (BER), frame aggregation, number of competing stations, and traffic load) on the QoS, considering flow features such as delay, jitter and packet loss. This QoE metric is very practical in production environments because it is based only on data known to the AP. Unfortunately, we could not employ it in this thesis because it is not publicly available.

Baltoglou et al. [2012] presented a simple mapping between QoS and video quality in an IPTV application, considering delay, jitter and loss in the network. The results are shown in Table 3.4. The MOS line was computed as the average of multiple users, and grouped into the three classes shown in the table.

We found in the literature proposals that account for the user satisfaction, but did not employ a QoE metric. For example, Souidi et al. [2015] propose a cloud provider selection algorithm that accounts for the user's needs. These needs are translated into key performance indicators. Their proposal considers objective performance indica-

Table 3.4: Video quality in IPTV. Source: Baltoglou et al. [2012]

Parameters\Quality	**Good**	**Average**	**Poor**
Delay	$<150\ ms$	$150 - 200\ ms$	$> 200\ ms$
Jitter	$0 - 20\ ms$	$20 - 50\ ms$	$> 50\ ms$
Loss	$0 - 1\%$	$0.1 - 1\%$	$> 1\%$
MOS	$5 - 4$	$3.5 - 4$	> 3.5

tors (time, performance, etc), the flexibility of the agreement between the client and provider, and the user's expectation (defined by a set of weights for each indicator) as features. A linear programming algorithm sorts the provider for each user. Next, the algorithm greedily allocates a provider to a user, according to the provider rank (given by the previous step) and its availability. Then the allocation process starts again. In another example, subjective experiments were preformed by Stefan Winkler [2003]. He varied the video features (encoders, compression, video format) and the network conditions (packet loss, throughput). Real viewers were subjected to two evaluations: they can only see the resulting video, or they see simultaneously the source and the received video. In both cases, the user ranks the video quality. Using the results, the linear and rank-order correlation between the quality and the features from the stream, and the network are analyzed. However, the authors did not present a predictor that correlates these features to MOS.

3.3 Wireless control using Reinforcement Learning and QoE

This section presents research using RL and QoE in wireless network, and summarizes the main references in Tables 3.5, 3.6 3.7, and 3.8. The summary tables show the reference in the last column. The **Problem** column characterizes the type of network problem. The second column – **Type of Network** – indicates in which wireless network the method was applied to. The third column indicates which reinforcement learning approach was used. The next column identifies if the solution uses SDN. From the fifth to the seventh column, each table shows, respectively, the set of **states** used, the set of **actions** available to the algorithm, and the components of the **reward** function. As all the papers found used simulations, we highlight in the column **Simulator** which simulator was used, when informed by the authors. If it not informed, the column is marked with an "N/A". Then, the next column shows which network device runs the decision algorithm.

Table 3.5: A summary of RL approaches using QoE in networks.

Problem	Type of Network	RL Method	SDN	State	Actions	Reward	Simulator	Control Location	Ref.
Content Delivery	cellular network	Q-learning	No	frame type, the level of the post-encoding (PE) buffer, the level of the MAC buffer, and the channel state	number of scalability layers to transmit, keep or drop from the PE buffer	three different estimated rewards: (1) when a single frame is transmitted or dropped; (2) the number of frames left in the PE buffer; and (3) a boolean indicator set when frames are transmitted from the PE buffer to the MAC buffer	N/A	video encoder	Changuel et al. [2012]
	cellular network	Q-learning	No	buffer status is aggregated into three buffer state zones, and discretized into 4 values	keep the content delivery rate unchanged; increase the content delivery rate by a constant percentage; decrease the content delivery rate by another constant	weighted sum of the client buffer states	OMNeT++[1] using INET framework[2]	content server	Yousaf et al. [2014]
	TCP network	Q-learning	No	quality level to request	quality level to request	a function that associates the current video quality level, the oscillation in quality levels during the video playout, and the buffer starvations	ns-3[3]	client	Claeys et al. [2013, 2014b]
	wireless networks	DQL	No	continuous space: buffer state and channel capacity	download a segment t with visual quality q_t	a function of the benefit of a higher quality q_t of the video, and two negative terms due to quality variations in consecutive frames, and rebuffering events	N/A	HTTP client	Gadaleta et al. [2017]
	wireless networks	DQL	No	the streaming service bandwidth, the client buffer occupancy, user's gender, the interestingness of the following video chunks, etc.[4]	select the bitrate for the next requested video chunk	predicted QoE	N/A[5]	N/A	Liu et al. [2019]

Notes:
[1] https://www.omnetpp.org/ [2] https://inet.omnetpp.org/ [3] https://www.nsnam.org/
[4] Notice that not all features are informed in the paper.
[5] The authors used real traces, but did not inform which ones. The FCC broadband and the 3G/HSDPA mobile datasets are used to simulate network conditions.

Table 3.6: A summary of RL approaches using QoE in networks - Part II.

Problem	Type of Network	RL Method	SDN	State	Actions	Reward	Simulator	Control Location	Ref.
Handover	IEEE 802.21	WoLF-PHC (Bowling and Veloso [2002])	No	stay at the current channel or switch to another available channel	select point of attachment	a function of the difference between the current and the previous (stored) mean user perceived quality (QoE) level, and of time the user has experienced degradations or improvements due to the recent action	ns-2[1]	mobile nodes	Ghahfarokhi and Movahhedinia [2013]
Spectrum handoff	CRN	Q-learning	No	the condition of channel, the arrival rate, and the service time	choose to stay at the current channel or switch to another available channel	the predicted MOS of multimedia transmission, for a handover, composed of: the packet error rate due to channel condition and the packet drop rate when the expected delay of spectrum handover exceeds the delay deadline	N/A	nodes	Wu et al. [2014]
Medium access control	wireless networks	spatial adaptive play	No	-	access the channel or not during the timestep	an utility function that accounts for the node i satisfaction when node j keeps silent	N/A	nodes	Yin et al. [2015a]
Resource Allocation	IoT wireless network	DQL	No	the set of conditions of cache nodes, and the set of transmission rates of content chunks	keep, remove or load a content-chunk	three MOS formulations: file download, video streaming and IPTV, and VoIP	discrete simulator ccnSim[2]	nodes	He et al. [2018]
Routing	WSN	Q-learning	No	routing table including alternative routes according to their quality	the next hop	considers three types of services, and provides a QoE value based on QoS features, such as delay, data rate and loss, and the PSNR value	ns-2	nodes	Matos et al. [2012]; Coutinho et al. [2015]
	wireless networks	Q-learning	No	each router in the route	select the link in each router for each route	an MOS value based on an MLP classifier based on QoS values	OPNET[3]	nodes	Tran et al. [2012]

Notes:
[1] https://www.isi.edu/nsnam/ns/ [2] See more information in Wu et al. [2013]
[3] OPNET is now incorporated into Riverbed's SteelCentral – https://www.riverbed.com/products/steelcentral/opnet.html

Table 3.7: A summary of RL approaches using QoE in networks - Part III.

Problem	Type of Network	RL Method	SDN	State	Actions	Reward	Simulator	Control Location	Ref.
Association control	cellular network	Q-learning	No	dynamic status, capabilities of the connected backhaul links, and the corresponding radio channels	associate user to a small cell	user's QoE formed by a weighted combination of performance indicators	N/A	small cells	Jaber et al. [2016]
	WLAN	gaussian process SARSA	No	overall network state (total network transmission rate), AP state[for each AP] (Number of nodes associated with AP, sum of Tx rate for AP nodes, average packet delay of AP nodes), and "Node To Be Moved" state (throughput, average packet retransmissions)	to which AP the "node to be moved" should be associated with	can use any metric like QoS features (throughput, delay, collisions) or QoE (fairness, priority class, etc)	OPNET	station	Aharony et al. [2005]
Congestion control	wireless networks[2]	POMDP	No	source rate, and the packet loss rate (infers the congestion status of the bottleneck links using observations – acknowledgements per RTT – and QoE feedbacks)	the congestion window updating policy	defined by the QoE measured through the multimedia quality MOS	N/A	user station	Habachi et al. [2012]
Traffic classification	TCP network	friend-or-foe Q-learning (Littman [2001])	No	available link capacity, the application and its target QoE level for each node, the average offered bitrate for each application	selects the most appropriate CoS[1] to be associated to each application instance	the minimum global QoE error, where each agent calculates its error as the difference between the perceived QoE associated with user n when using application m, and the target QoE associated with user n.	N/A	supervi-sor agent (con-troller)	Stefano et al. [2015]
Radio resource management	5G	deep actor-critic	No	number of users, channel conditions, QoS indicators QoS requirements defined by the user, arrival rates, queue lengths, traffic characteristics, device types	most convenient scheduler rule	a combination of QoS features: guaranteed bit rate (GBR), delay, and packet loss rate (PLR)	N/A	BS	Comsa et al. [2018]

Notes:
[1] Class of service.
[2] QoE trace of three months of Microsoft Lync 2010 service.

Table 3.8: A summary of RL approaches using QoE in networks - Part IV.

Problem	Type of Network	RL Method	SDN	State	Actions	Reward	Simulator	Control Location	Ref.
Computation load balancing	fog network	restless MAB	No	—	which fog node to pick	total load of the fog node[2]	N/A	user station	Zhu et al. [2018]
Game streaming	5G	actor-critic deep RL	No	composed by the bitrate, current buffer size, past bitrate, jitter, packet loss, and NACK	select the bitrate	a function of the bitrate, and stalling time	real testbed[4]	edge node	Zhang et al. [2019]
Power control	LTE	Q-learning	No	transmit power is (0) unchanged, (1) increased or (2) decreased	keep the same value, and increase/decrease one-fold or three-folds	a range of five values that depend on thresholds defined over the downlink SINR	N/A	BS	Mismar and Evans [2018][1]
Rate adaptation	LTE	DQL rainbow	No	13 past chunks data: video bitrate, buffer occupancy, chunk size, delay, remaining chunks, and sizes of the next chunk	select the rate	video chunk QoE based on the bitrate, and stalling rate	N/A	node	Liu et al. [2018]
Resource allocation	LTE	episodic Monte Carlo policy iteration	No	timestep, remaining budget at this timestep, the requested resources, and the bundle of offered resources	Select the value to bid	equal to the utility function if agent wins the bit, otherwise zero	SimuLTE, and INET	eNodeB	Harishankar et al. [2019]
Spectrum access	5G CRN	Q-learning	No	two binary values that reflect the interference caused by the SU	select the transmission power, and the transmit bit rate	a function that reflects the MOS based on the state, and action	N/A[3]	SU	Mohammadi and Kwasinski [2018]
Power control, and channel selection	WLAN	MAB	Yes	—	Select channel, and transmission power level	Average perceived web QoE	real testbed	SDN controller	Moura et al. [2019c]

Notes:
[1] The MOS is only used to evaluate the network performance, not in the optimization.
[2] The authors consider that QoE is related to the network latency, which is associated with the queue condition, but they do not optimize a QoE metric.
[3] The authors just inform that they used Monte Carlo simulation.
[4] Tested using a real datacenter in mainland China.

As we can see from the column **Problem**, RL has been successfully applied to various network problems. We observe that the research covers several layers of the OSI model, such as application layer (D-DASH, resource allocation), transport layer (congestion control), routing layer (route selection and traffic classification), and link layer (handover and medium access control). Further, many approaches consider information from several network layers at the same time.

Ghahfarokhi and Movahhedinia [2013] consider that mobile users may have different judgments about the QoE depending on their environmental conditions, and personal and psychological characteristics. They proposed a handover method that accounts for the QoE, and exploits the trajectory of mobile users. The station decides to which point of attachment it should connect when the station perceives a link layer event (e.g., link goes up or down, link is up or down, link is detected) and upon quality degradation. They used WoLF-PHC (*win or learn fast – policy hill climbing*). The agent does not exchange information with other agents, since each agent learns the other agents' policies from its directly observed reward and its average policy matrix. In order to create the learner matrices, the authors had to add to the protocol a new point of attachment report, in order to obtain the information about the neighborhood. The proposal improves the quality of received video traffic especially when compared to the default decision method of IEEE 802.21, having similar signalling overhead. The authors pointed out that the major limitation of the proposal is the convergence time to the steady state, especially for large environments with many mobile users.

Wu et al. [2014] proposed a spectrum handoff for multimedia transmissions in cognitive radio network (CRN). Their queuing model uses QoE to maximize the quality perceived by multimedia clients (the paper tested the model for video transmission). Their proposal considers different aspects of the network, such as multiple interruptions, secondary users' heterogeneous QoE requirements, channel contention among multiple secondary users, packet drop rate due to handoff delay, and channel packet error rate. These features are grouped to generate the QoE perceived by the client, which is then used in the learning algorithm to decide if the agent remains on the channel or changes it.

The Spatial adaptive play (SAP) algorithm addresses medium access control Yin et al. [2015a]. It searches for a Nash equilibrium towards the maximum payoff, which is given by an utility function that accounts for the satisfaction of node i when node j is silent. The nodes exchange information about the channel contention (packet arrival probability, required basic signal-to-interference-plus-noise ratio (SINR), and tolerable access delay), and about the individual satisfaction (internal reward). Actions are selected based on the Boltzmann distribution.

Matos et al. [2012] proposed a framework for wireless mesh network (WMN) routing that takes into account the heterogeneous requirements of different services. Their work was is extended by Coutinho et al. [2015], where the source node also learns the video transmission rate in the case of a video application. Thus, as the selected route depends on the service type, flows from different services can be routed along different paths. The route selection also adjusts to resource availability and QoS constraints. Matos et al. [2012] provided three QoE functions, one for each service (audio, data transfer, and video). These functions map QoS features, such as delay, data rate, and packet loss ratio, and a video quality metric (PSNR), into QoE values that are used as a reward. Neighbor nodes inform the transmitting node about their Q-values. Unfortunately the paper does not show how the node discovers the type of service in a flow. They also do not show how the PSNR value is obtained for the video transmission in a non-simulated environment. One option is to infer it using the method proposed in Reis et al. [2010], however it imposes a great cost to the network, because it needs to identify each P-frame, and only works for MPEG videos.

Some proposals consider the control of the whole network to improve the QoE. DOQAR connects clients to the APs offering the best end-to-end QoE (Tran et al. [2012]). A QoE estimate is transmitted downstream from the server to the AP. The AP consolidates the QoE feedback in an acknowledgment (ACK) message, so that the wireless client updates the Q-value of the link. The action (select the AP) is modeled using a Boltzmann distribution. The client then sends upstream a feedback with its Q-value. Each upstream network router updates its own Q-value using the previous router's[2] calculated Q-value and maximum Q-value.

Jaber et al. [2016] control cell range extension offset (CREO) parameters. CREO is broadcasted by small cells to bias their ranking and attract users to select them. RL optimizes CREO parameters for each performance indicator in a way that it maximizes the system performance and end-to-end user QoE. The proposal uses three performance indicators: capacity, latency, and resilience, but the authors highlight that other indicators, e.g. the level of energy efficiency, the cost per bit, a relative security indicator, etc, can be used as well. The paper presents a case study with one macro-cell having three sectors and 21 small cells in fixed locations. Q-learning maximizes the user's expectation using a weighted combination of the performance indicators. The authors used a multilayer perceptron classifier to provide the QoE estimator used as the agent's reward. The proposal improved capacity (number of satisfied users with

[2]The AP uses the feedback directly from the client.

respect to throughput), latency, and resilience, respectively, by 70%, 9.6% and 14%, at the cost of a 3.3% degradation in cumulative throughput in the wireless network.

Aharony et al. [2005] proposed an association control mechanism for WLAN. In current implementations, the wireless station scans the channels to obtain a lists of all APs in-range and their signal strength (usually the received signal strength indication (RSSI)). The station then chooses to associate with the AP with the strongest RSSI, disregarding the AP's current load. This can cause load imbalances in the APs, thus the APs may provide bad quality of service. The proposal takes advantage of overlapping coverage areas in order to balance the number of users during the association process. It uses Gaussian processes temporal differences (GPTD), as proposed by Engel et al. [2005], to address value function estimation in continuous state spaces. The temporal difference is then used in the state-action-reward-state-action (SARSA) policy iteration method. The reward is a function that weighs QoS and QoE metrics. The authors define a multi-agent system, in which one agent runs on each station. The simulation results showed a reduction on the delay of up to 72% smaller when compared to the same network without the agent.

Learning was also applied to dynamic adaptive streaming for HTML (DASH) adaptation algorithms, in which Deep Q-networks with experience replay were used to speed up the convergence and adaptability of the system (Gadaleta et al. [2017]). The authors used three datasets: a) real traces of HTTP video streaming sessions over long term evolution (LTE) provided by van der Hooft et al. [2016]; b) synthetic traces obtained from the ns-3 simulator; and c) Markovian traces obtained by means of a simple Markov model. Simulations show that, for real traces, D-DASH achieves higher rewards after just a few videos. The D-DASH approach presented less rebuffering events, because it can use the buffered segments when the available capacity drops, and build up the buffer when it is convenient.

A cross-layer approach was proposed by Changuel et al. [2012] to maximize the received video quality, while accounting for the characteristics of the transmitted content, and channel variations. Q-learning defines the number of scalability layers to transmit, and if the frame must be kept or dropped from the post-encoding (PE) buffer.

Another content delivery approach was proposed by Yousaf et al. [2014], where the Q-learning algorithm is embedded into the fair scheduler for video pacing (FSVP) application-level scheduler. This scheduler provides a special form of rate limitation, where traffic delivery is slowed down in order to provide a "just-in-time" delivery. The goal is to prevent against buffer starvation and to ensure that enough video data is available in the playout buffer in advance to the current playtime. This enables a smooth playback during adverse network conditions, for example congestion in the

backhaul or in the access links. The learning algorithm orchestrates between concurrent video flows with reference to the size of the user playout buffer and prioritizes flows in accordance with the user class and the current playout buffer size (the size is classified into three intervals). The proposal reduced the average starvation frequency from 27% of pure TCP to 1% using Q-learning.

Some papers improve the performance of HTTP adaptive bitrate (HAS) traffic based on QoE. In Claeys et al. [2013, 2014b], the HAS streaming client adapts the requested quality level based on the perceived network and device conditions. The papers vary the Q-learning approach: Claeys et al. [2013] uses Q-learning and Claeys et al. [2014b] uses frequency adjusted Q-learning as proposed by Kaisers and Tuyls [2010]. Both papers tested two exploration policies: softmax and VDBE[3]-softmax. Simulations show that both approaches improve video quality metric over traditional heuristics. The simulations are very simple, with only one client, so the impact of concurrent access, burstiness of traffic and routing, as well as connection changes are not accounted for. De Vriendt et al. [2013] intercepts the video client requests to the video server delivered over a radio network, and uses this information to predict the quality of each video segment.

Liu et al. [2019] propose using deep learning to predict the QoE of video transmissions, and also presented a bitrate adaptation mechanism based on DRL for HAS. The authors used five types of features to predict the QoE: text, video, categorial information, continuous information, and sequence data (e.g., bandwidth information). However the paper does not enumerate what are the features for all the types. These features are used to represent the state of the DQL agent, thus the full state representation is also not known. The predicted QoE was used to drive the adaptive bitrate (ABR) algorithm. The Q-function is approximated using a two-layer fully-connected neural network, which receives as input the state, and predicts the action (the bitrate for the next requested video chunk).

Other approaches consider explicit application feedback to control the video coding. For example, Xing et al. [2014] designed a reward function that considers QoS requirements such as the startup latency, playback fluency, average playback quality, playback smoothness and wireless service cost. The proposal allows the client to request different parts of one video segment over different links.

He et al. [2018] proposed green resource allocation based on DQL for Internet of Things (IoT) content delivery. The proposal controls the cache allocation, and the transmission rate. The proposed QoE considers two perspectives: user experience, and

[3]Value-difference based exploration.

network cost. From the user's perspective, the proposal uses three MOS models: a file download service model proposed by Song et al. [2016], and the video streaming and VoIP service models proposed by Anegekuh et al. [2015]. From the network's perspective, the proposal considers if the chunk requested by the user is cached or not at the nearest cache node. The learning algorithm uses a combination of these perspectives to learn if a chunk of data must be cached or dropped at each node.

Stefano et al. [2015] allocate classes of service to a flow based on QoE. The selection uses Friend-Q, which is an adaptation of Q-learning for multi-agent environments. This algorithm evaluates the flows based on network features (such as delay, transfer rate, packet loss, etc.), security features (such as those related to data security, privacy, reliability), mobility features (e.g., roaming and delivery performance), as well as explicit user feedback (e.g., based on an online survey). A controller evaluates the QoE and assigns a class of service for each network flow. In this proposal, agents located at the user station report their QoE error (the difference of the expected QoE and the obtained QoE) to the controller, who decides the class allocations, and transmits them to the agents. The use of a controller simplifies learning, which is modeled as fully cooperative (i.e, all nodes receive the same reward). The simulations show that the proposal reduces the QoE error of the network. The main disadvantages of this approach is that the entire network must be QoE-aware, and a central decision point is needed. The authors did not evaluate the overhead of the allocation tasks, nor the scalability of their architecture.

Habachi et al. [2012] proposed a congestion control algorithm for networks containing wireless branches, i.e, clusters of wireless networks connected to a backhaul. It uses a two-level congestion control mechanism, adapting the congestion window size according to the source rate and QoE feedback. Because the hosts (stations or servers) cannot directly observe the loss rate of every other node in the path, the host has to infer the congestion status of the bottleneck link using observations, *i.e.*, it uses the acknowledgements per round-trip time (RTT) to infer congestion. The decision also uses QoE feedbacks as parameters. Therefore, the problem was modelled using a partially observable Markov decision process (POMDP) that is updated in discrete intervals. Their proposal reduces congestion by 10% up to 30% over the same network without the agent.

3.4 Discussion

Some papers shown in this chapter deal with problems similar to ours. We will, in the following paragraphs, highlight what they have in common with our work, and what are the differences.

Mismar and Evans [2018] control the transmission power using RL, however their proposal applies to small LTE cell, and only to voice transmission. Mohammadi and Kwasinski [2018] also control the transmission power, and use QoE to guide the Q-learning algorithm, however their proposal alters the bitrate of the downloaded video and data, while ours selects the channel and the transmission power. Their proposal applies to secondary users in 5G cognitive networks, and the objective is to limit the amount of interference over the primary users below a limit, while our proposal is to maximize the QoE and cope with the co-interference from other APs.

There are many proposals (Egilmez et al. [2013]; Khan et al. [2010, 2009b]; Amour et al. [2017]) that deal with the improvement of the video transmissions using QoE, but none of them use RL. And some, such as Baltoglou et al. [2012]; Issa et al. [2012]; Dalal et al. [2012]; Dalal [2011]; Machado et al. [2011]; Khan et al. [2009a], only measure the QoE but do not use it to improve the video quality.

Xing et al. [2014] improves the quality of the video stream by allowing the client to request different chunks of one segment of the video using different links. Their methods uses policy iteration to find the best action, *i.e.*, select the resolution of chunk for each link. Thus, their proposal is for a multi-link client, which performs load balancing and rate adaptation among the links. They do not control the network to improve the quality.

There are approaches that control the video transmission at the source or the destination, while ours uses the default DASH configuration and alters the user's connection to the network. Claeys et al. [2014a] use RL to improve the video quality at the wireless client, and the learner adapts the client requests to the DASH server, and do not control the network. Amour et al. [2017] also use RL to decide what is the best video quality segment, being very similar to Claeys et al. [2014a]'s proposal but with a different reward function. Also the decision is different, the former proposal defines the value needed, while the latter decides if the quality should increase, decrease or remain the same. Khan et al. [2010] alter the video sender bitrate (SBR) at the video server, according to users' QoE requirement, thus they do not control the network. Also their proposal defines a function that predicts the SBR, however this function is obtained by a regression over a simulation, thus it remains fixed, and, therefore, cannot adapt to network variations that occur in the real world.

He et al. [2018] adopt another approach to improve the video quality. They control the caching of the video segment on the IoT nodes. Thus their approach and ours are different, because our proposal controls the Wi-Fi network conditions, while theirs control where the segment is located in a mesh network.

We found proposals that calculate the web browsing experience but did not try to improve it. Reichl et al. [2010] and Khorsandroo et al. [2012] tested Weber-Fechner's Law for web browsing QoE, while Hora et al. [2016] propose an MOS regressor using machine learning for web traffic, but both did not try to improve the QoE. Shaikh et al. [2010] proposed a correlation between the QoS and QoE for web browsing, but they focused on the impact of QoS features on the user satisfaction without proposing how to improve the QoE. Our work uses the regressor proposed by Hora et al. and proposes a control loop to improve the QoE perceived by the user.

3.5 Summary

This chapter discussed several QoE estimators. It also shows solutions that seek to improve QoE metrics in wireless networks using RL. The research presented in this chapter indicate that there are several approaches to calculate QoE metrics. In this thesis, we have selected from literature some QoE metric applicable to video traffic and web browsing, which were used in our experiments. These metrics will be explained in detail in the following chapters.

Chapter 4 shows an architecture that controls wireless networks in order to improve the user's QoE. The next two chapters show the implementation of this architecture for web browsing and video traffic, respectively. A common trend in the related work shown is the use of simulators. Our work goes one step ahead, implementing the proposals over a real testbed.

Chapter 4

Control loop architecture

This chapter discusses a generic architecture for wireless network control using RL. Section 4.1 shows the general design of the proposed architecture. This architecture uses a closed control loop based on reinforcement learning whose output determines (and learns) the action to be performed on the APs based on the QoE. The first section describes the architecture. The following section instantiates the architecture for Web browsing, followed by a section showing the same architecture for video streams. The last section shows a summary of the results of this chapter.

4.1 The architecture

Figure 4.1 shows in general lines the proposed architecture. It is composed of three layers, which are discussed in detail in the next three subsections.

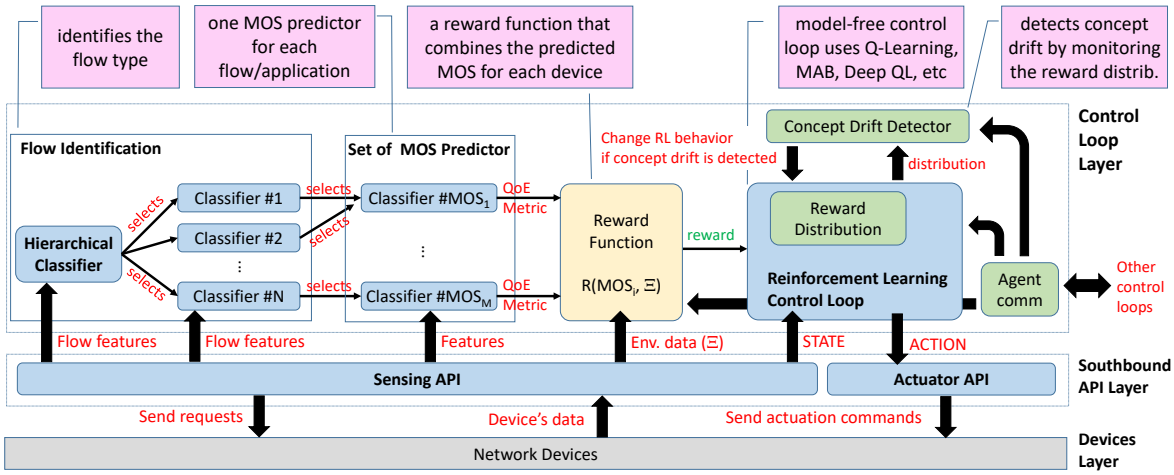

Figure 4.1: Generic architecture to perform automatic control of wireless networks.

4.1.1 Network Devices Layer

The bottom layer (shown in gray) consists of the wireless network devices controlled by the architecture. These devices perform the traffic forwarding/routing for users connected to the network. They can be APs, wireless stations, or Ethernet switches that connect the APs to the Internet.

4.1.2 Southbound API Layer

The intermediate layer consists of one or more APIs, such as OpenFlow, CAPWAP, Ethanol (Moura et al. [2015]**d**), etc., capable of interacting with network devices, sending configuration and control messages, and information requests to these network devices. The communication between the controller and the network devices can be *direct*, for example, an OpenFlow command from the controller to a switch to install a flow in the switch's OpenFlow table, or *indirect*, such as an information request using 802.11k that is relayed by the AP to a station. This layer abstracts the network devices, thus upper layers can profit from a common API, in order to implement control algorithms with higher levels of abstraction.

4.1.3 Control Loop Layer

The upper layer consists of the algorithms that control the network, i.e., it determines the action to be performed on the network devices in order to guarantee a good QoE. In order to do that, the algorithms optimize a metric (defined by the operator). The RL method used by the control loop is treated as a black box, i.e., one can easily exchange one RL. One only must respect the selected method's needs for the feedback signal and the state information.

Notice that the RL module receives the state, and the reward as input, and provides the selected action as output. It is transparent to the RL method, e.g. MAB, Q-learning, etc., how many actions are available or how many features compose a state. If the feature that compose the state is discrete or continuous, it affects how the Q-value is stored, e.g., if all features are discrete and finite, the Q-value can be stored as a matrix. The number of states and action affects the system's memory usage. It can also influence the time needed to explore the state-action space that grows linearly with the number of actions, and linearly with the number of states, which may increase the convergence time.

The following subsections detail the design of the control loop layer. The next subsection shows how the proposed architecture deals with changes in the behavior of

the environment. The following section explains why the proposal has to identify data flows in order to cope with various user's requirements. Then the sections show how this process is done in two steps: flow classification, followed by the selection of the MOS predictor. Finally, we explain the advantages of using a model-free solution as proposed in this thesis.

4.1.3.1 Choice of independent agents

When there are multiple agents involved, it may be necessary for these agents to exchange information with each other, for example by reporting the effect of one agent's action on another agent's performance, to implement a biding operation, etc. Information obtained from other agents may influence the concept drift detection, the control loop behavior, or may influence the reward function as a penalty or reward. The architecture could provide a communication module (not implemented in our thesis, but shown in Figure 4.1), which can be used by the agents to share information about the learning process. For example, the agents can send to a coordinator the gradient used to update of its Q-function, thus the coordinator can update a global Q-function as proposed in A3C (Mnih et al. [2016]) or they can communicate trajectories of experience (sequences of states, actions, and rewards) to a centralized learner, such as in IMPALA (Espeholt et al. [2018]).

In this work, we will focus on a central agent, or in other scenarios multiple agents will infer the effect of their actions directly from environmental measurements. This way this module will not be implemented, however it may be important in a multi-agent RL, because exchange information may improve collaboration (Yau et al. [2010]). In multi-agent systems, multi-agents taking actions on the same environment can make it not stationary (Yang and Gu [2004]).

The current architecture therefore allows wireless devices to be monolithically controlled from a single control loop or multiple control loops (*i.e.* agents). For zero-sum games (*e.g.*, a scenario with multiple APs co-interfering), each agent tries to maximize its payoffs in the worst situation, thus the agent's interests in the game are opposite. In order to get agents' cooperation, each agent can (1) infer from the environment the result from the other agents, and maximize its expected value in the face of the worst-possible action choice of the opponent; (2) maintain Q-values for both the learner itself and other players or maintain the beliefs of other players' policies based on their past play; or (3) use the effect of the action in the reward function as a penalty or reward (Aref et al. [2017]; Stefano et al. [2015]; Schwartz [2014]; Petrangeli et al. [2014]; Buşoniu et al. [2010]; Busoniu et al. [2006]; Tan [1993]).

4.1.3.2 Coping with a dynamic environment

RL learns a distribution of the returns obtained by the actions taken in each state. If the process is stationary, this distribution does not vary. However, if the process is not stationary, a behavior learned in one period may not be valid in another period. To address this, the control loops employ a drift detector concept. This detector monitors changes in the reward distribution. When these changes are detected, the learned model should be updated. We believe that proactive reactions based on concept drifts will make the system better, as the control loop adapts to a new situation to maintain a good QoE. In this thesis we do not perform experiments with this approach, due to a lack of time, leaving it for future work.

4.1.3.3 Flow identification

The control system must identify the type of each flow, and consequently select the best QoE predictor to be used in the learning process (see the next section). Christophides et al. [2018] analyzing web QoE, concluded that building a one-size-fits-all model is doomed to fail, therefore, an approach that considers different, and multiple MOS predictors for a flow is better, and that's what our architecture does. The flow identification has two stages that:

1. *Classifies the flow type*: In this stage, a hierarchical classifier identifies the type and subtype of the flow. For example, video streaming and web browsing are types of a flow. Live action and comedy streaming can be subtypes of a video flow. Such a classification is important because each flow type will require different actions on the network to improve its QoE. As an example, the QoE for web browsing will demand actions mostly to improve the throughput. Meanwhile, for a live conference, the QoE may be centered around providing low latency. Likewise, the subtype of a flow may influence the QoE. As an example, live action movies may require more vivid images and no rebuffering, while in comedy it may be more important to ensure good sound, and some rebuffering is accepted.

2. *Selects a MOS predictor*: Based on the type of flow determined in the previous stage, the control loop selects a MOS predictor that will infer the QoE perceived by the user. Hoßfeld et al. [2016] stated that given the same network conditions, the QoE of different services is often different. Thus the need for different MOS predictors. Also even the same service might need different predictor, for example, Chapter 5 shows three MOS predictors that are selected based on the three types of web pages (*light*, *average*, and *heavy*).

A *hierarchical classifier* is used in our proposal because there may be flows of certain subtypes with different SLAs. Consider for example the flow classifier proposed by Li and Moore [2007]. This classifier identifies different types of traffic such as Web-browsing, Mail, FTP, Attack, Peer-to-Peer, Database, Service, and Interactive. Applying this classifier corresponds to the first level proposed in the architecture. However, for web traffic, if we use the predictor proposed by Hora et al. [2016], a second level is needed to discriminate which of the three MOS predictors should be used for a given stream, such as the site classifier proposed in Section 5.2.

This thesis does not investigate the first step of flow classification, since this is already a well-studied topic, and many effective classifiers already exist in the literature. We encourage the interested reader to refer first to two surveys for an overview of traffic classification: Goli and Ambika [2018] and Nguyen and Armitage [2008]. For traffic classification of encrypted traffic, please refer to Orsolic et al. [2017]; Dubin et al. [2017]

4.1.3.4 The choice of a model-free approach

The proposed architecture may seem a bit counter-intuitive, since it derives actions from a model-free approach. However, there are a number of papers in the literature (Malik et al. [2015]; Seddiki et al. [2014]; Daneshgaran et al. [2008]; He and Pung [2006]; Bianchi [2000]) that model the QoS of wireless links. Those could be a starting point for a control loop. However, we chose not to use them because of their complexity and also because of the need to map QoE metrics into QoS metrics.

Models of the IEEE 802.11 channel dynamics, such as those cited above, usually model aspects such as throughput and delay in specific situations (e.g. on a saturated channel, or on a non-saturated channel with a fixed packet generation rate). Beyond that, hard to model physical effects such as multipath fading can also significantly affect the performance of the network in dense environments. Another aspect of channel dynamic models is the fact that they rely on optimization models, and those are expensive to re-run. Meanwhile, RL is less expensive to run, since the updates of the state transitions are usually performed locally (i.e. only at the current state).

Next, channel dynamics models are able to infer the QoS of the channel, and not the QoE. QoE depends on user perception, so in order to use WiFi channel models in a QoE-aware control loop, one would first have to derive models that map QoS into QoE perception.

Instead of relying on two separate models (i.e. a channel dynamics model and a QoS to QoE mapping model), this work directly maps the effect of actuation into the end-user QoE. Since there are no pre-existing models in the literature to map the QoE

of an application into actionable parameter changes, we adopted a model-free approach in which the system learns the relation of states and actions.

We opted in our work to adopt a prototyping approach to test our proposal in a real testbed. Thus in the next two sections we show how the proposed architecture is instantiated to generate the agents that will be tested in two types of network applications: web browsing and video streaming. As Morocho-Cayamcela et al. [2019] point out, using data generated in simulators is not a good approach, because the agent will learn the rules that were programmed in the simulator, which may not reflect the unobservable features of the real world. However, it may be possible to use the simulated environment to accelerate agent learning as long as the simulator provides a good approximation to the real world. Therefore, due to this and the lack of time to perform also good simulations, we chose to perform all the experiments presented in Chapters 5 and 6 only in a real testbed. Combining the use of simulation and experimentation might be addressed in a future work.

Network performance is impacted by a variety of factors, and QoE is a metric that considers end-to-end service, *i.e.*, from the server up to the presentation to the user. The last mile connection usually has lower bandwidth (if compared to the core network), and a wireless connection presents more performance issues than a wired one (Sundaresan et al. [2016]), because if the connection is wired, it will be provided by a switch that has higher throughput (tens of Gbps instead of hundreds of Mbps), and does not suffer the same performance issues as the wireless connection (*e.g.* shared medium, higher interference, etc.). Notice also that most Internet providers act as a content provider by hosting, near to the users, the content server (Herbaut et al. [2016]; Jiang et al. [2009]) from companies like Akamai, and FAANG (Facebook, Amazon, Apple, Netflix and Google), just to list the bigger ones (Mitra et al. [2016]). Thus in this thesis we focus only in the wireless connection to the user station.

4.2 Architecture applied to Web QoE

Chapter 5 shows the proposed closed loop control system for wireless networks, using RL and SDN to improve the QoE of Web applications. To do that we instantiated the proposed architecture to deal with web browsing, as shown in Figure 4.2. Some itens are greyed out, because they are not used for Web control. It is worth noticing that the control loop does not require changes to the user's applications and device. The four steps of the control loop were instantiated as follows:

1. **Flow identification**.

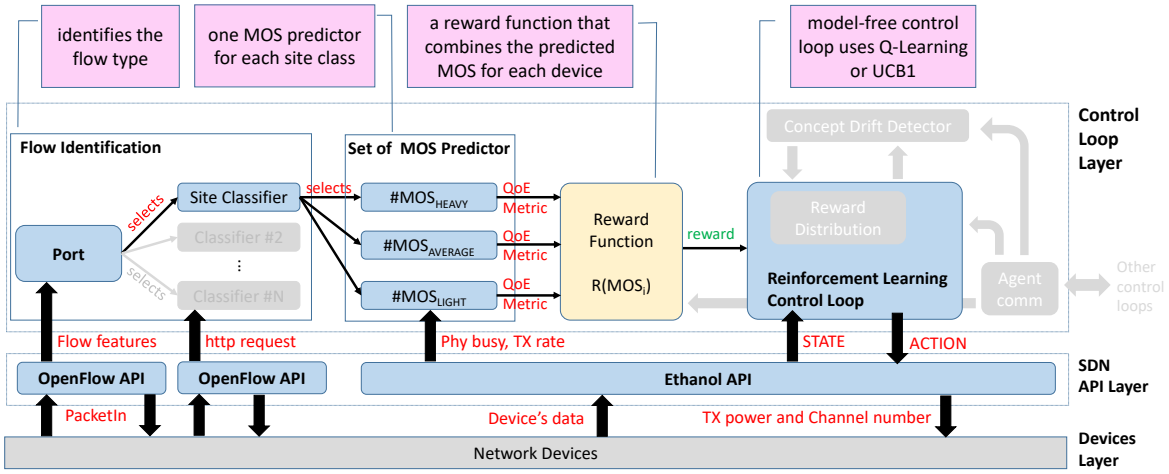

Figure 4.2: Architecture for automatic control of web flows in wireless networks.

In this study, the first step of flow identification is based on the ports of the TCP flow (ports 80, 8080 and 443 for web flows). More complex approaches proposed in the literature can be employed, but this was left as future work.

For the second stage of flow classification (the identification of the flow type), we proposed a new site classifier using transductive support vector machine algorithm based on spectral clustering (TSVMSC). The *site classifier* returns which type of site was accessed by the user (*light*, *average* or *heavy*). The online site classification is described in details in Section 5.2.

The site classification model is recalculated periodically, using an off-line classifier, as shown in Section 5.2. This update is necessary because the user browses new sites, and that information can improve the *site classifier*. On-line learning was avoided due to the need of fast responses, as described in Section 5.2.

2. **MOS predictor**.

   Our control loop uses the MOS predictor proposed by Hora et al. [2016]. They proposed three predictors based on the type of the site that is accessed by the user. These predictors are shown in Figure 4.2 as $\#MOS_{light}$, $\#MOS_{average}$, and $\#MOS_{heavy}$. We chose this predictor because: (1) it has been extensively evaluated using real traces and surveys with real users; (2) it uses only data obtained in the AP, which implies that it can run with regular wireless stations (no modification required), and (3) it maps PHY parameters into QoE with good accuracy, which is useful for the RL algorithm.

3. **Reward Function**.

The *reward function* is based on the values provided by the MOS predictor. Since the control loop can simultaneously manage multiple APs, and each AP can have multiple stations connected and surfing the Internet, this function returns the MOS for each AP. The reward is the average of the MOS for all the APs. Section 5.4.3 details the reward function.

4. **Reinforcement learning control loop**.

   The controller uses RL to select the best transmission power, and wireless channel for each controlled AP. We evaluated two types of RL: Multi-Armed Bandit (UCB1), and Q-Learning. The control loop is presented in details in Section 5.4.

4.3 Architecture applied to Video QoE

For video QoE, we use a REST-based southbound interface specifically designed to control IEEE 802.11 APs, in order to demonstrate that the architecture proposed in Section 4 can use several southbound interfaces.

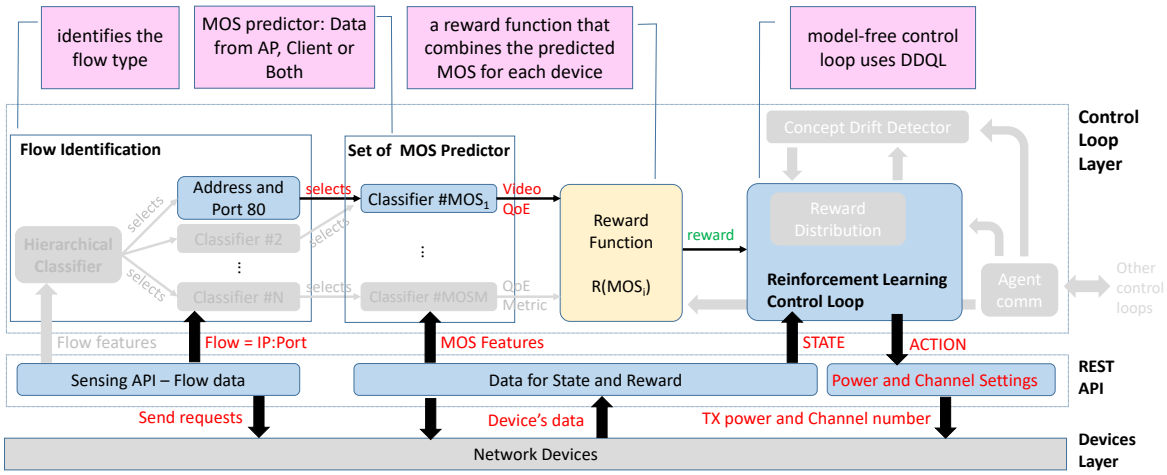

Figure 4.3: Architecture for automatic control of video flows in wireless networks.

Figure 4.3 shows the instance used to control the wireless network. Some areas in the figure are grayed out because they will not be covered in the control proposed in Chapter 6. The network devices layer is composed by the APs controlled by the control loop. A Python control loop communicates with the *Southbound API* layer. In this instance of the architecture, the *Southbound API* was implemented as a REST API that uses HTTPS to exchange messages, and was installed on the APs, and the controller. This interface, and the program that run in the APs are available in https://github.com/h3dema/command_ap.

The control system runs on the control plane of the wireless network. The control loop occurs in four steps:

1. **Flow identification**.

   As we know what kind of traffic will be generated at the stations during our experiments, it will not be necessary to identify the type of traffic (e.g., discriminate between web and video traffic), or select a specific predictor for a video type (e.g., a QoE for IPTV or DASH).

   In our implementation, we use a simple classification system that only identifies whether the flow is directed to our video server or not, and it is only based on the server address and the HTTP port (port 80).

2. **MOS predictor**.

   In a network we identify two main types elements: the hosts (servers and stations) and the nodes of the network (network devices). Based on the location that the features are collect to feed the MOS predictor, three different approaches can be identified in the literature:

   a) The first approach uses only information that can be obtained from the network devices managed by the controller (Paudel et al. [2014]; Kim et al. [2012, 2010]). Thus, this kind of solution does not require changes on the clients nor the servers connected to the network. Because of that, this solution is simpler to deploy on production systems.

   b) In the second approach, only the hosts provide information to compose the current QoE (Mao et al. [2017]; Yin et al. [2015b]; Testolin et al. [2014]; Piamrat et al. [2009]). This feedback can be done through user intervention, who responds to a questionnaire (e.g. click on the video page), or the information can be collected automatically from the customer's application. User feedback provides a more refined information, as it portrays the user's subjective perception. However, this approach is intrusive (even if the user has the option to respond only if she wishes), and there is no guarantee that the user's response represents her objective assessment of the situation. The server can also be the source of data to compose the QoE. For example, in Mok et al. [2012] a measurement proxy is installed in the DASH server to perform inline measurement of the client's flows, which allows the computation of the flow quality.

c) A hybrid approach uses data from the client and from the network. For example, in Shaikh et al. [2010], they correlate page download time from the application, and network throughput and packet loss to create a QoE metric.

In this thesis, we have adopted only one approach where the features are obtained in the client side. For that, the client software was altered to automatically send information to the controller, which computes the QoE metric. The metric used in the thesis is discussed in Section 6.2.2.

3. **Reward Function**.

   The reward function is a crucial part of the system. In the previous case (web), the reward function was the average of the predicted MOS values for the stations in each AP. For video, we want the control loop not only to improve the average MOS, but it also should try to balance the stations' MOS. Suppose there are two stations. We would prefer that both stations have MOS equal to 3, than a MOS of respectively 1 and 5. This function is discussed in detail in Section 6.2.1.

4. **Reinforcement learning control loop**.

   The controller uses RL to select the best transmission power, and wireless channel for each controlled AP. The RL control loop is modeled using DDQL. The control loop is presented in full details in Section 6.3.4.

4.4 Summary

This chapter shows the proposed architecture for handling network flows using QoE as a reward in RL. We show how the architecture can be adapted for wireless network control situations such as web browsing and video streams. The chapter also shows how the middle layer, which abstracts the wireless network, can be built using a wireless SDN platform or an API built specifically for the purpose. The next chapters will present each instantiation of the architecture, together with the experimental results.

Chapter 5

Wireless Control using RL for Web QoE

It is important to provide a good QoE for Web users. For example, Hochstadt et al. [2018] highlighted that 40% of visitors will abandon a website that takes more than 3 seconds to load, and from those, 80% won't return.

This chapter presents a closed loop control for Wi-Fi networks, using RL and SDN, which optimizes the Web QoE. It uses a metric developed by Hora et al. [2016] to estimate the QoE using data collected only from the APs. For that end, we devised a flow classifier, based on TSVMSC, to identify which QoE estimator from Hora et al. [2016] should be used in the control loop. The proposed RL method does not need to know the system model. It follows the changes in the network conditions (adapts to a non-stationary environment), and seeks to continuously improve user satisfaction. To the best of our knowledge, our work is the first that uses SDN to control 802.11-based wireless network using RL using a QoE metric as feedback. Table 5.1 lists the key symbols used in this chapter, and provides a brief explanation of them.

The next section shows the main contributions of this chapter. Section 5.2 describes the site classifier that identifies the type of web flow. It shows how the sites accessed by the users are classified into the three classes used to estimate the QoE. The results of the classifier evaluation are shown in Section 5.3. Section 5.4 presents how the control loop was modeled using Q-learning and UCB1, highlighting the reward function, states, and actions used with these methods. The following section contains the control loop evaluation results in three real scenarios. It compares the results using both learning algorithms (using Q-learning and UCB1) with the baselines. The last two sections of this chapter discuss the results obtained, and then summarize this chapter.

Table 5.1: Key symbols used in the chapter.

Symbol	Description
ν	number of features of the TSVMSC classifier
X	the features for the labeled examples
Y	the labels of the labeled examples
$\grave{n}$	number of unlabeled examples in the TSVMSC classification
$\mathbf{x}_i^*$	the vector of features on the unlabeled data i
X^*	matrix $\mathbb{R}^{\grave{n}\times\nu}$ of unlabeled data $\mathbf{x}_i$ as rows
σ	parameter that controls the width of the neighborhood in the distance matrix generation
C	regularization parameter for labeled data in the TSVMSC classification
C^*	regularization parameter for unlabeled data in the TSVMSC classification
ξ_i, ξ_i^*	slack relaxation variables for the transductive support vector machine (TSVM)
c_n	centroids for the labeled data, where the index $n \in \{light, average, heavy\}$
c_u	centroids for the unlabeled data, where the index $u \in \{1, \ldots, K\}$
K	number of clusters discovered by the spectral clustering procedure
c	the loop parameter that represents the class evaluated
$\mathbf{s}$, s	the current state
$\mathbf{s'}$, s'	the next state (after the action a)
$\mathbf{a}$, a	the selected action in state $\mathbf{s}$
$\mathbf{r}$, r_t, r	the reward received when in state $\mathbf{s}$ perform the action $\mathbf{a}$
t	the current timestep
ϵ	controls the exploration rate in the ϵ-greedy algorithm
$MOS_{i,t}$	the MOS perceived by the station i at the timestep t
M	the number of the connected wireless stations with web flow
Q^*	the optimal Q-value
$Q(s,a)$	the estimated Q-value for state s and action a
α	the learning rate
γ	the discount rate
AP_{id}	the id-th AP
c_i	the channel change in Q-learning: increase 1 channel, decrease 1 channel or remain the same
p_i	the transmission power change in Q-learning: increase 1 dBm, decrease 1 dBm or remain the same

5.1 Contributions

The main advantages of the proposed method in relation to the literature is that the RL method used does not need to know the system model. Further, the learning method adapts to network changes, and seeks to continuously improve user satisfaction. The main contributions of this chapter are:

- We propose a power control, and channel selection scheme using a model-free approach to maximize the Web QoE. The proposed architecture is shown in Figure 4.2 using SDN-based control loops, and it is tested under three real scenarios (Section 5.5).
- We investigate the trade-offs of a distributed versus centralized decision approach in Section 5.5.3.2, and Section 5.5.3.3.
- We propose in Section 5.2 a web site classifier based on the similarity to three labeled sites in Hora et al. [2016], employing a semi-supervised learning approach.

The results obtained using the UCB1 approach shown in this chapter were published in Moura et al. [2019c]. The results comparing UCB1 and Q-learning were submitted to a journal (Moura et al. [2020]) and approved for publication.

5.2 Site classification

This section details our site classification model. In summary, it identifies, for each site, the similarity to the three examples provided by Hora et al. [2016]. This measure of similarity is important because it defines which of the three MOS predictors will be used in the control loop.

Hora et al. [2016] rank sites into three categories - *light*, *average*, and *heavy*, indicating the site complexity. Then, they built a MOS predictor for each category. The *light* site represents a site with very lightweight pages such as the front page of search engines (e.g. Google). On the other hand, a *heavy* site represents sites with pages with many objects/images, e.g. Amazon. An *average* site represents a site with pages of intermediary complexity, like the Facebook's front page.

Hence, it is critical for our system to classify any site into these classes based on similarities with the three examples provided by the authors, and infer the page load time without altering the client. Since there are very few labeled examples, the classifier must deal with a problem called low-density separation, placing boundaries in regions where there are few (labeled or unlabeled) data points (Chapelle and Zien

[2005]). This thesis employs TSVMSC to classify the sites, as explained in the following subsection. Section 5.3.2 compares TSVMSC against other methods in the literature. Only TSVMSC was used in the experiments, because it performed best.

Our implementation uses a QoE metric from the literature. Hora et al. [2016] only presented in their paper, one example for each class, and did not provide a full qualification of what defines each classes. One way to get more examples would be to use a multi-person panel that would provide information on how different sites on the Internet would look similar in terms of perceived QoE for the three examples provided by the authors. This was done on a small scale in this thesis, as shown in Section 5.3. However, this showed that there is a high cost in terms of time as the respondent has to compare each desired site to the three known ones. Using recorded examples for comparison limits the variation perceived by users, as the quality of web site access depends on the network performance. Notice also that the examples are noisy, thus machine learning methods are a good way to deal with this classification problem, if proper care to bias and overfitting is taken.

Supervised learning methods did not work well (see Section 5.3.2) as they work best when there are many labeled examples. Because the features that would characterize the class are not defined, we use only four features, because using more features implies a collection cost from the network interfaces of the APs. Unsuppervised methods work best when there are many features, which in our case impose a high cost of gathering information at runtime. We use a semi-supervised method, because the obtained result is better than the other ones, as the method improves the results by also considering the unlabeled examples. This method defines a frontier that separates the classes. Having this frontier defined in training, one might think that these thresholds can provide the classification in a more efficient way. However, this frontier is dynamic, because the mapping of a site to a class can vary due to the network performance, and also due how the site dynamically compose their pages, *e.g.*, some high resolution image or video can be added to a clean page site turning it from a light at an average site. The robustness of our classifier should be addressed in a future work.

5.2.1 Transductive Support Vector Machine with Spectral Clustering

Classic ML approaches such as supervised learning are not adequate to our problem due to the need of a large number of previously classified examples, and prior training. Also we do not know if the features naturally split into two sets, purging generative methods from our options. Thus, TSVMSC became a good option because it is a semi-

supervised learning classifier that works with a small amount of pre-classified data (Yu et al. [2012]). TSVMSC is a technique with two phases (Yu et al. [2012]).

The spectral clustering (SC) phase (Ng et al. [2002]) generates K clusters among the unlabeled data. It uses eigenvalues of the data similarity matrix to perform dimensionality reduction before clustering. The similarity matrix is built using the normalized Euclidean distance between every unlabeled data. The number of clusters generated by SC is selected using a technique proposed in §3 in Zelnik-Manor and Perona [2005], which analyzes the eigenvectors of the similarity matrix. The Laplacian is rotated so there is more than one non-zero entry in some of the rows. The authors define a cost function based on the Laplacian values, and minimizing this cost over all possible rotations will provide the best alignment with the canonical coordinate system. The number of clusters is the one that provides the minimal cost. Notice that this number can be greater than three (the number of site classes).

The classifier then obtains three centroids c_n, considering the Euclidean distance amongst the labeled data. Next, the method calculates the centroid c_u for the clusters of unlabeled data. The final phase of SC maps unlabeled clusters into one of the three labels (*light*, *average* or *heavy*). To do so, the algoritghm uses the Euclidean distance between c_u, and centroid c_n. This provides the first approximation for the unlabeled data. We refer the reader to Zelnik-Manor and Perona [2005], and Ng et al. [2002] on the details of this algorithm.

In the second phase, the classifier is created using TSVM. This phase uses the clusters discovered by SC as a hint to the number of positive examples in each class. This algorithm receives as inputs the labeled examples $(\mathbf{x}_i, y_i)$, the unlabeled examples $\mathbf{x}_j^*$, the number of positive examples, C and C^* parameters set by the user to penalize the margin around the labeled and unlabeled examples, respectively, and solves Equation 5.1 (non-separable case).

$$\begin{aligned} & \min_{(y_1^*,...,y_q^*,\mathbf{w},b,\xi_1,...,\xi_p,\xi_1^*,...,\xi_q^*)} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{p}\xi_i + C^*\sum_{j=1}^{q}\xi_j^* \\ \textit{subject to} \quad & \forall i \in \{1,p\} : y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i \\ & \forall j \in \{1,q\} : y_i^*(\mathbf{w}\cdot\mathbf{x}_i^* + b) \geq 1 - \xi_i^* \\ & \forall i \in \{1,p\} : \xi_i > 0 \\ & \forall i \in \{1,q\} : \xi_i^* > 0 \end{aligned} \tag{5.1}$$

TSVM finds a tuple $\left(y_1^*, ..., y_q^*, \mathbf{w}, b, \xi_1, ..., \xi_p, \xi_1^*, ..., \xi_q^*\right)$ that minimizes Equation

5.1, where p is the number of labeled examples, q is the number of unlabeled examples, y_i^* are the estimated label to the unlabeled examples, $\mathbf{w} \in \mathbb{R}^m$ and $b \in \mathbb{R}$ are the estimator parameters (they define the hyper-plane that separates the classes), and ξ_i and ξ_i^* are slack relaxation variables for the TSVM. We applied the $QN - S^3VM$ heuristic proposed in Gieseke et al. [2012] to obtain the solution to the TSVM problem.

5.3 Site Classifier Evaluation

This section presents the evaluation of the proposed site classification algorithm. It shows the effect of altering the hyperparameter values on the accuracy obtained in the test set. After that, it describes the classification results when compared to different methods.

In order to test the performance of the proposed site classification model, we collected 2,880 probes for the three site examples that define the site types. The development and test data acoounts for the web's ten most visited sites[1], based on the rank in December 28, 2016. We excluded from this list the three reference sites, Google's local search sites, porn sites and sites that do not respond to *ping* requests[2]. A total of 40,000 probes were collected for the top ten sites. These new sites were classified by five people, who were asked to give a rating of 1 (less likely) to 10 (equal) on how much the site download is similar to the three known sites. The classifier uses three features: page load time, page size, and RTT. The scores are shown in Table 5.2, with the best score in bold. The class with the highest average score is selected to represent the site.

5.3.1 Site Classification hyperparameter search

We tested the classifier using $\{500, 1000, 1500, 2000\}$ unlabeled examples. We also considered four values of $C^* = \{0.1, 1, 10, 100\}$, and four values of $C = \{1 \times 10^{-07}, 1 \times 10^{-05}, 0.001, 0.1\}$. Training was performed with only 15 labeled examples (5 of each site type). Twenty repetitions were performed for each combination of parameters. A linear, and a Gaussian kernel functions were used with TSVM, but we show only the linear kernel values because it was faster, and produced better results. The results shown in this section are collected in a regular PC (see the controller specification

[1] https://en.wikipedia.org/wiki/List_of_most_popular_websites

[2] This approach can be changed to obtain the RTT using the TCP connection to the web server, using data obtained in the three-way handshake.

Table 5.2: Classification of new sites

	Site	Scores (from 1 to 10)		
		Light	**Average**	**Heavy**
1	www.youtube.com	1,20	**8,00**	6,20
2	www.baidu.com	**9,60**	1,00	1,00
3	www.wikipedia.org	**8,00**	1,80	1,00
4	www.reddit.com	1,00	4,00	**8,20**
5	www.yahoo.com	1,00	6,20	**8,00**
6	www.twitter.com	**7,40**	2,20	1,00
7	www.sohu.com	2,60	6,40	**8,20**
8	www.instagram.com	**6,80**	1,20	1,40
9	www.vk.com	1,20	2,00	1,20
10	www.live.com	1,20	5,40	**6,80**

in Section 5.5.1). The algorithm is implemented in Python version 2.7, as all other controller modules.

Table 5.3: Average score and runtime with test set size = 500 and $C = 1e-05$

C*	**Average Score**	**Confidence Interval (95%)**	**Average Runtime (s)**	**Confidence Interval (95%)**
0.1	0.872000	0.016752	0.621188	0.044558
1.0	0.872700	0.016696	0.679447	0.045388
10.0	0.876067	0.017418	0.735572	0.047512
100.0	0.878467	0.016391	0.798879	0.044099

Table 5.3 shows the average score in the test set, and average execution time obtained with 500 unlabeled examples, and $C = 1 \times 10^{-05}$. It also shows the 95% confidence interval (CI) of both values, using Student's t-test. The score values range from zero (worse fitness) to one (perfect fitness). The classifier is trained using 3-fold cross-validation. The difference of the scores obtained by varying C^* shown in the table is very small. For this reason, we only show $C^* = 0.1$ results in Figures 5.1, and 5.2.

Figure 5.1 shows the execution time[3] of the classifier with $C^* = 0.1$, considering the variations in the size of the test set and the parameter C. It takes about one second to train the classifier with 500 unlabeled examples. Since the TSVM algorithm is not incremental, to add a new example implies in running the algorithm again. We observe an exponential growth of the execution time as a function of the size of the test set, thus we prefer to perform the classifier's training off-line. Note that further improvements in the execution time can be obtained using runtime versions

[3]Time that takes to execute the training and test phases of the classifier.

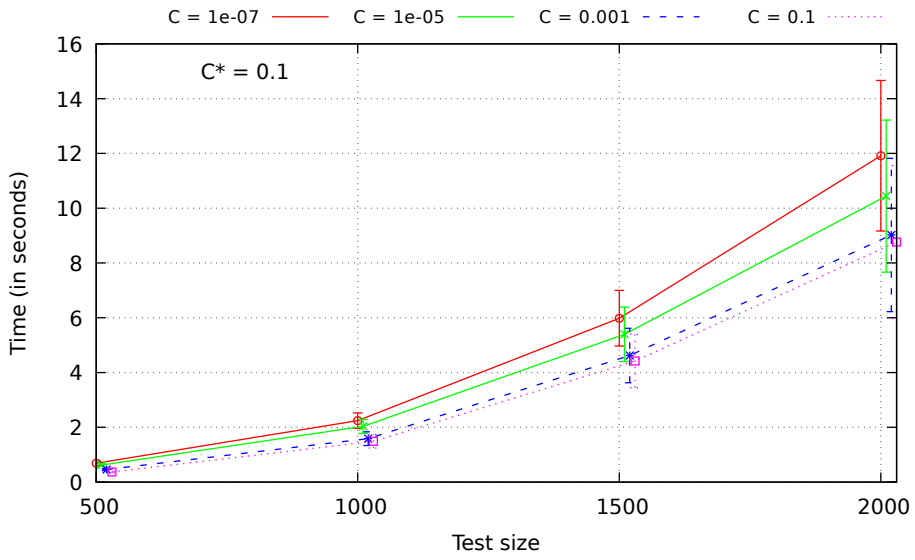

Figure 5.1: Execution time with $C^* = 0.1$

of our Python code (e.g. Cython or PyPy). The labels provided by SC can be used to obtain the first estimation of the TSVM's $\mathbf{w}$, and b values, and by that improve the performance. Further, the methods presented in Chapelle et al. [2008] might also improve the classifier's performance.

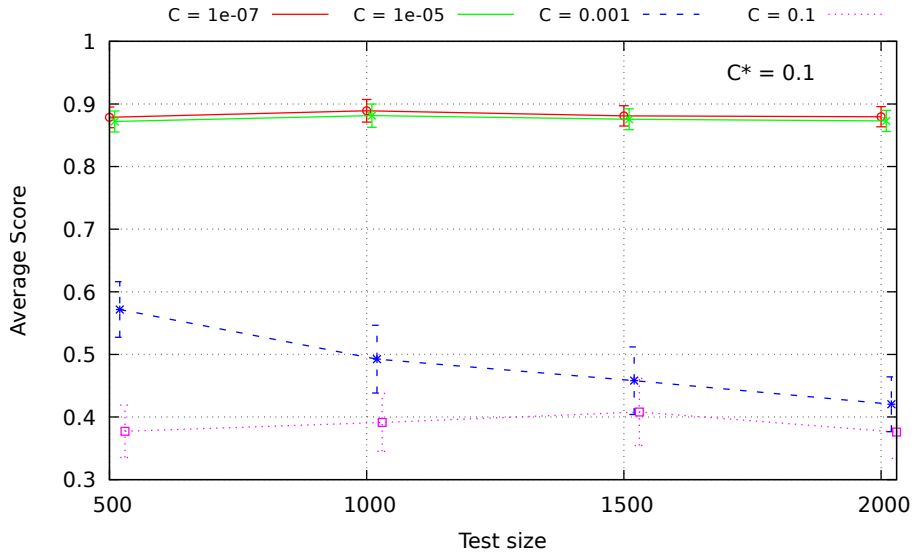

Figure 5.2: Score in the test set with $C^* = 0.1$

Figure 5.2 shows the average scores obtained for the test set data, with $C^* = 0.1$, and by varying the size of the set, and the C parameter of TSVM. The scores assume values between 0 to 1 (100%), but the figure only shows the interval where our values fell (notice that the Y-axis starts at 0.3). It also shows the 95% CI for each value. We

observe that the C parameter has a strong influence on the score obtained, however, the two smaller values $C = 1 \times 10^{-05}$ and 1×10^{-07} obtain practically the same result, because their 95% CIs overlap. The experiments evaluating the control loop use $C = 1 \times 10^{-05}$.

The classifier from time to time must be retrained to improve the quality of the classifier. Since good scores are achievable with 500 unlabeled examples, we chose to set the batch size to 500. Using this size, the model can be retrained more frequently.

The classifier generates an overhead in the network, however, our experiments provide a worst case scenario for the overhead, since each station only accesses one site during the entire experiment. In the controller initialization, it makes five accesses to each of the typical sites (Google, Facebook, and Amazon). The downloaded content average sizes are $36,019 \pm 1$ bytes for light, $47,626 \pm 5,884$ bytes for average and $975,562 \pm 56,208$ bytes for heavy, so at start up it downloads 5 MB[4]. Compared with our worst case in the centrally-controlled multi-agent (CA) experiment (Section 5.5.3.3), that represents a 350% overhead over the useful traffic, but it is less than 2% in the best case (in stand-alone (SA) experiment, Section 5.5.3.1).

5.3.2 Comparison of TSVMSC with other classification methods

We evaluated several classification methods using supervised, and unsupervised learning. Table 5.4 shows the best results for each of the evaluated methods, and the model's hyperparameters. The first column shows the name of the method. The second column identifies if the method is supervised, unsupervised or semi-supervised. The next column shows the best accuracy in the test set, and the fourth column shows the hyperparameters for the model. All classifiers were tested from the sklearn library[5], so the default values are not shown.

The evaluation of the classifiers shown in this section were made by dividing the data in a train/test dataset and a validation dataset. The first group was used to train the model and select the hyperparameters using 3-fold cross-validation. The validation

[4]The Google page is the simplest of them. It has some text, and CSS elements, and about five images (three icons — page, magnifying glass, and keyboard), the Google logo, and a small Google advertising image). The Facebook page is a bit more complex. It contains two midsize images (globe connections image, and company logo). However, it contains a few more HTML text elements since its form is more complex as it is the page footer. The Amazon page provides a large amount of images (static and animated), and eventually video. Notice the 56 kB confidence interval for its downloaded content size. The Amazon page is much bigger than the others. The two other pages are smaller, however the size difference between them is not negligible, since the test of the difference of means with 99% confidence indicates that the means are different.

[5]Version 0.21.0 — https://scikit-learn.org/stable/user_guide.html

dataset was used only to obtain the performance metrics shown here. Although a model can overfit the training data, this model might provide bad results in the validation set.

Table 5.4: Performance comparison of methods for site classification

Classifier	**Category**	**Accuracy**	**Hyperparameters**
k Nearest Neighbors	Supervised	39.75%	K=500 leaf size=5 uniform weights
k-Means	Unsupervised	39.96%	K=4 distances=auto
Gaussian Mixture models	Unsupervised	40.02%	components=4 spherical covariance
SVM	Supervised	44.8%	C=0.1 gamma=0.1 RBF kernel
Gaussian Process	Supervised	50.7%	RBF kernel length scale=16
Linear Regression with Gradient Descent	Supervised	55.3%	alpha=10^{-5} epsilon=0.1 $eta_0 = 0.0001$ L1 penalty learning rate="optimal"
TSVMSC	Semi-supervised	87%	linear kernel

The results obtained show that the semi-supervised method achieves better results because it uses additional information from the unlabeled examples. The supervised, and unsupervised methods showed in most cases overfitting in training, and thus underperformed the semi-supervised method. This is due to the small amount of features used to characterize each site, and also because we only know one site for each class. Thus we could not add enough variability to allow for extrapolations in the supervised case or to allow better definition of clusters in the unsupervised method. However, we believe that for a production system, investing in labeling more sites, and creating rules and clear characteristics of these classes would allow us to create a more flexible system. Thus, supervised methods may also provide adequate results.

5.4 RL-based control loop

This section presents the proposed control loop. It describes how the control loop obtains the reward used by the two learning algorithms tested in this chapter, and also how the MOS was obtained using interpolation. This section also describes the states, and the actions available to the controller.

5.4.1 Control loop using Q-Learning

One of the approaches that we selected for the control loop was Q-learning (explained in Section 2.5) with an ϵ–greedy exploration strategy. Q-learning uses a matrix, called Q-matrix, which stores the reward obtained for each of the possible states and actions. This section shows what are these states and actions, as well as how the Q-value is stored.

State in Q-learning: To estimate the value of a user's QoE, we use the estimator constructed by Hora et al. [2016]. This estimator considers two parameters: PHY medium busy (in percent), and the average transmission PHY rate (user bitrate in percent). Those two parameters are discretized according to Hora et al.'s heat map. The state space is thus represented by a three-dimensional matrix. The first dimension represents the n controlled APs – AP_{id}, and the other two dimensions represent the input values of the QoE predictor: the *PHY medium busy*, and the *station bitrate*. The authors provided 63 discrete values for PHY medium busy, and 37 for user bitrate. Thus, all features that compose the state have discrete values. Therefore, the Q-learning's state matrix size is $n \times 63 \times 23$.

$$s = (AP_{id}, PHY\ medium\ busy, Average\ station\ bitrate) \tag{5.2}$$

As will be presented in the evaluation section, the control will be evaluated under different numbers of controlled APs. This representation allows the controller to distinguish what is happening at each AP, so the algorithm can learn how to handle cases where different APs have different amounts of stations and are subject to different environmental conditions. Therefore, consider, for example, a scenario with 2 APs. The tuple $(1, 0.25, 0.75)$ would represent the first AP and its environmental condition, while the tuple $(2, 0.25, 0.75)$ would represent the state of the second AP. Notice that $PHY\ medium\ busy$, and $Average\ station\ bitrate$ are percentages of the maximum allowed value.

Actions in Q-learning: Our control loop can alter two WiFi parameters: the transmission power and the transmission channel. The transmission power is in the range of 1 dBm up to 15 dBm, in our devices, and there are eleven transmission channels, from 1 to 11 (allowed channels in Brazil in 2.4 GHz).

The actions for the channel change (c_i) are:

1. go to the next valid channel;

2. go to the previous valid channel; and

3. remain on the same channel;

when the first valid channel is reached, the action "go to the previous valid channel" does not change the channel. The same occurs when the last valid channel is reached and the action is "go to the next valid channel". The actions for power change (p_i) are:

1. to increase transmission power by 1 dBm;

2. decrease transmission power by 1 dBm; and

3. remain with the same transmission power.

When the power reaches the maximum of the wireless network card (or reaches the minimum value of 1 dBm), the action that increases (or decreases) the transmission power does nothing. When the channel reaches the maximum (or the minimum) supported by the wireless network card, the action to change the channel does nothing. These invalid moves receive a reward of -1.

Thus, there are nine possible actions, and they are a expressed as:

$$A_{Q-learning} = \{a_1, a_2, \ldots a_9\} \tag{5.3}$$

where $a_i = (c_i, p_i)$ is the proper combination of both parameters modification.

We decided to use to use two actions in our implementation. However, it is transparent to the implementation how many actions are there avalaible. The number of action affects the memory usage by the system, and the time needed to explore the state-action space that grows linearly with the number of actions, which may increase the convergence time.

5.4.2 Control loop using Multi-armed Bandit

The second learning algorithm used in this chapter is MAB. The algorithm uses the UCB1 index proposed by Auer et al. [2002], which has two components, one that accounts for the estimated average reward, and another one that accounts for the confidence that the calculated average (first component) represents the distribution's true mean. UCB1 is explained in Section 2.4, while in this section the focus is on how to represent the states, and actions in the algorithm.

State in Multi-armed bandit: The UCB1 implementation has no state, but the algorithm keeps track of the arms' probability distribution. Our implementation saves the action probability distribution as a matrix of size $l = n \times C$, where n is the number of APs, and C is defined by the number of the probability distribution parameters and the number of actions. In our implementation $C = 2 \times K$, because UCB1 (Equation 2.8) contains two parameters: $\hat{r_j}$, and n_j. K is the number of actions (or arms in each AP). Thus the algorithm's memory usage increases linearly with the number of controlled APs, and the number of actions.

Actions in Multi-armed bandit: The actions in UCB1 are mapped to the arms, so in our implementation there is an arm for each combination of (channel, transmit power) for each AP, thus the action set A_{UCB1} is a discrete and finite set. Thus for each AP, the actions are:

$$A_{UCB1} = \{a_1, a_2, \dots a_K\} \tag{5.4}$$

where $a_i = (c_i, p_i)$ is the proper combination of both parameters. $c_i = \{1 \dots 11\}$ channels, and $p_i = \{1 \dots 15\}$ dBm. Thus, in our setup, there is $l = 11$ channels $\times 15$ power levels. Notice that in UCB1 there are no invalid moves.

5.4.3 Reward

The reward is defined as whether the new system configuration increases or decreases the QoE. At time t, the MOS perceived by a wireless station i browsing a site of class k is defined as $MOS_{i,t}^{(k)}$. The system's reward is defined as the average MOS of all the stations shown in Equation 5.5.

$$r_t = \frac{1}{n} \sum_{i=1}^{n} MOS_{i,t}^{(k)} \tag{5.5}$$

The station's QoE ($MOS_{i,t}^{(k)}$) is approximated using Hora et al.'s predictor for each class k. Hora et al. [2016] signed a non-disclosure agreement (NDA), thus they only provided us the predictors' heat map values. These values are represented by three 63×23 bi-dimensional matrices, one for each site type. The MOS obtained by their predictors represents the perceived QoE at one station in state s. Notice that we use the site classifier described in Section 5.2 to determine which predictor should be selected for each station.

Since the MOS values provided by Hora et al. [2016] are discrete, and the values read from the APs are continuous, the $MOS_{i,t}^{(k)}$ values are calculated as a linear regression of neighbor points of the read value as shown in Figure 5.3. Three cases are identified as follows, where $(Phy, Busy)$ is the value read from the AP:

(1) If the point is provided by Hora et al. [2016], i.e, it is in the heat map, $(Phy, Busy) = (Phy_i, Busy_i)$, then $MOS_{i,t}(Phy, Busy) = MOS(Phy_i, Busy_i)$;

(2) If $(Phy, Busy)$ is between four points, as shown in Figure 5.3, the four points define a 3D surface, and the estimated MOS value is the interpolation of those points.

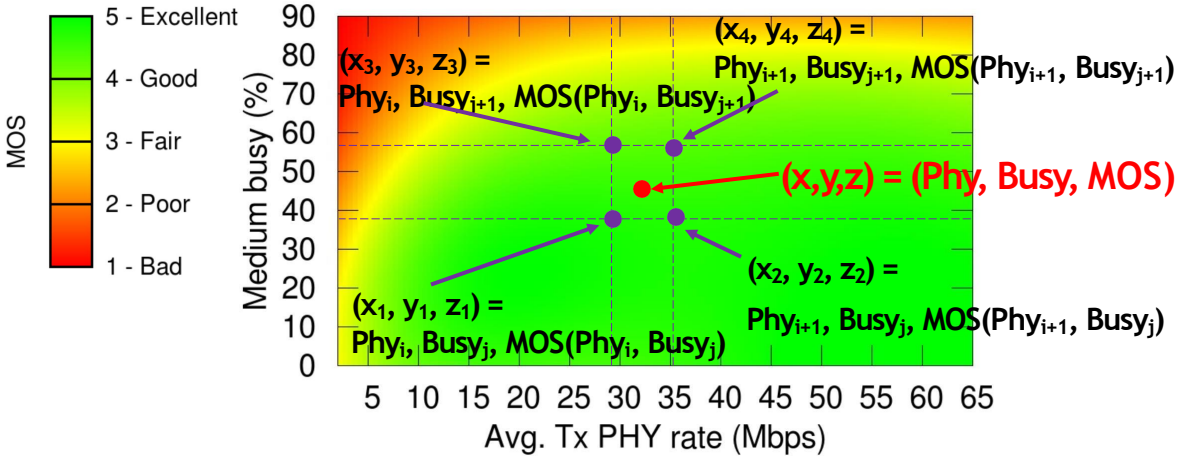

Figure 5.3: How MOS is calculated in our Experiment

(3) If $(Phy, Busy)$ is outside the figure, then the MOS value is truncated to the value at the nearest border, i.e, if $Phy > 65$ or $Busy > 90$, then $MOS_{i,t}(Phy, Busy) = MOS(\min(Phy, 65), \min(Busy, 90))$

5.5 Control Loop Evaluation

This section presents the evaluation of the control loop. Section 5.5.1 details the test setup. The baselines are also described. Section 5.5.2 evaluates the performance for different hyperparameters of the control loops. The performance evaluation of the prototype is shown in Section 5.5.3.

5.5.1 Experiment setup

The closed control prototype contains up to two *Ethanol* APs, and two wireless stations. The user stations do not need to be changed, relying only on IEEE 802.11 standard support. As the AP may change the channel several times, we used stations compatile with channel switch announcement message (CSA). CSA is used by an AP in a BSS to advertise the new channel before changing channels. This way the station recognizes in advance the channel change, and can accelerate the migration process to the new channel, reducing the disconnection time.

In each experiment, a station downloads a web page from a web site, requesting a new one when the previous ends (with no caching). Figure 5.4 and Figure 5.5 show the physical layout of our devices. The controllers are connected to a gigabit Ethernet network, as the stations, and the APs. The controllers and the stations use the Ethernet to send, and receive control data. The APs use the Ethernet to receive control data, but also to forward the station traffic to the Internet. The devices are aligned, and the distances between them are shown in Figure 5.4. The experiments are performed in an environment with more than 60 other APs, as shown in Figure 5.6.

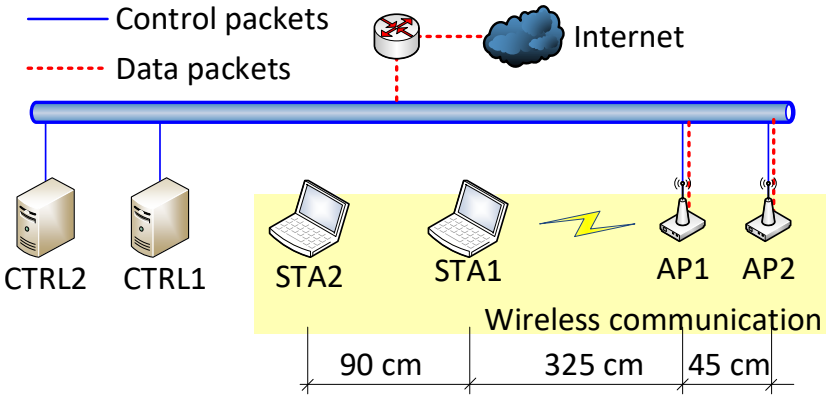

Figure 5.4: Experiments Layout

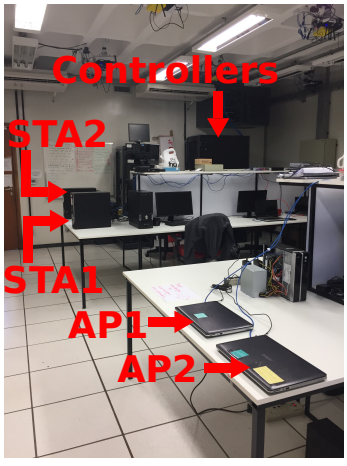

Figure 5.5: Equipment

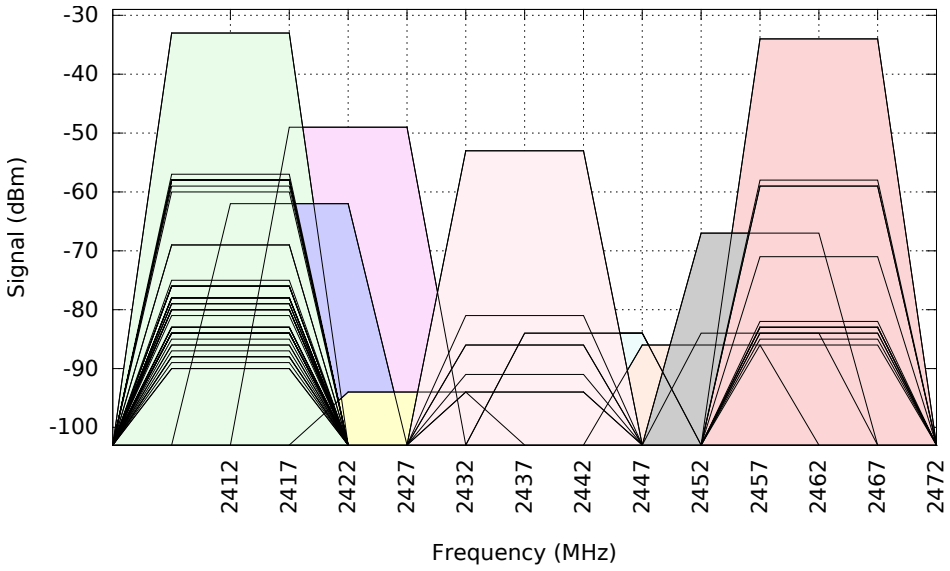

Figure 5.6: Wireless neighborhood of our experiment

The controllers are dual core Intel(R) i5 PCs with 16GB of RAM. The stations are PCs with dual core Pentium(tm) 2.4GHz CPU with 2GB of RAM and RT3062 802.11n wireless card. Because *Ethanol* runs on Linux, the APs are ASUS notebooks with Intel(R) i7 CPU @ 1.80GHz with 8GB of RAM and Atheros AR9485 wireless network adapters. All computers run Ubuntu Linux version 14.04 LTS.

Because Hora et al. rank sites into three categories, and since there are two clients, the experiments are executed considering six possible combinations of access by the clients: *light–light*, *light–average*, *light–heavy*, *average–average*, *average–heavy*, *heavy–heavy*. Each experiment is repeated 30 times for each traffic combination. Each experiment runs for 30 minutes, when it is forced to stop.

The experiments begin with the system having zero knowledge, so:

- In Q-learning, all values in $Q(s, a)$ are set to zero (Line 2 of Algorithm 2). We set $\alpha = 0.5$, so there is an equilibrium between what is already learned and the new reward. All experiments with Q-learning use $\epsilon = 0.05$, so 5% of the algorithm's choices are random in order to explore new configurations. The discount parameter γ is set to 0.8, so the Q-value is not only affected by the immediate reward r_t but also by the maximum predicted reward $Q(s', a')$. We will discuss more about these parameters later in this section.

- In UCB1, all arms start with an initial average value ($\hat{r_j}$) equal to one, *i.e.*, the actions are equiprobable, and are set to the minimum MOS possible.

Each step of the algorithm takes one second. This period can be adjusted by

the user. Therefore, state and reward information are obtained with this granularity. There are two factors that need to be considered when defining the step interval: (1) the total control loop execution time, which consists of communication, and processing time in the controller and devices; and (2) if the controller makes a channel change decision, it is necessary to wait for the CSA time. Thus we have adopted a conservative period of 1 s so that these activities can be performed successfully.

5.5.1.1 Baselines

We used two baselines in all scenarios. The first baseline corresponds to the case where the user has a traditional AP, such as the default *hostapd* implementation. The second baseline corresponds to an environment where the APs can be controlled centrally, and the controller can perform an online configuration procedure. We describe each of these baselines in more detail below.

***Fixed Baseline* — uses a regular AP:** The APs are started on a randomly selected channel, and with maximum transmission power.

ACS baseline — uses SDN to perform the ACS optimization: In the second baseline, the controller selects the best channel based on the automatic channel selection (ACS)[6] function implemented in *hostapd*. This algorithm is implemented in all Linux-based AP, thus it can be considered the state of practice.

This algorithm was implemented in the controller. The controller receives information from the environment of each of the APs it controls, which is fed into the ACS formula to identify the best channel. The algorithm queries the occupation time of each valid channel for each AP. The interference factor (IF_c) is calculated for each channel c, and it represents the rate of occupancy observed in the channel. It is calculated using Equation 5.6, shown below.

$$IF_c = 10^{\frac{chan\_nf}{5}} + \frac{busy\_time - tx\_time}{active\_time - tx\_time} \times 2^{10^{\frac{chan\_nf}{10}} + 10^{\frac{band\_min\_nf}{10}}} \tag{5.6}$$

Based on the interference factor, the channels are ordered from the one with the least interference factor (best option) to the one with the highest factor. The controller performs a greedy selection of the best channel. However, in the CA experiment (see Section 5.5.3.3), since the controller knows the best channel for both APs, the greedy selection contains two rules:

[6] https://wireless.wiki.kernel.org/en/users/documentation/acs

- if the best channel for both APs are different, the controller selects the best channel given by the lower interference factor for each AP;

- if the first ranked channel is equal for both APs, the controller checks if the second best channel has a idle time greater than half the time of the first channel. If so, the controller selects the second channel for the AP with the bigger idle time, otherwise it selects the first ranked channel for both APs[7].

We used *Ethanol* to request the controlled AP, every 5 seconds, to measure the parameters needed to calculate this heuristic, thus the controller performs the calculation, and, if there is a better (*i.e.* less busy) channel than the current one, it requests the AP to switch to this channel. The period to run the ACS procedure was manually selected, trying to beat the RL control loop performance. This implementation depends heavily on the amount of time spent gathering the survey data. It should be noticed that this is a costly procedure, because when the AP is scanning the channel, it cannot forward traffic. For that reason, in the default *hostapd* implementation, this procedure is only executed during *hostapd* startup. ACS can be suboptimal because short traffic bursts may be missed, making the algorithm pick a suboptimal channel. It also has to be performed several times, in order to obtain statistically meaningful data. Thus there is a tradeoff between channel interference measurement, the quality of the information obtained, and the AP performance in forwarding station traffic. In our implementation, the controller requests five samples, one per second. Some commercial solutions use a secondary wireless interface card to perform the scanning. This feature, however, is present only in high-end devices, and it was not implemented in our testbed.

5.5.2 Evaluating Q-Learning hyperparameters

There are two hyperparameters in Q-learning (Equation 2.11): ϵ and γ. ϵ controls the degree of exploration of the algorithm, and γ determines how the algorithm updates the value of $Q(s, a)$, *i.e.*, whether the previous value or the current reward is prioritized. Since the relation between the value function $Q(s, a)$, ϵ and γ is linear, we perform an analysis only for the worst combination of URL types: *heavy–heavy*.

To test the effect of ϵ and γ variation, we built a scenario with one AP and two stations. The parameters were tested using $\epsilon = \{0.01, 0.05, 0.10, 0.20, 0.40, 0.80\}$, and $\gamma = \{0.1, 0.2, 0.4, 0.6, 0.8, 1\}$. We did not test ϵ values above 0.8, because in this case

[7]This rule works well for our setup with 2 APs. Channel allocation is a hard problem, because the k-coloring problem can be reduced to it.

the control strategy is very close to a random strategy. The results are shown in Figure 5.7. Each line represents the variation of the number of steps necessary to reach the maximum MOS in experiments with different γ for a fixed value of ϵ (shown in the legend). For each point we show the average number of steps with a confidence interval of 95%.

Due to the results shown in the Figure 5.7, we cannot identify a pattern in the behavior of the algorithm related to the change of the parameters, because the confidence intervals overlap. However, we can observe that the confidence intervals are smaller when γ is close to 0.4, presenting greater variability (larger intervals) in the extremes, when $\gamma \to 0$ and $\gamma \to 1$. Thus, we assume that, for this scenario, there must be a balance between future, and immediate rewards.

5.5.3 Performance evaluation of the control loop

This section shows the results obtained in three scenarios using both RL methods. Each subsection describes the scenario, and shows the results comparing with the baselines described in Section 5.5.1.1.

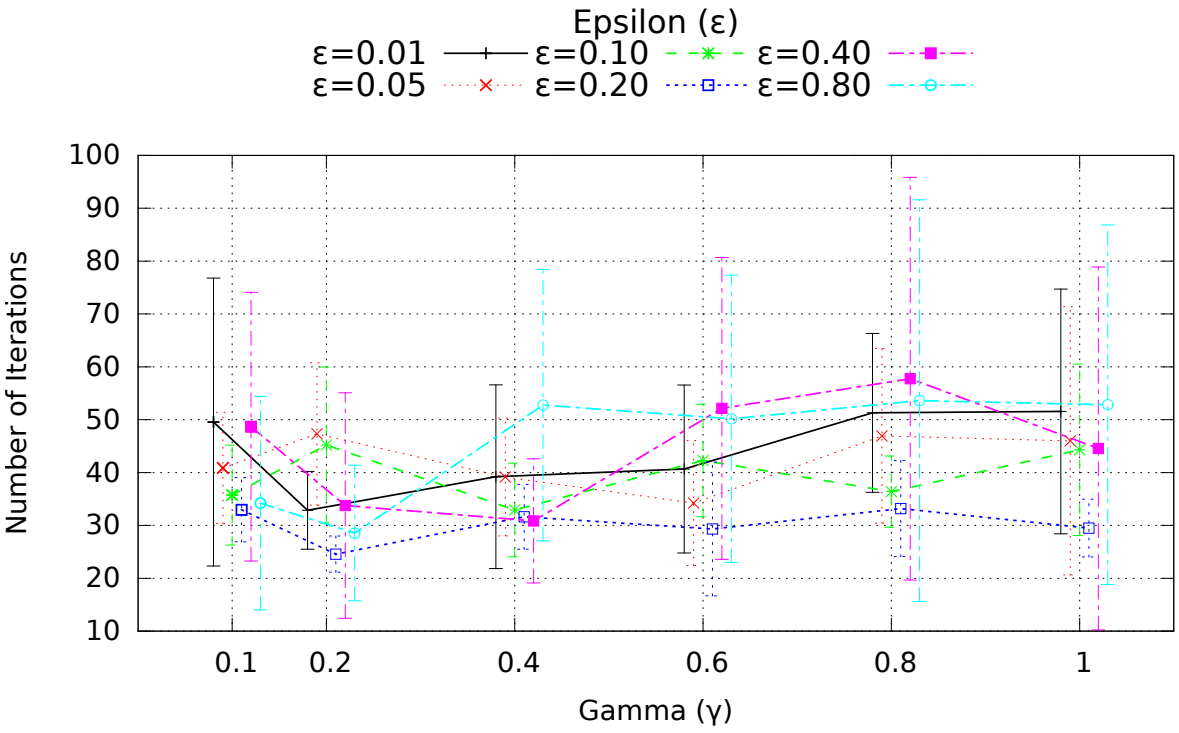

Figure 5.7: Relation between ϵ and γ from the Q-learning equation when both stations have heavy traffic. We show the values of the curves with small displacements around the point on the γ value in the X axis so that the confidence intervals are visible.

5.5.3.1 Stand-alone experiment

This experiment uses one controller, and it manages one AP and two clients, as shown in Figure 5.8. This scenario simulates a place with a single AP, like a home network. The control is centralized, so this is a RL scenario with a single agent. In this way the algorithm is executed on a state space that contains both stations, and therefore it aims to maximize the MOS of both stations.

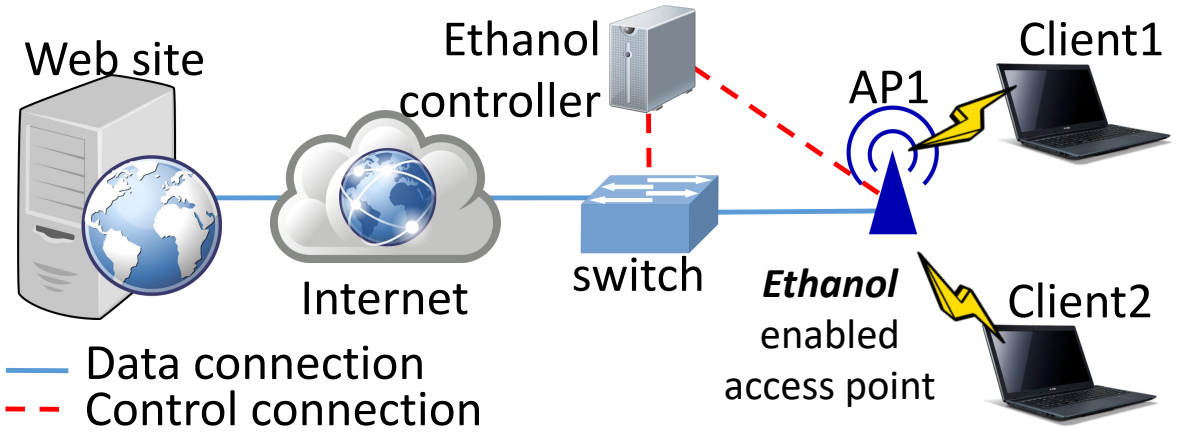

Figure 5.8: Web QoE – SA experiment - Setup

Table 5.5: Web QoE – SA experiment – Number of steps to reach maximum MOS using the controller

Client 1	Client 2	Q-Learning		UCB1	
		Number of Steps	Median	Number of Steps	Median
light	light	**22.3** ± 7.46	11.0	32.8 ± 56.46	**2.0**
light	average	**16.4** ± 3.10	14.5	166.6 ± 126.84	**2.0**
light	heavy	**36.4** ± 31.76	16.0	195.7 ± 177.47	**2.0**
average	average	15.1 ± 2.60	11.5	**6.9** ± 8.08	**1.0**
average	heavy	**32.0** ± 21.60	16.5	243.5 ± 279.73	**3.0**
heavy	heavy	**54.8** ± 26.45	**38.5**	100.7 ± 44.15	69.0

MOS evaluation Table 5.5 shows the number of steps that took for both wireless stations to simultaneously reach the maximum MOS = 5. The table rows show the values for the site combinations, followed by the average number of steps with a 95% confidence interval, and the median. As each step occurs in approximately one second, these values also represent an approximation of the time spent to reach this maximum. We observe that the cases when a *light* site was involved are the ones with lower convergence times for the Q-learning method. Since the download requires few network resources, the controller can quickly reach the maximum MOS. It is hard to say the the same for UCB1, because the variance is larger in most of the cases. However, UCB1

shows a lower median in most of the cases, but higher average number of steps until convergence. Notice that the confidence interval of the average value is large, *i.e.*, there are some runs that took a longer time for UCB1 to reach the best result. On the other hand the cases where the *heavy* site is present are those that show the longest average convergence time, as well as the larger confidence intervals, because *heavy* sites need more network resources, so the algorithm has a hard time to reach the maximum.

We highlight the variation of the confidence interval in the cases where one station downloads from the *light* site, and the other from the *heavy* site using Q-learning. In this case, in one of the repetitions, the algorithm took 481 steps to converge. All other runs are between 7 and 61 steps. Running this experiment without the control algorithm, we observed that all the runs reached the experiment time limit – 30 minutes, but none of the runs reached the maximum MOS value.

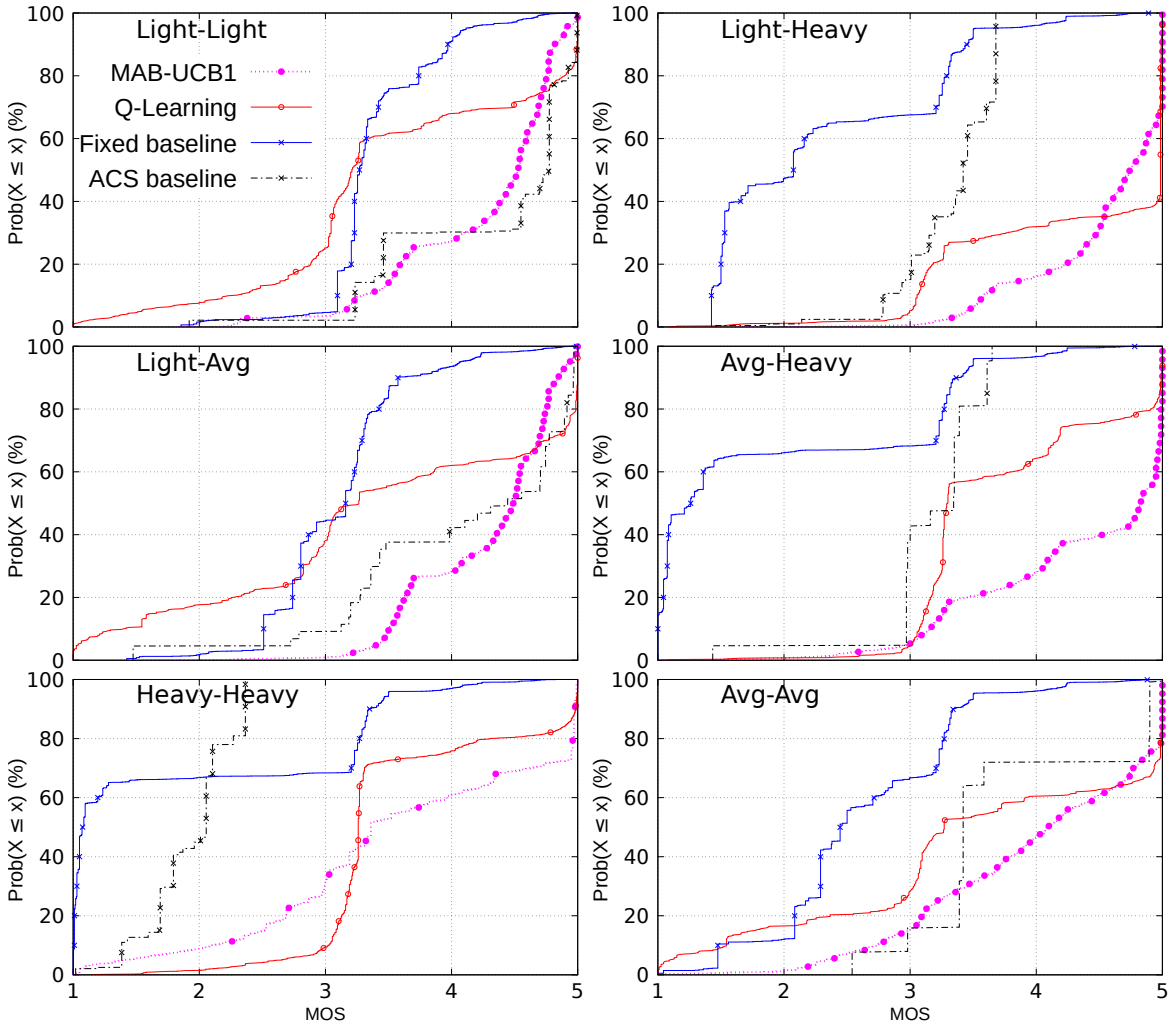

Figure 5.9: SA experiment - Cumulative distribution of MOS

Figure 5.9 shows the cumulative distribution function (CDF) of MOS values during the experiment using both control loops as well as the baselines. The curves show, in the X-axis, the value of the MOS, which varies between 1 and 5. The Y-axis shows the cumulative probability. For the cases where both stations access *light* sites using Q-learning, only the last 40% of the measurements have values greater than the *Fixed baseline*. This behavior occurs because the APs in *Fixed baseline* work at maximum power, favoring the connections of the stations in a CSMA/CA environment, improving the bitrate. However, both baselines are unable to reach the maximum MOS during the experiment. On the other hand, in cases where one of the stations accesses a *heavy* site, the curve obtained for Q-learning is better than *Fixed baseline*. UCB1 outperformed *Fixed baseline* in all cases, and in most of the cases it also outperformed Q-learning.

Table 5.6: Web QoE – SA experiment – Average Page Load Time using Q-Learning and UCB1 compared to *Fixed baseline*

Client 1	**Client 2**	**Fixed**	**QL**		**UCB1**		**UCB1 / QL**
		Page Load (s)	**Page Load (s)**	**Gain (%)**	**Page Load (s)**	**Gain (%)**	**(%)**
light	light	5.8 ± 0.6	3.0± 0.11	90 %	2.8 ± 0.1	108 %	9 %
light	average	13.0 ± 0.1	4.1± 0.61	217 %	3.8 ± 0.4	240 %	7 %
light	heavy	9.6 ± 0.1	5.4± 1.37	77 %	4.4 ± 0.4	118 %	23 %
average	average	20.6 ± 0.2	6.4± 0.97	223 %	7.2 ± 0.5	186 %	-11 %
average	heavy	16.5 ± 0.1	11.9± 2.28	39 %	8.5 ± 0.8	94 %	40 %
heavy	heavy	12.6 ± 0.2	8.0± 0.92	56 %	7.3 ± 2.2	73 %	11 %

Page load time measurements We also evaluated the impact of the control algorithm on the average page load time. The page load consists of the time taken to download the whole page, *i.e.*, the HTML file, all script files, RSS files, and images files. Table 5.6 shows the average page load time, in seconds, considering both stations. All values are given with the overall average of the experiment and the confidence interval range with 95% confidence.

The perceived variation occurs because the servers return a slightly different set of images, and scripts for each request, e.g., different publicity for each request. In all cases shown in the table, the results using the controller are better than *Fixed baseline*. The worst result shows a 39% improvement obtained using Q-learning for the *average–heavy* combination. The right-most column (called " UCB1 / QL ") shows how much the UCB1 algorithm outperforms Q-learning. Q-learning performed better than UCB1 only in the *average–average* case.

UCB1 also provided a more stable result than Q-learning, as shown by the narrower confidence intervals in Table 5.6. Notice that only in the *heavy–heavy* case, Q-learning showed a narrower confidence interval, and in this case both algorithms

provided statistically the same average page load time. This is due to changes in the medium busy, and average transmission PHY rates by exterior factors, e.g., increase of environmental interference, noise, other BSSID transmissions in interfering channels, etc. Those exterior factors change the reward distribution, and the algorithms must readjust.

Overhead of the control loop During the experiment, the controller, and the AP exchange, on average, 193 ± 92 control messages. This generates an overhead of 509.2 *kilobytes*[8]. It is important to note that these messages are not transmitted in the wireless medium. For the experiment with lower traffic, which corresponds to access to sites with both light types, the total overhead was 7.7 MB, while for the situation of higher traffic this number raises to 280 MB. Both Q-learning and UCB1 use the same set of messages, and the messages are also sent with the same pattern.

Analyzing the regret One of the ways to analyze the behavior of an agent is to compare its performance with that of an optimal strategy. This strategy consistently selects the best action in the first n steps, for any horizon of n steps, which is defined as the regret as shown in Equation 2.7. In other words, the regret is a measurement of the agent's loss by not always playing optimally (Kuleshov and Precup [2014]). Although we do not know the optimal strategy, we know that the maximum reward is equal to five, so the results can be compared to a strategy that would always reach the maximum reward.

Table 5.7 shows the average regret with a 95% confidence interval. We observe that most of the time the regret using UCB1 is lower than the others, and its confidence interval is narrower. The confidence interval overlaps for the *heavy–heavy* case, thus the results for the two learning methods are statistically equivalent.

Table 5.7: Web QoE – SA experiment – Comparing Regret

Regret by Combination of Type	**UCB1**	**QL**	***Fixed***	***ACS***
light-light	0.743 ± 0.004	1.486 ± 0.072	1.611 ± 0.005	**0.671** ± 0.015
light-average	**0.711** ± 0.003	1.568 ± 0.104	1.904 ± 0.006	0.936 ± 0.052
light-heavy	**0.440** ± 0.003	0.642 ± 0.053	2.706 ± 0.011	1.724 ± 0.005
average-average	**1.013** ± 0.005	1.468 ± 0.108	2.386 ± 0.009	1.263 ± 0.081
average-heavy	**0.634** ± 0.005	1.246 ± 0.049	3.108 ± 0.013	1.838 ± 0.037
heavy-heavy	**1.411** ± 0.014	1.454 ± 0.038	3.164 ± 0.014	3.066 ± 0.008

[8]Those values are obtained via *tcpdump*, including TCP's three-way handshake.

Performance during convergence To illustrate how the initialization affects the rewards, Figure 5.10 shows the MOS obtained in ten executions of the experiment when the clients access *light*-type sites. Three curves are shown. The first one corresponds to the cumulative distribution of the rewards during the first 200 seconds of 10 runs. The others correspond respectively to the 200, and 400 seconds following this initialization phase. The results after the initialization are better, since these curves are more to the right in the graph than the curves obtained with the first 200-second results. Further, they also have a shorter initial tail than the curve in blue.

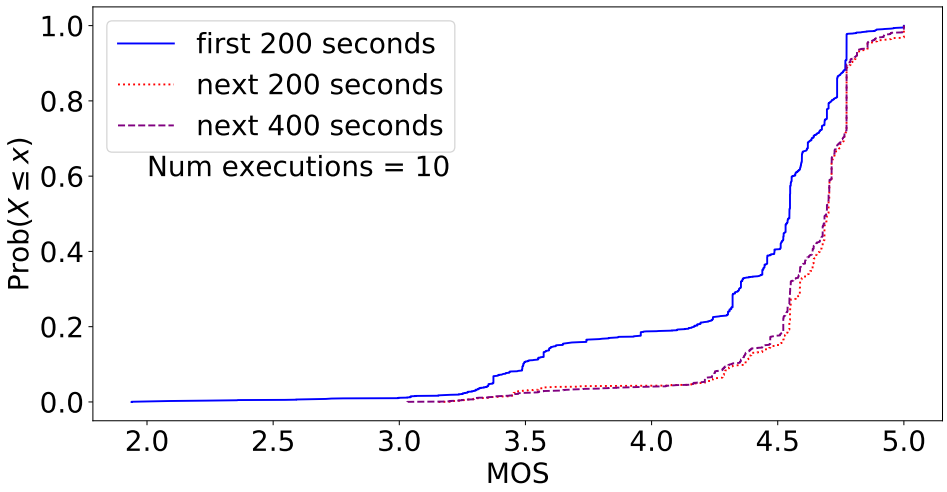

Figure 5.10: Comparison of MOS during initialization for (*light–light*) experiment

The average MOS, and its 95% confidence interval are, respectively, 4.36±0.02 for the 200 second intervals, 4.61±0.01 for the next 200 seconds, and 4.60±0.01 for the next 400 seconds. This means that the first interval is different from the last two intervals (using the Student's t test). Comparing the averages, one can notice a gain of 5.8%, and 5.5% from the intervals of the next 200, and 400 seconds respectively in relation to the initialization phase (first interval). Another way to highlight the difference of the results is to perform the Kolmogorov-Smirnov (KS) test for two samples (Engmann and Cousineau [2011]). Comparing the initialization phase with the subsequent 200 seconds, there is a 40% difference in the area. This ratio indicates that the agent achieves a cumulative performance in the subsequent 200 seconds of almost one-and-a-half times better than the initialization phase. The initialization compared to the next 400 seconds provides a similar difference of the order of 34.4%. The KS test for a 95% confidence indicates that the curves are statistically different.

5.5.3.2 Multi-agent experiment

This scenario simulates a deployment with multiple APs, such as a shopping center, a condominium, etc., and each AP is managed by a different administrator who does not exchange information with the others. We will show that it is possible to obtain benefits to the stations connected to the various networks. This is a classical case of multi-agent RL where the agents do not communicate, but have to cope with the cross-interference generated by neighbor networks.

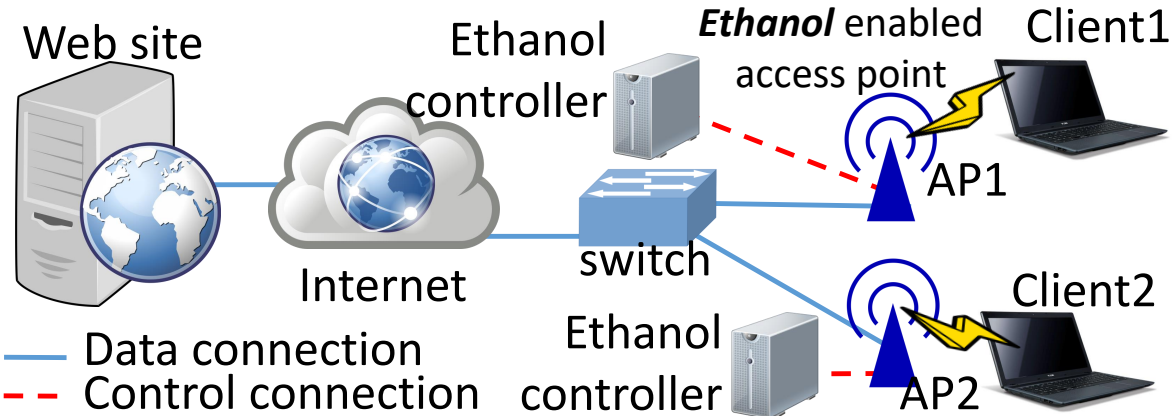

Figure 5.11: Web QoE – MA experiment - Setup

This experiment uses two independent controllers. Each controller manages one AP, as shown in Figure 5.11. The APs are near each other (less than one meter apart), so are the stations. Please refer to Figure 5.4 to see the physical layout of the devices. The APs are started on the same channel, and with the same power. This experiment is similar to the SA experiment because the controllers can only obtain information by reading the environment, but here the effects of adding intelligence to the system can be evaluated, so that the cross interference that is generated by the stations' transmissions to their respective APs can be reduced by the learning algorithm.

The RL theory states that there is no guarantee of convergence for the situation where agents are controlled separately, but there is empirical evidence that convergence can occur in practice (Tan [1993]; Seah et al. [2007]; Shirazi et al. [2009]). In this thesis we will only show that convergence was obtained for the evaluated scenarios. A better approach to show that our method works would be to prove that convergence always happens, e.g. by showing that there is a Nash equilibrium. To ensure convergence, we can apply the multi-agent RL theory, requiring some kind of communication among agents (Yau et al. [2010]; Khan and Rinner [2012]; Baidas [2014]), however, this is left as future work.

MOS evaluation Table 5.8 shows the number of steps until convergence. UCB1 presented in all cases a larger number of steps (at least ten times larger than Q-learning). The table also shows that the convergence time increases as the stations demand more information when using Q-learning. However, the results for both learning methods are still better than the baselines, since, in these cases, the stations never reached the maximum MOS value simultaneously.

Table 5.8: Web QoE – MA experiment – Number of steps to reach maximum MOS

Client 1	Client 2	Q-Learning		UCB1	
		Number of Steps	Median	Number of Steps	Median
light	light	**48.3** ± 12.64	40.0	1356.4 ± 233.31	1515.0
light	average	**37.1** ± 7.23	40.0	1309.7 ± 243.56	1255.0
light	heavy	**63.2** ± 24.05	45.5	1123.9 ± 213.27	1205.0
average	average	**43.8** ± 5.35	44.5	860.2 ± 171.32	803.5
average	heavy	**71.4** ± 42.17	46.0	804.4 ± 150.03	840.5
heavy	heavy	**63.7** ± 14.52	55.0	635.5 ± 108.77	679.5

It is interesting to note the variation of the confidence interval for three combinations in the experiments with Q-learning: (i) when both stations are downloading from the *light* site, this variation occurs because there are three runs where the algorithm took, respectively, 104, 203 and 393 steps to converge; (ii) in the *light–heavy* case, we notice four runs that took 100, 108, 240, and 327 steps; and (iii) in the *average–heavy* case, two cases took 316, and 598 steps.

We observe that, running this experiment without the control algorithm, only one of the runs reached the maximum MOS combination, but it took 443 steps to reach the limit. The *Fixed baseline* runs were terminated when the experiment time limit was reached (30 minutes). The results with UCB1 when there is a *light* site showed the longest convergence times, if compared to Q-learning. Observing the actions taken by the agents, in many cases both selected similar actions, *i.e.*, they selected equal or interfering channels. As a consequence, this generated improvements for one station, but worsened the other, or the actions worsened the MOS in both stations.

The scenario shown in the multi-agent (MA) experiment resembles the behavior of a group of agents in a collaborative multi-agent setting known as independent learners. The actions of the other agents are ignored in the representation of the local action-value function, and, while learning, these agents change their behavior, and the system may become non-stationary from the perspective of an individual agent, leading to oscillations (Kok and Vlassis [2006]). We can add an adapted payoff propagation method (Kok and Vlassis [2006]) to improve the convergence time and prevent network oscillations. Although we did not observe oscillations in our experiments, the message exchange might improve the results discussed in the previous paragraph.

Figure 5.12 shows the CDF of the MOS values during the MA experiment. In this scenario, the SDN controllers improve the MOS in most of the cases. The exceptions occur in less than 5% of the *light–light*, and *average–average* combinations using Q-learning. UCB1 outperforms both baselines in all cases. It is important to highlight that *Fixed baseline* used maximum transmission power, and there is a high probability of APs being in non-interfering channels[9].

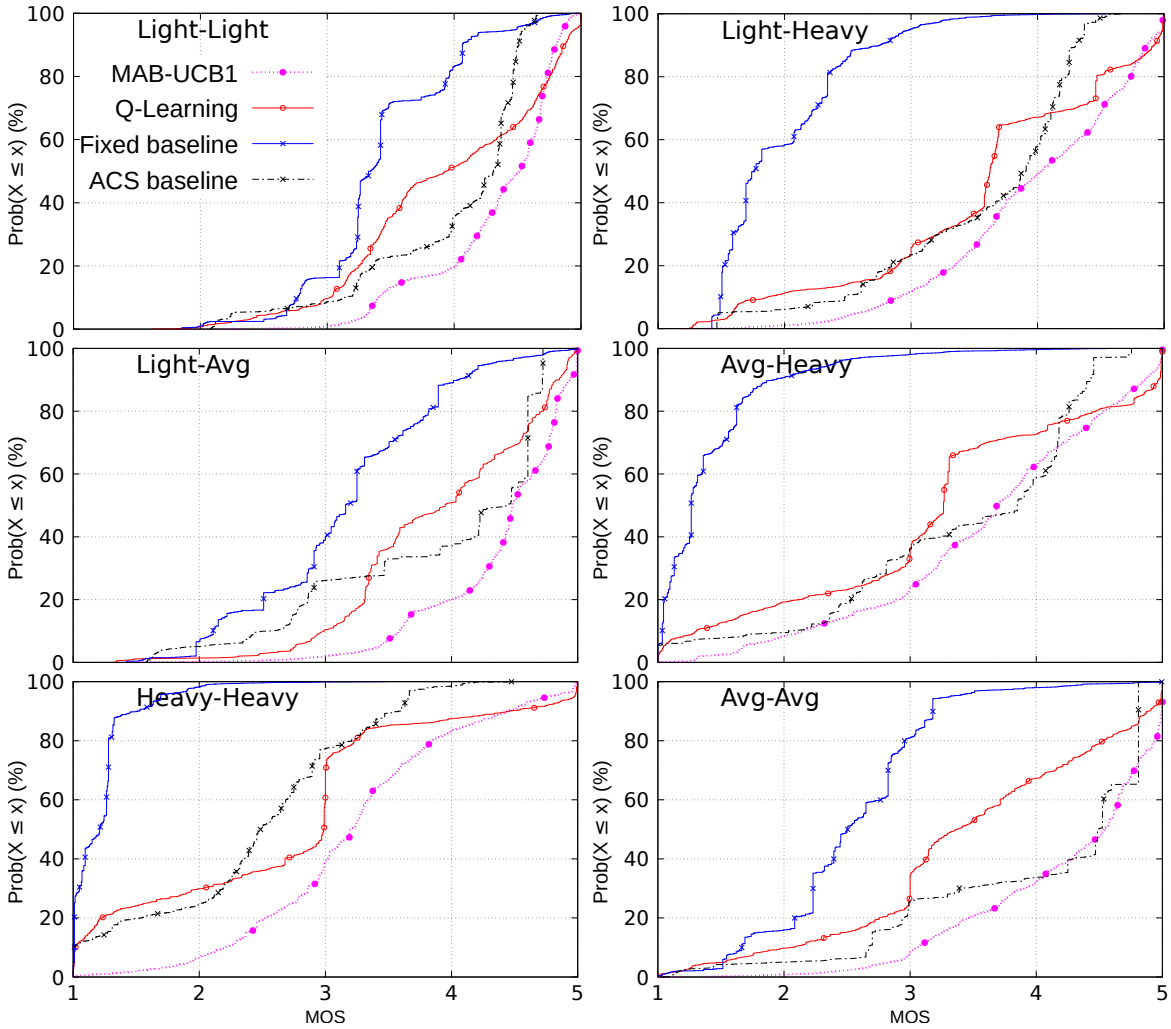

Figure 5.12: Web QoE – MA experiment – Cumulative distribution of MOS

Page load time measurements We also evaluated the impact of the control algorithm on the average page load time of both stations. The results for each combination are shown in Table 5.9, for both RL methods, and the *Fixed baseline*. In all combinations, the page load time is lower when a control loop is used.

[9] $P(\text{non-interfering}) = 1 - P(\text{interfering}) = 1 - 3 \times \frac{C(3,2)}{C(11,2)} > 0.836$.

Table 5.9: Web QoE – MA experiment – Average Page Load Time using Q-Learning and UCB1 compared to *Fixed baseline*

Client 1	Client 2	*Fixed baseline*	QL		UCB1		UCB1 / QL
		Page Load (s)	Page Load (s)	Gain (%)	Page Load (s)	Gain (%)	(%)
light	light	5.7 ± 0.0	**2.6** ± 0.17	121 %	3.3 ± 0.9	73 %	-22 %
light	average	13.6 ± 0.2	**7.7** ± 0.42	76 %	9.6 ± 1.6	41 %	-20 %
light	heavy	9.7 ± 0.1	6.2± 0.52	56 %	**5.4** ± 0.2	79 %	15 %
average	average	20.9 ± 0.1	**7.1** ± 0.43	194 %	9.0 ± 2.6	131 %	-21 %
average	heavy	17.1 ± 0.1	10.9± 1.08	56 %	**8.5** ± 5.3	101 %	29 %
heavy	heavy	13.1 ± 0.1	9.8± 2.02	33 %	**8.0** ± 5.0	63 %	22 %

UCB1 showed large confidence intervals for the number of steps until convergence, reflecting the difficulty of the algorithm to consistently obtain good results. However, we can observe that in three situations (mainly with *light* sites), Q-learning presented better results. However, due to the confidence intervals, in cases where the average nominal value of UCB1 is lower than Q-learning's, these values can be considered statistically equal.

Overhead of the control loop During the experiment 278 ± 55 control messages were exchanged between the controller, and the AP, on average, causing an overhead of 743.8 *kilobytes*. Note that the MA experiment exchanged more messages than the SA experiment, on average. However, the confidence interval in both cases overlap, so statistically both values are equal. 9.8 MB were downloaded for the experiment with lower traffic, while 77 MB were downloaded for the higher traffic scenario using Q-learning.

Table 5.10: Web QoE – MA experiment – Comparing Regret

Regret by Combination of Type	UCB1	QL	*Fixed*	*ACS*
light-light	**0.675** ± 0.006	1.070 ± 0.045	1.579 ± 0.007	1.011 ± 0.007
light-average	**0.610** ± 0.006	1.101 ± 0.063	1.843 ± 0.009	1.102 ± 0.009
light-heavy	**1.034** ± 0.008	1.423 ± 0.048	3.019 ± 0.006	1.433 ± 0.006
average-average	**0.771** ± 0.010	1.535 ± 0.060	2.477 ± 0.008	1.004 ± 0.008
average-heavy	**1.388** ± 0.012	1.790 ± 0.063	3.583 ± 0.006	1.611 ± 0.006
heavy-heavy	**1.770** ± 0.013	2.334 ± 0.058	3.780 ± 0.003	2.581 ± 0.003

Analyzing the regret Table 5.10 shows the average regret with a confidence interval of 95% for all combinations. UCB1 showed lower regrets than Q-learning, and the baselines. Further, UCB1's confidence intervals are narrower. Finally, the confidence intervals do not overlap in any combination for the experiments using Q-learning, and UCB1, thus UCB1 outperforms Q-learning in this experiment.

5.5.3.3 Centrally-controlled experiment

This scenario simulates a place with multiple APs managed by a single administrator, such as in a company, a campus, etc. We will show that a single control loop generates benefits for stations connected to the wireless network, even when there are neighboring APs, since the controller makes coordinated decisions to improve the global reward.

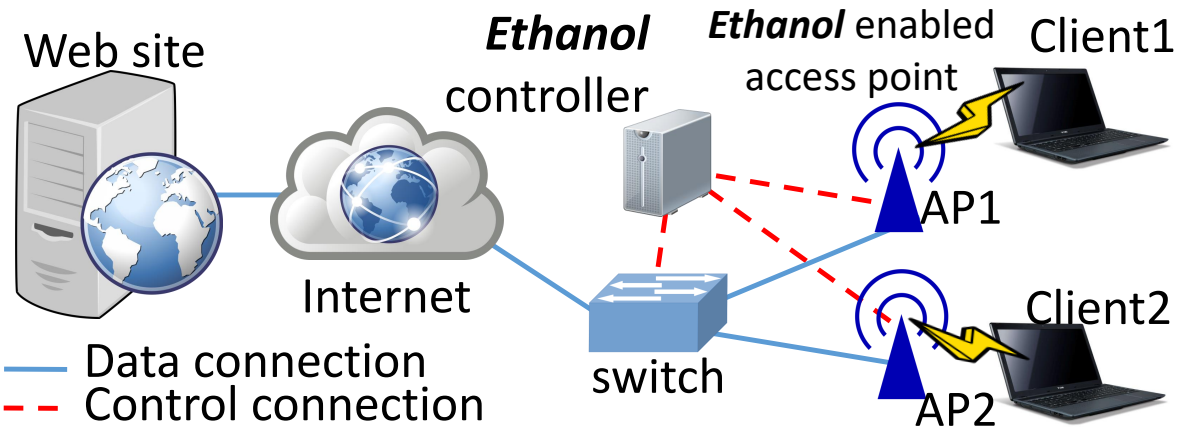

Figure 5.13: Web QoE – CA experiment - Setup

The setup is shown in Figure 5.13. The algorithm is executed on a state space that simultaneously contains both stations, and therefore the system seeks a global reward equivalent to the maximum average of the rewards obtained by the stations. As a consequence, the controller has to handle the cross interference that is generated by the transmissions of the whole wireless network.

Table 5.11: Web QoE – CA experiment – Number of steps to reach maximum MOS

Client 1	Client 2	QL		UCB1	
		Number of Steps	Median	Number of Steps	Median
light	light	**73.6** ± 17.77	67.0	383.5 ± 168.94	223.0
light	average	91.7 ± 12.35	88.0	**60.9** ± 37.33	2.0
light	heavy	**76.3** ± 8.66	3.0	226.2 ± 57.65	193.0
average	average	114.7 ± 30.73	764.0	**18.9** ± 6.38	3.0
average	heavy	**83.4** ± 14.67	72.0	211.6 ± 42.50	202.0
heavy	heavy	**77.0** ± 9.71	72.5	199.7 ± 37.65	184.0

MOS evaluation This scenario also evaluates a second baseline. Alves et al. [2018] proposed a greedy algorithm to adjust the channel of WLAN networks, which they called HNR. The proposed algorithm evaluates the channel quality, and the throughput in the AP's wireless interface to decide the best channel considering the interference generated by the other controlled APs as well as the interference of devices in the

neighborhood. We tuned the threshold parameter used in the algorithm for the CA scenario. To define the threshold for each combination, we ran the experiment 20 times for 30 minutes, and the best threshold among these runs was used to collect a baseline, which we called HNR.

Table 5.11 shows the same information as in Table 5.5, but for the CA setup described above. In this experiment, the algorithms also achieve good convergence time using Q-learning, but UCB1 showed worse convergence times. Both the mean, and the confidence intervals for UCB1 are bigger than for Q-learning, except in the *light–heavy* and *average–average* combinations.

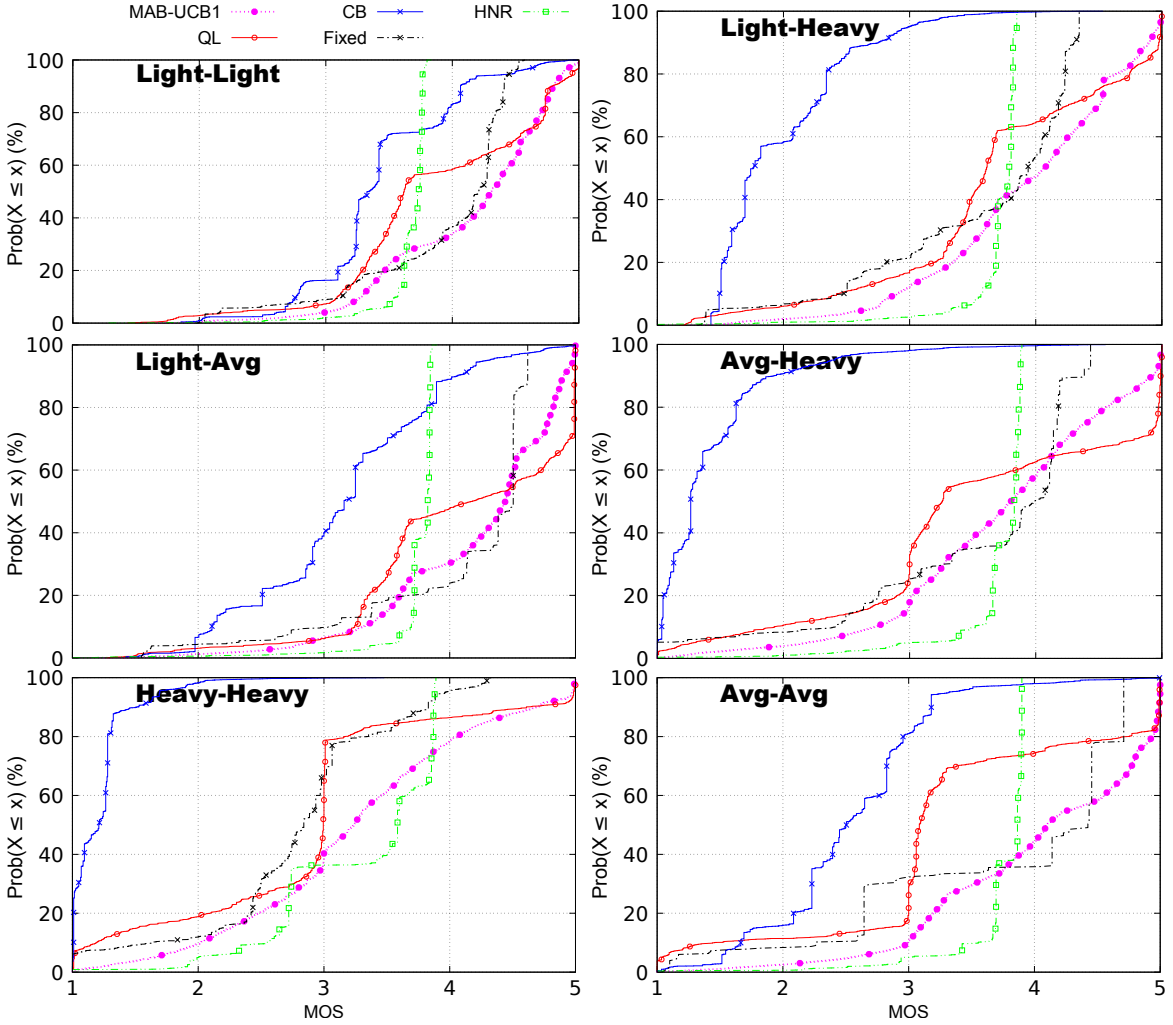

Figure 5.14: Web QoE – CA experiment - Cumulative distribution of MOS

Figure 5.14 shows the CDF of MOS values. Both learning algorithms improve the MOS with regards to the *Fixed baseline*, due to the coordination provided by the SDN controller. With the *heavy–heavy* combination, Q-learning showed lower MOS than *Fixed baseline* for 10% of the cases. This occurred only for MOS values less than

1.8. We observe for the other 90% of the cases that the MOS of Q-learning is greater than that of the *Fixed baseline*. In cases where one of the accessed sites is *light*, Q-learning also showed a MOS that is larger than *Fixed baseline* in more than 97% of the cases. If compared to the *ACS baseline*, Q-learning obtains better results in half of the cases. It is important to highlight that this baseline was manually configured to beat Q-learning. Altering the scanning interval or the network setup affects the baseline performance, while Q-learning can learn how the environment behaves, and adapt the APs' configuration accordingly. The overall MOS in this experiment using Q-learning is, on average, improved by 55.1% if compared to the *Fixed baseline*, and 2.3% when compared to the *ACS baseline*. However, UCB1 improves on average by 71.7%, and 13.3%, respectively.

The HNR performance depends on a flow threshold that is used to select the channel. This is a weak point of their proposal, as this value varies for each network configuration. Despite that, HNR provides a reliable and consistent performance for their clients, and outperforms Q-learning, and UCB1 (by a narrow margin) in the *heavy–heavy* case, as we will show when analysing the regret.

Table 5.12: Web QoE – CA experiment - Average Page Load Time using Q-Learning and UCB1 compared to *Fixed baseline*

Client 1	**Client 2**	***Fixed***	**QL**		**UCB1**		**UCB1 / QL**
		Page Load (s)	**Page Load (s)**	**Gain (%)**	**Page Load (s)**	**Gain (%)**	**(%)**
light	light	5.7 ± 0.0	**4.4** ± 0.73	29 %	4.5 ± 0.7	25 %	-3 %
light	average	13.6 ± 0.2	9.8 ± 7.26	38 %	**6.7** ± 5.4	101 %	45 %
light	heavy	9.7 ± 0.1	5.2 ± 2.07	87 %	**4.0** ± 7.0	140 %	28 %
average	average	20.9 ± 0.1	**17.4** ± 6.74	20 %	18.3 ± 4.8	14 %	-5 %
average	heavy	17.1 ± 0.1	8.8 ± 6.59	94 %	**8.7** ± 5.5	95 %	1 %
heavy	heavy	13.1 ± 0.1	7.8 ± 1.59	67 %	**7.2** ± 10.4	81 %	8 %

Page load time measurements The proposed control loops obtained lower average page load times than *Fixed baseline* in most cases, as shown in Table 5.12. Notice that due to the overlapping confidence interval, Q-learning, and UCB1 must be considered statistically equal. Therefore, the gains shown in the right-most column (called "UCB1 / QL") cannot be used to say that one algorithm outperforms the other.

Table 5.13 compares the gain obtained in the page load time between the MA, and CA experiments. Each row in the table shows one of the combinations of the three different types of sites. The table shows two sets of columns: the leftmost contains the comparison between the experiments using Q-learning, and the rightmost compares the experiments with UCB1. Each group of columns presents the gain (*i.e.* the reduction of page load time) considering both learning control loops, and a column showing if

the confidence interval for the average values of the MA, and CA experiments overlap. If they overlap, a checkmark in the column indicates that both results should be considered statistically similar. We observed that MA obtains better results than CA, when compared to *Fixed baseline*, since in cases where the gain is negative, the results should be considered statistically equal. The same result is observed for UCB1.

Table 5.13: Comparing the gain in the page load time between MA and CA experiments

Client 1	**Client 2**	**MA x CA**		
		QL Gain (%)	**UCB1 Gain (%)**	**Overlap**
light	light	71.21 %	38.61 %	
light	average	27.11 %	-29.91 %	✓
light	heavy	-16.81 %	-25.51 %	✓
average	average	145.21 %	103.21 %	
average	heavy	-19.51 %	3.11 %	
heavy	heavy	-20.61 %	-10.21 %	✓

Overhead of the control loop During the experiment 402±79 control messages were exchanged between the controller, and the AP, on average. This caused an overhead of 1070 *kilobytes*. Note that in the CA experiment more messages are exchanged than in the MA experiment because of the longer convergence times. The confidence interval in both cases overlap, so statistically the algorithms are equivalent. For the experiment with lower traffic, 1.4 MB were downloaded, while 21 MB were downloaded by the stations for the situation of higher traffic when using Q-learning.

Table 5.14: Web QoE – CA experiment - Comparing Regret

Regret by Combination of Type	**UCB1**	**QL**	***Fixed***	***ACS***	***HNR***
light-light	**0.869** ± 0.005	1.153 ± 0.032	1.579 ± 0.007	1.082 ± 0.007	1.337 ± 0.007
light-average	**0.788** ± 0.004	0.899 ± 0.032	1.843 ± 0.009	0.908 ± 0.009	1.252 ± 0.009
light-heavy	**1.055** ± 0.006	1.306 ± 0.038	3.019 ± 0.006	1.421 ± 0.006	1.302 ± 0.006
average-average	**0.969** ± 0.006	1.685 ± 0.037	2.477 ± 0.008	1.273 ± 0.008	1.265 ± 0.008
average-heavy	**1.260** ± 0.007	1.432 ± 0.046	3.583 ± 0.006	1.469 ± 0.006	1.297 ± 0.006
heavy-heavy	1.737 ± 0.008	2.135 ± 0.042	3.780 ± 0.003	2.239 ± 0.003	**1.710** ± 0.003

Analyzing the regret As shown in Table 5.14, UCB1 presented a better performance in respect to the regret than Q-learning, *Fixed baseline*, and *ACS*. It is interesting to notice that in the *average–average* case, the regret is almost half the value obtained by Q-learning. This behavior occurred also in the MA case, as shown in Table 5.10, because about 50% of Q-learning MOS values remain near to three, while UCB1 keeps improving its values.

We also observe that during the learning process, the APs sometimes reach a local maximum in situations where the MOS is not maximized. In fact, the increase in the number of controlled APs implies in an increase in the number of states, and thus in the Q-matrix size. The size of the Q-matrix is $|Q| \propto |A|\,|S|$, so for the MA experiment, each controller explores (and fills) simultaneously both Q-matrices of size $|Q|$. However, in the CA experiment there is only one Q-matrix with size $|Q'| \propto |2A|\,|2S| \propto 4\,|Q|$.

Analyzing the executions in both scenarios, we observed that on average the agent explores more states in the MA scenario than in CA. In the first scenario, the Q-learning agent explores, on average, about 0.0042% of the total space, while in MA the agent explores 0.0051% of the space, as shown in Table 5.15. Further, the search space for UCB1 is smaller, which may explain the more favorable regrets for this method.

The small percentage of explored states indicate that (1) in MA the agent needs to explore further in order to achieve a result similar to CA; and (2) the Q-matrix is sparse, thus the agent is unaware of the effect of similar actions. Thus, using function approximation in the control policy could lead to a better performance. However we leave this for future work.

Table 5.15: Percentage of the search space explored by each method for each use case

Use case	**State space exploration (%)**	
	Q-learning	**UCB1**
SA experiment	0.0188 %	0.260 %
CA experiment	0.0042 %	0.162 %
MA experiment	0.0051 %	0.302 %

5.6 Discussion

The proposed RL methods use a QoE metric to adjust the settings of the controlled devices. For that reason, the quality of this metric is essential for a good system performance. The metric used in this chapter uses general information about the state of the wireless connection, so it has (at least) three points that can be improved for a production system: (1) it cannot discriminate flows, so the traffic of other applications on the station is counted as web browsing; (2) the metric uses the station bitrate, which is mainly related to the quality of the connection, but does not identify if the station is transferring data, and (3) the current metric does not allow the discrimination of several web streams from the same user, *i.e.*, if a user simultaneously accesses, for example, a *light*, and a *heavy* website. This last issue is not that important for the

user, because a *light* site being treated as *heavy* will be benefited by the stricter QoE requirements of the *heavy* category.

In general, our results show that there is a trade-off when choosing UCB1 or Q-learning. This occurs because UCB1 presents smaller regrets. However, the rate of convergence of Q-learning is faster than that of UCB1. We see from the literature that in many cases Q-learning provides better results than MAB for various problems. In ours, the result is the inverse, however, this occurs for several factors. We observe that Q-learning, given the same period of execution of UCB1, explores a much smaller fraction of the state and action space. Thus, UCB1 gets better results considering QoE. In addition, the Q-matrix is very sparse, which makes the Q-learning agent unaware of adjacent positions in the Q-matrix that could lead to improved results, and since the exploration is random, the algorithm provides worse decisions. As a consequence, UCB1 would do better in more stable environments, while Q-learning would work better in more dynamic networks.

Our scenario shows control applied over up to two AP. This corresponds to a small office or a home that is immersed in an area full of interference. They are important because, in all cases, the control has to adapt to interference provided by other APs. Besides that, the configuration tested was design to provide high cross-interference. The testbed can be rearranged by separating the APs. There are two consequences when we separate them: (1) if they are in the interference range, the controller has to cope with the co-interference, which decreases as they are taken apart; and (2) if one cannot interfere with the other one, then this case reduces to the SA scenario described in Section 5.5.3.1. However, further research should be made considering: (1) more stations per AP; (2) more flows per station; and (3) scenarios with four or more APs, so it is hard for the controller to find a non-interfering channel to allocate the forth AP and so on.

5.7 Summary

This chapter presented learning control loops for transmission power, and channel selection, based on SDN and RL, which improve Web QoE.

A semi-supervised learning algorithm classifies web sites based on their similarity to a small number of known sites. It uses a two step classification algorithm (on-line, and off-line) to provide faster response to a user request. Results show that we can obtain a good score in the test set ($87\% \pm 1\%$) with 500 unlabeled examples,

and 15 labeled examples, with an execution time of about one second using on-line classification.

Table 5.16 shows the best results obtained by both learning methods (Column 1) for each of the baselines (Column 2). The values presented in the other columns represent the gain (bigger is better) when the learning method is applied in relation to the baseline. The rows with the HNR baseline only show results in the last column because this baseline was run only in the CA experiment. Recall that the method used by HNR aims to reduce co-interference between the controlled APs, so the CA experiment is the only scenario where this can occur. Observe that a 100% reduction means getting the ideal regret, that is, regret equal to zero.

Table 5.16: Best regret results when applying the proposed learning methods compared to the baselines

Learning method	**Baseline**	**SA**	**MA**	**CA**
UCB1	**Fixed**	84%	69%	65%
	ACS	74%	45%	26%
	HNR	-	-	35%
QL	**Fixed**	76%	53%	60%
	ACS	53%	10%	8%
	HRN	-	-	28%

Based on the site classification, a closed loop uses RL to optimize Web QoE. We evaluated the proposed control loops using a prototype in three case studies, in which a number of APs are controlled by a central controller or are independently controlled. Results show that our control loops improve the QoE experienced by the clients. Further, with a small number of iterations, our closed control loop obtains the maximum MOS for the clients' Web demands. UCB1 reduces the average regret by at least 45% in the worst case, and 84% in the best case. Q-learning, on the other hand, reduces the average regret by at least 8% in the worst case, and 76% in the best case. The page load time was reduced by 0.2x and 0.17x, using UCB1 and Q-learning, respectively, in the worst case, and 0.71x and 0.69x in the best case, when compared to the baselines.

In order to ascertain how long the network would need to converge, we evaluated the distributions of MOS for different periods of the experiment. In our setup, the learning algorithms reach a stationary return after only 200 seconds of operation (recall Figure 5.10). This result is despite the fact that the agents explore less than one percent of the state space during the experiments.

Chapter 6

Wireless Control using RL for Video QoE

This chapter proposes an automatic control loop that improves the user's QoE for a video application. The controller can change physical and link layer parameters of the wireless network, such as transmission power, channel, and queue assignments. To perform this work, a QoE metric estimates the user satisfaction when watching a video. We used the features of a metric from the literature to train a new regressor that infers the MOS based on information collected at the client.

The RL methods applied in the previous chapter present some generalization problems due to how the Q-matrix is constructed, *i.e.*, the agent has no clue which action to take for the states that it has not seen before. The agent has no ability to estimate (extrapolate) the Q-value for unseen states. To deal with that, this chapter explores function approximation to estimate the Q-value. Since the update occurs on the approximation function weights, the function extrapolates the Q-value for unvisited states. If the given value is in the vicinity (even a larger vicinity) of the correct value, this approach gives a better estimate than a random Q-value initialization. Table 6.1 lists the key symbols used in this chapter, and provides a brief explanation of them.

The next section highlights the contributions of our proposal. The architecture used for video QoE was shown in Section 4.3. The following section (Section 6.2) shows how the reward is obtained, how a Video QoE is trained to our particular setup, and the implementation, explaining what actions are available to the control loop, how the states are represented in both algorithms, and how the Q-function was approximated. Next, Section 6.3 describes the experiment used to test the control loop using DQL with the proposed QoE metric, and analyzes the results obtained. The last section summarizes this chapter.

Table 6.1: Key symbols used in the chapter

Symbol	Description
s, s	the current state
s', s'	the next state (after the action a)
a, a	the selected action in state **s**
r, r_t, r	the reward received when in state **s** perform the action **a**
t	the current timestep
ϵ	controls the exploration rate in the ϵ-greedy algorithm
Q^*	the optimal Q-value
$Q(s, a, \theta)$	the estimated Q-value for state s and action a using the parameters θ
α	the learning rate
γ	the discount rate
F_t	the fairness index at time t
U_j	he number of users in the j-th AP
N	N is the number of controlled APs
M	the total amount of users connected to all the controlled APs
$MOS_t^{(i)(j)}$	the MOS perceived by the station j connected to AP i at the timestep t
S	the set of all MOS values perceived by the stations in time t
$q(R_n)$	some function $q(\dot{})$ applied to the variable R_n
tx_power	the transmission power is in the range of 1 dBm up to 15 dBm
ch_number	the transmission channel index (1 to 11)
d	the dilation factor in temporal convolutional neural network (TCNN)
k	the kernel size in TCNN
C	weight parameter that puts more or less importance to the fairness index in the reward function (Equation 6.3)
$reward_t^{(i)}$	the value of the reward function evaluated at time t for the AP i

6.1 Contributions

Mobile subscriptions grow exponentially, and are expected to reach 8.9 billion by 2023 (Ericsson AB [2018]). Video was 73% of Internet traffic in 2016, and that figure will rise up to 82% in 2021 (Cisco [2017]). This trend implies in a drastic hike on the demand for capacity. As video streaming has become the main contributor in an always increasing Internet traffic, it is a challenging task to meet the user's expectations for this type of flow. This chapter intends to meet the Specific Objective 4 presented in Section 1.3. Therefore, we aim to:

1. Refine a QoE metric in the literature that uses features obtained from the video application in the client. We propose a new regressor, reusing the features from the literature in order to cope with the requirements of our control loop.

2. Compare the proposed metric with PSNR (Huynh-Thu and Ghanbari [2012]), a common metric adopted in the literature to evaluate the MOS of a video stream;

3. Propose a reward function based on the MOS, and fairness;

4. Analyze the use of DQL in the control of the wireless network parameters;

5. Simplify the system complexity using interpretability techniques to analyze the proposal.

The results shown in this chapter are not published yet. We are now writing the paper to submit to a journal.

6.2 RL-based control loop

RL uses a reward to guide the search for better settings on the controlled APs. The reward function is explained in the next section. Section 6.2.3 shows the states, and actions in our implementation. The deep network used to implement the Q-value approximation is also discussed.

6.2.1 Reward

RL methods use a scalar value to drive the goal, which in our case is to improve the quality perceived by the user. Further, we want to incorporate fairness into our goal. As an example, we will consider as a better distribution of MOS the case where both stations perceive a MOS equal to 3, when compared to the case where one station gets $MOS = 5$, and the other gets $MOS = 1$, which also averages to 3. Thus, the reward function proposed in this section combines the QoE metric with a fairness indicator.

Since we want to optimize the QoE for the entire system, we initially considered that the QoE values of the video should be averaged, providing the following reward at time t:

$$avg\_reward_t = \frac{1}{M} \sum_{i=1}^{N} \sum_{j=1}^{U_j} MOS_t^{(i)(j)} \tag{6.1}$$

where N is the number of controlled APs, U_j is the number of users in the j-th AP, $M = \sum_{i=1}^{N} U_i$ is the total number of users in the N APs, and $MOS_t^{(i)(j)}$ is a video MOS metric (explained in the next section), obtained from time t on AP i for user j. This reward function may present an imbalance among clients, as it does not consider the

dispersion of MOS values. For example, consider the following MOS values: $\{5, 5, 1\}$. The reward for this distribution is 3.7, however one of the devices realizes the lowest MOS possible, while the others are at the maximum level.

To reduce this problem, we modified the reward function to consider the dispersion of results. Egger et al. [2012] and Hoßfeld et al. [2016, 2017, 2018] proposed two approaches to fairness: (a) the resources are shared evenly among the devices or users; and (b) a utility function is used to scale the results. We adopted the second approach in this thesis, because we want the reward function to account for the MOS as well. The proposed reward function is discussed below.

The reward function is based on the MOS value, and a fairness index. In our work we will consider the fairness index defined in Hoßfeld et al. [2016], which we call F_t. This fairness index is defined as

$$F_t = 1 - \frac{2 \times se_t(S)}{H - L} \tag{6.2}$$

where $se_t(S)$ is the standard deviation of the set $S = \{MOS_t^{(i)(j)}, \forall i, j\}$, which represents all MOS values perceived by the stations in time t, $H = 5$ and $L = 1$ (since H and L are the higher and upper bounds of the samples to be considered). This index assumes values between 0 and 1, inclusive. The fairness index is zero in the most unbalanced situation, i.e., when 50% of the values have the lowest MOS, and the other half are at maximum. The value is one when all MOS values are equal.

Putting it all together, we proposed a reward function that is the average MOS calculated by Equation 6.1 scaled by a penalty factor related to the fairness index, where $C \in \mathbb{R}^+$ is a weight parameter that puts more or less importance to the fairness index.

$$reward_t = 1 + (avg\_reward_t - 1) \times F_t^C \tag{6.3}$$

This reward function produces the reward shown in Figure 6.1 when $C = 0.4$. Notice that this function shows a gradient upwards, and to the right, thus it stimulates the agents to seek better MOS, and also stimulates the equilibrium among the rewards received by the agents. We used Equation 6.3 with $C = 0.4$ as the reward function in our experiments. The value of C was selected empirically. We intend to evaluate the impact of C on the average MOS, and fairness obtained in a future work.

Consider now the case where there are two APs, which are in the same vicinity, and channel. One AP has *avg_ reward* equal to 1, and the other AP equal to 5. If we consider $reward\_t$ as shown in Equation 6.3, both APs will be penalized by F_t.

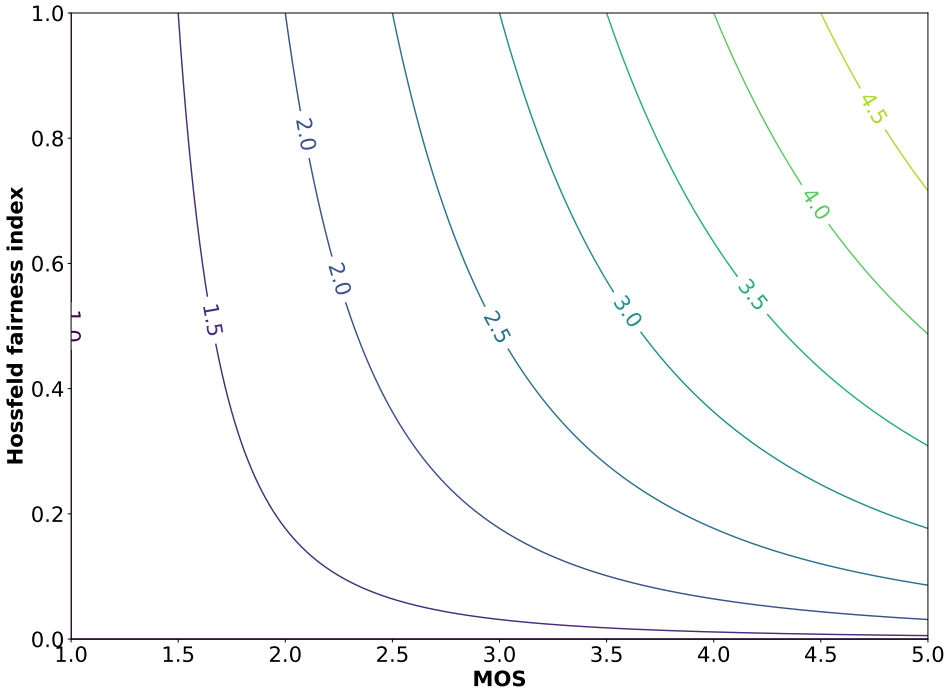

Figure 6.1: Hossfeld penalty applied to MOS values using $C = 0.4$

However, we only need to penalize the controlled AP where the perceived QoE is higher than the average, stimulating the learning mechanism to take actions that frees the medium, so that the other AP can get a higher QoE. Notice that only APs that are on the same channel compete for capacity, because they are co-interfering. In our experiment, two APs are co-interfering if they are on overlapping channels, and at least one of them detects the other using "Neighbor Report" messages, as defined in the IEEE 802.11 protocol. The final formulation of the reward for a co-interfering AP is:

$$reward_t^{(i)} = \begin{cases} avg\_reward_t & if\ MOS_t^{(i)} \leq avg\_reward_t \\ 1 + (avg\_reward_t - 1) \times F_t^C & otherwise \end{cases} \tag{6.4}$$

where $MOS_t^{(i)}$ is the average MOS perceived by the users of the AP i, which is obtained using $MOS_t^{(i)} = \frac{1}{U_i} \sum_{j=1}^{U_i} MOS_t^{(i)(j)}$, and $F_t^{(i)}$ is the fairness index calculated using only the set $S^{(i)} = \{MOS_t^{(j)}, \forall j \in Neighborhood(i)\}$. $Neighborhood(i)$ is a function that returns the APs in the neighborhood of AP i.

6.2.2 QoE metric in the literature

We have evaluated some video QoE metrics from the literature in order to obtain a MOS predictor for our experiments. As we built a real-life prototype, it is not always possible to obtain all the necessary parameters for the calculation of the proposed QoE metrics in the literature. In this way, we limit ourselves to those metrics whose parameters could be obtained on a testbed during the experiment, and which did not demand a large load on the client nor on the wireless network.

In this thesis, we use two MOS metrics described below. The first metric called MOS_PSNR analyzes the errors between the received and transmitted video images, and also takes into account the duration of video stops. This metric was used as a reference, because it can only be obtained offline. Thus a second metric was considered (MOS_CLIENT). It is obtained using features provided by the video application in the wireless client. Thus it requires changes to the client so that it provides the controller with the desired information. We leave to a future work to obtain, and test a metric with features coming only from the core network. Both metrics are detailed below.

MOS_PSNR PSNR (Huynh-Thu and Ghanbari [2012]) is a simple, and well accepted metric in the literature. This metric was calculated using Video Quality Measurement Tool (https://mmspg.epfl.ch/downloads/vqmt/), and we provide a preconfigured docker container in GitHub (https://github.com/h3dema/ubuntu-vqmt). PSNR allows the automatic evaluation of the MOS, and therefore there is no need to inquire the user about her perception. This estimator compares the original video (sent by the video server) with the video received by the station. Thus, this comparison can only be performed off-line. PSNR identifies the differences between the frames from both videos, and determines how much the received video has changed in the transmission. This metric can be converted to MOS using mappings found on the literature. Klaue et al. [2003] and Piamrat et al. [2009] use Table 6.2 to convert PSNR into MOS, which was also used in our work.

Table 6.2: Mapping PSNR to MOS scale

PSNR in dB	**MOS**	**Quality**
> 37	5	Excellent
31 – 77	4	Good
25 – 31	3	Fair
20 – 25	2	Poor
< 20	1	Bad

Because PSNR evaluates the difference between the reference video, and the received video, we consider the following cases to produce MOS_PSNR:

1. We set the reference resolution to ultra high definition (3,840 x 2,160 pixels), which is the highest resolution available for download from the video server. So if the video is playing at a lower resolution, PSNR will understand it as a loss of quality.

2. Video stops also generate quality reductions. They are counted as a stalled MOS, which we define as one during the duration of the stop ($idle\_time_t$). For example, consider receiving two one-second chunks at full resolution ($MOS = 5$), but suffering from a 2-seconds stop between the chunks, which implies that the final MOS in this period is halved ($MOS = 3$).

Thus, the MOS_PSNR metric in interval t can be computed as a weighted mean of the perceived MOS in the interval, and the stalled MOS:

$$MOS\_PSNR_t = \frac{M_t(PSNR_t) \times playing\_time_t + idle\_time_t}{total\_time_t} \tag{6.5}$$

where $total\_time_t = playing\_time_t + idle\_time_t$ is the total time of sampling in the interval t, $playing\_time_t$ is how much time (in seconds) the chunks ran in the video player, $idle\_time_t$ is how much time (in seconds) the video stopped during the interval t, and $M_t(PSNR_t)$ is the MOS value obtained by converting the calculated video PSNR in the interval t using Table 6.2.

A small poll was done when the research for Moura et al. [2019b] was in progress. In this poll about 10 interviewees rated the quality of a 1-minute video. The results from this poll correlated in more than 90 % with the results with MOS_PSNR.

MOS_CLIENT A video QoE metric can be obtained exclusively through features obtained in the client, such as the metric proposed by Yin et al. [2015b]. The QoE metric proposed by Yin et al. [2015b] is defined for a video with N chunks as

$$\text{MOS} = \sum_{n=1}^{N} q(R_n) - \mu \sum_{n=1}^{N} T_n - \sum_{n=1}^{N-1} |q(R_{n+1}) - q(R_n)| \tag{6.6}$$

where R_n represents the relative bitrate of chunk n, and $q(R_n)$ is a function that transforms R_n. T_n represents the rebuffering time that results from downloading chunk n at bitrate R_n. The final term penalizes changes in video quality to favor smoothness. We considered two choices of $q(R_n)$ proposed by the authors:

1. Linear: considers the identity function, *i.e.*, $q(R_n) = R_n$; and

2. Log: captures the notion that, for some users, the marginal improvement in perceived quality decreases at higher bitrates, *i.e.*, $q(R_n) = \log(R_n)$.

Another estimator based on these parameters was proposed by Mao et al. [2017] that fitted those three features using SVM with radial basis function kernel. These authors employ SVM because they identified that the models in Yin et al. [2015b] had a worse correlation in their scenario. Likewise, we proposed a refinement of Yin et al. [2015b]'s model. We used the same features in our model — $q(R_n)$, and T_n. We called this refinement MOS_CLIENT, and the details are shown in Section 6.3.3.

6.2.3 DQL model used in the prototype

This section presents how the states and actions used in the prototype were modeled. The prototype implemented DDQL, and considered that the inputs (states) provide temporal information, *i.e.*, they provide information to evaluate a trend in the environment behavior. With this, the last subsection shows the neural network used to build the Q-function, as well as the hyperparameters used.

In the prototype used in the Video QoE experiments, we used the DDQL algorithm shown in Section 2.6 with ϵ–greedy as the exploration strategy. DDQL uses a function approximation to store the estimation of the expected return for each of the possible states and actions. This section shows what are these states and actions, and how they are coded into the neural network. The last section discusses the neural network used to implement the Q-function.

One of the uses of deep learning in the literature is to cope with the curse of dimensionality. That is not the case in our work, as can be seeing by the state and action definitions shown below. However, deep learning was applied in this chapter due to two characteristics of the system we want to control: (1) using deep learning in the Q-function provides a way to approximate the Q-value for states and action not yet tested; and (2) it allows us to use continuous features in the state representation.

State in DDQL: In order for the system to learn a Q-function that can be generalized, the state must be independent of which AP is providing information to the learner. This allows for transfer learning, which can be tested by adding a new AP to the test environment and seeing if its behavior is better than if it did not know anything. Thus we propose to use the features shown below to represent the state. After the name of the features, we show in brackets if the feature is continuous or discrete.

- $\#stations$ [discrete]: Represents the number of stations connected to the AP in the period defined by the control loop execution time. This period is described in Section 6.3.4.1. This is an integer number greater than or equal to zero;
- $ch1, ch2, ch3, ch4, ch5, ch6, ch7, ch8, ch9, ch10, ch11$ [discrete]: Identifies the AP's transmission channel. This is a one hot encoding of an integer, which identifies the channel index. The number of channels is configured by the user in our implementation;
- tx_power [discrete]: This feature identifies the current transmission power of the AP. It is a real number (tx_power), with two decimal places, that measures the configured power in dBm. This value ranges from 1 to 15;
- $\#num\_neighbors$ [discrete]: It indicates how many controlled APs are in the vicinity of the current AP;
- ch_noise_max [continuous]: indicates the maximum noise detected in the current channel by the AP. This values ranges from 0 to 100%;
- $perc\_phy\_busy\_time$ [continuous]: measures the occupation of the channel, and it indicates the percentage of PHY busy time detected in the current channel;
- Some features are related to the stations connected to the AP:
  - sta_signal_min [continuous]: is the minimum signal received by the AP from the connected stations in the period;
  - $rec\_bitrate\_min$ [continuous]: is the minimum received bitrate from the stations;
  - tx_byte_avg [continuous]: the number of bytes transmitted by the stations in the period;
  - rx_byte_avg [continuous]: is the number of bytes received from the stations in the period.

In this implementation, we were tied by the information that is available using Linux commands, because the southbound interface uses them to configure, and gather information from the the APs' wireless interface. Thus, despite the fact that richer information from the physical and link layers could improve the model, we did not have access to them.

Because we want the algorithm to notice a trend in the change of the features above, we use a neural network suited for time series (described in the next section),

and thus the state is represented by a tuple (s_{t-1}, s_t), that contains the current state and the previous state, where the state entry in period t is the tuple below. We used two periods because the MOS predictor is evaluated using two periods (see Section 6.3.3).

$$\begin{aligned} s_t^{(i)} =& (\#stations, ch1, ch2, ch3, ch4, ch5, ch6, ch7, ch8, ch9, ch10, ch11, \\ & tx\_power, \#num\_neighbors, ch\_noise\_max, perc\_phy\_busy\_time, \\ & sta\_signal\_min, rec\_bitrate\_min, tx\_byte\_avg, rx\_byte\_avg) \end{aligned}$$

Actions in DDQL: The actions are taken in a timely basis, as shown in Section 2.6. Our controller can alter the following parameters:

- the transmission power (tx_power) [discrete]: The transmission power is in the range of 1 dBm up to 15 dBm, in our devices;
- the transmission channel index (ch_number) [discrete]: there are eleven transmission channels, from 1 to 11 (in the 2.4 GHz range in Brazil).

  Thus the action is represented by the following tuple:

$$a = (tx\_power, ch\_number)$$

DDQL details: A wireless network is a dynamic environment. It shows variations in signal strength, neighborhood, user mobility, noise, traffic patterns, etc. Thus, an important factor in order to capture this behavior using RL is to consider the state seen by the agent as a sequence of values, i.e, the learner has to deal with a time series, so it can perceive, for example, if the noise in the current channel is increasing or decreasing. Therefore the approximation function used in DRL must consider this condition.

There are several methodologies for treating temporal dependencies on sequential data, such as using Windowed Multilayered Neural Networks (Alberg and Lipton [2017]) or recurrent neural networks (RNNs) such as long short-term memory (LSTM) or gated recurrent unit (GRU). Another method is to use a variation of a convolutional neural network (CNN) architecture, called TCNN. The CNN's architecture is altered using masked convolutions (Oord et al. [2016]) to cope with the temporal series. The idea is to convolve only on the elements of the current timestamp or earlier in the previous layer, thus generating a causal relationship, meaning that there is no information leakage from future to past. This architecture uses a one dimension fully-convolutional

network architecture. This basic architecture uses dilated convolutions to look back at history with size linear to the depth of the network, and the filter size (Kalchbrenner et al. [2016]). The main advantage of TCNN is that it uses less memory during training than the RNNs because the filters are shared at each layer. Moreover, it is computationally more efficient than RNN because convolutions can be done in parallel, whereas in RNN, later time-steps must wait for their predecessors to complete. To exemplify the difference between these neural networks, for the input shown in Section 6.2.3 and the hyperparameters shown in Table 6.3, the number of trainable parameters in TCNN, LSTM and GRU is 16077, 77205 and 59925, respectively. Thus TCNN uses 79.2% less parameters than LSTM, and 73.2% less than GRU. This reflects in less memory, and computer power footprint.

Figure 6.2 shows an example of a dilated causal convolution with one input layer, one hidden layer, and one output layer. The dilation factors are $d = \{1, 2\}$, which define the number of layers in the network. Notice in the figure that there are two convolutional layers (lines in blue). The figure shows a filter size $k = 3$, which defines how many inputs from the previous layer are processed by the convolutional network.

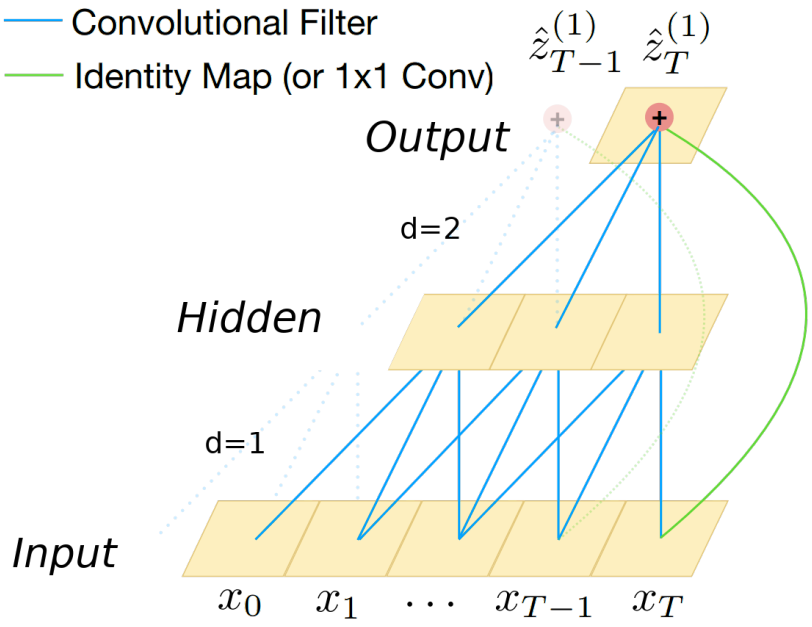

Figure 6.2: An example of a dilated causal convolution with a residual connection in a TCNN. Adapted from Bai et al. [2018].

Our DDQL uses two function approximations, called Q-Network, and Target-network. Both have the same size, defined by the neural network size. Also, because we use memory replay, DDQL uses memory proportional to $2 * O(\text{NN size}) + O(\text{memory replay size})$.

Table 6.3: Hyperparameters

Where	Hyperparameter	Description	Value
TCN	Number of filters	Multiple filters are taken to slice through each layer's input to extract features. This parameter defines the number of filters in each layer.	24
	Padding	This pads the layer's input with zeros in the front so that we can also predict the values of early time steps in the input frame.	causal
	Kernel size	Defines the number of inputs from the layer that are processed by the filter.	2
	Kernel initialization	During initialization, TCNN needs to set the initial values for the network weights. This initial setup can influence the convergence time.	He normal[2]
	Activation function	After applying the convolution, the results pass through a non-linear function, which is called the activation function.	relu[1]
	Number of dilation levels	Defines the number of levels in TCNN. For example, Figure 6.2 has 3 levels.	4
	Dropout	It is used reduce overfitting, thus it is a regularization method in neural networks. This process ignores random sets of neurons during the training phase. The parameter defines how many neurons (in percent) are frozen.	5%
Optimization	Optimizer	It is the iterative method used to minimize loss function.	Adam[3]
	Learning rate (α)	It controls how quickly the learning occurs. Refer to Equation 2.11.	0.002
	Epochs	It controls the number of complete passes through the training dataset.	50
Replay	Episodes	Number of runs the agent takes before executing the replay	10
	Batch size	It controls the number of training samples drawn from the replay memory that are used to update the model's internal parameters.	32
	Capacity	Controls the number experiences that are saved in the replay's circular memory.	2000
Q-Learning	Initial epsilon (ϵ_0)	The value of ϵ_t during initialization.	0.1
	Epsilon decay (ϵ_δ)	If this value is None, the algorithm does not perform decay, otherwise at each timestep $\epsilon_t = \epsilon_\delta \epsilon_{t-1}$	0.995
	Discount factor (γ)	Controls the emphasis over past rewards. Refer to Equation 2.11.	0.95

Notes:
[1] The last layer uses a linear activation function.
[2] Initialization proposed by He et al. [2015]. It draws samples from a truncated normal distribution centered on zero with standard deviation based on the number of input units in the weight tensor.
[3] The method is described in Kingma and Ba [2015].

Table 6.3 shows the hyperparameters, a brief description, and the values used for each hyperparameter in our experiments. The values shown were obtained empirically by the author. We will analyze the effect of changing these parameters on the results in future work.

6.3 Control Loop Evaluation

This section describes the results for video QoE. Section 6.3.1, describes the prototype built to evaluate the control loop. Next, Section 6.3.2 shows how data was collected in the testbed to retrain the MOS predictor. Section 6.3.3 evaluates the models for QoE prediction. We used a search method to find the hyperparameters. Section 6.3.4 presents the evaluation of the control loop as a whole. Section 6.3.5 shows how interpretability is used to analyze the complexity of the state representation, and simplify it. The next section show the simplified state representation, and the results obtained with this model against the full model. Finally, the last subsection discusses the results.

6.3.1 Experiment setup

In the testbed, the controllers, the stations, and the APs are connected to a gigabit Ethernet network, as shown in Figure 6.3. The controllers, and the stations use Ethernet to send and receive control data. The APs use Ethernet to receive control data from the controller, but also to transmit the video flow to the Internet. Each AP provides access to the wireless clients. The APs are started on the same channel and with the same power. The APs' Ethernet interface, and the video server belong to the same public IP subnet.

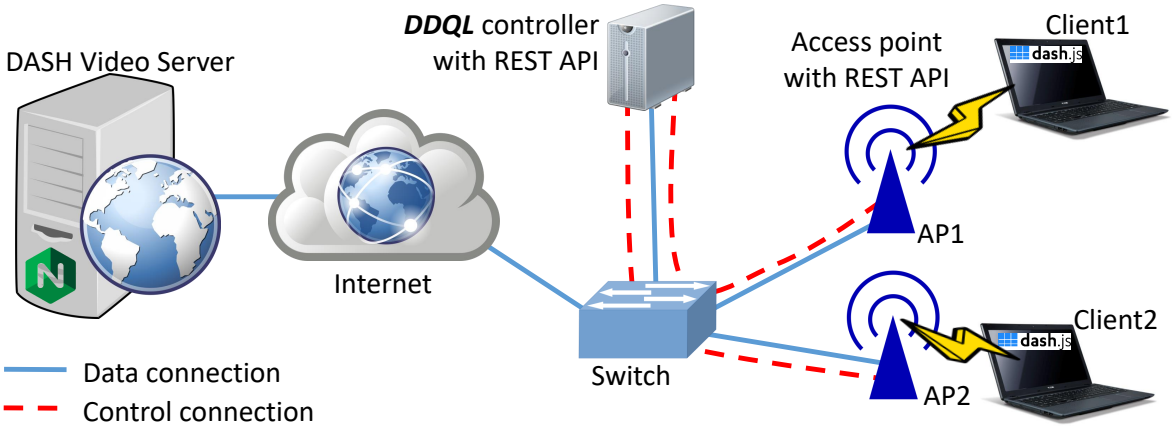

Figure 6.3: Video QoE – Setup

The hardware used as the APs, and the controller are the same as shown in Section 5.5.1. The experiment controller is a dual core Intel(R) i5 PC with 16GB of RAM. The APs are ASUS notebooks with Intel(R) i7 CPU @ 1.80GHz with 8GB of RAM and Atheros AR9485 wireless network adapters. However, the stations' hardware had to be changed, because the hardware used in the Web QoE experiments dit not have enough memory, and compute power to run the high definition video. Thus the stations are

now Intel i5-based computers, with two 3.33 GHz cores, 8 GB of RAM, one Atheros ath9k-based wireless network cards, and one on-board gigabit Ethernet network card. The APs run Ubuntu Linux version 14.04 LTS, and the stations and controller run Ubuntu Linux version 18.04 LTS. The stations run Mozilla Firefox version 69.0.2.

The video server provides an HAS streaming protocol, which uses DASH. DASH works by alternating among different bitrate (or resolution) streams, as shown in Figure 6.4. This type of technology is used in the most popular video streaming systems today (e.g. YouTube, VIMEO and Netflix use DASH). It was the most popular protocol for adaptive video streaming in the transmissions of the 2012 London Olympics by the Belgian operator VRT (WOWZA [2012]). The video server runs in a virtual machine with 2GB RAM and an Intel 2.3GHz processor. The web server is nginx version 1.15.1 (https://www.nginx.com/) with the MPEG-TS Live Module (https://github.com/arut/nginx-ts-module), which is DASH compatible.

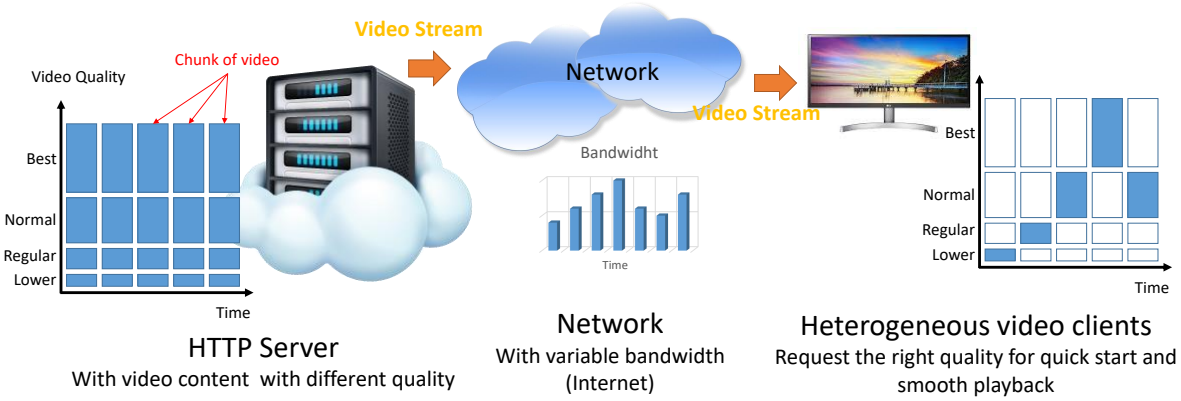

Figure 6.4: HAS Streaming Video – Source: adapted from Ravi and Shah [2014]

The client's video player used in the experiments is *Firefox* with a *Javascript* video player. We use Dash.js video player (https://github.com/Dash-Industry-Forum/dash.js/wiki) in our clients, because it is open source, and because it can send video performance metrics to the controller. Dash.js leverages the Media Source Extensions API defined by the World Wide Web Consortium in https://www.w3.org/TR/media-source/. This API is used to gather the video information, and then send it to the controller. In our experiments, the video file is split into several segments (chunks) of the same duration, and each video segment is encoded with multiple discrete bitrate levels as shown in Table 6.4. DASH.js uses one out of three ABR algorithms (Spiteri et al. [2019]):

(a) a strategy based on the recent throughput history. It estimates the throughput

in the next time frame, scales it down to 90%, and selects the highest bitrate supported by this scaled throughput;

(b) BOLA (Buffer Occupancy based Lyapunov Algorithm) (Spiteri et al. [2016]), which determines the bitrate from the buffer occupancy. As the buffer occupancy grows, higher-bitrate segments are preferred. However if the buffer occupancy drops, lower-bitrate segments are selected to download; and

(c) a heuristic that switches between BOLA, and the throughput strategy in real time.

We used the default strategy, which is the dynamic one, because Spiteri et al. [2019] indicate that this strategy presents the strengths of both strategies.

Table 6.4: Experiment's video parameters

Video parameters	**Value**
Codec	avc3.640032; hev1.1.6.L60.90; vp09.00.40.08; av1.experimental.84dc6e97cb6bfaee4185e63ac78dcd5e080f378c
Resolutions	256x144; 320x180; 384x216; 512x277;604x360; 768x432; 1024x576; 1280x720; 1920x1080; 2560x1440; 3840x2160
Frame rate	30 fps
Chunk size	2 s

6.3.1.1 Evaluation Scenario

The proposed scenario simulates a site with multiple APs managed by a single administrator, such as in a company, a campus, etc. It is similar to the scenario shown in Section 5.5.3.3 but instead of a web page, the stations download a video, as shown in Figure 6.3.

This scenario is important because it has great business potential as it affects locations with large numbers of users. The controller has to handle the cross interference of the whole wireless network, seeking a global reward that improves the average QoE and also improves the QoE provided by individual AP. Moreover, as we saw in the previous chapter, a configuration with multiple APs managed by a single administrator showed better convergence than in the fully distributed case, as well as it may benefit from the reduction of the required memory due to the use of an approximation function, as studied in this chapter.

6.3.2 Training data for the MOS predictor

This section shows how we obtained the training data for the video QoE metric. Traffic generation between stations, and AP is done using *Firefox*, which plays continuously a selected video, from a video server in the testbed. Each round takes 12 hours. Data was collected in a 15-day period, providing over 1 million samples.

An independent wired network forwards the data collected in the experiment, leaving the wireless network exclusively for the video traffic. Additional flow features were obtained in the AP with *iw* (https://wireless.wiki.kernel.org/en/users/documentation/iw). The stations, and the AP's clocks were synchronized using NTP. The data collected with each tool was combined according to the timestamp, and the MAC address. Data from *Firefox* and *iw* are collected using 1 s intervals. The dependent variable y (the MOS perceived by the user in the station) is collected using *Firefox*.

The AP can sense another 72 APs, which we do not have control. Also there are 802.15.4 nodes operating in the same frequency range. Thus, the data collected portrays variable and uncontrolled interference over time.

6.3.3 QoE prediction Evaluation

This section shows the final model obtained from training a video MOS metric using the features described in Section 6.2.2. The hyperparameters were selected using 3-fold cross-validation over a fraction of the training data. Having selected the best model, it was trained using the full training dataset, and then tested with the test dataset (10% of our data). The examples in the test dataset were never used to train the model.

We have evaluated three MOS metrics. One with features obtained only on the AP, one with features only from the client, and one with both types of features. However, only the second approach showed an RMSE value that we considered acceptable for our scenario ($RMSE = 0.728$). The best RMSE obtained for metrics that use AP-only features was 1.233, while for the hybrid metric it was 1.3409. Therefore the analysis in there rest of the thesis will consider only the metric using features obtained in the client (shown below).

MOS_CLIENT To generate a more general form of the equations in Yin et al. [2015b], we considered the format shown below. Notice that this equation reduces to the equation proposed by those authors when $a_0 = 0$, $a_1 = 1$, $a_2 = -\mu$, and

$a_3 = -1$. We tested using $N = \{2, 4, 8, 16, 32, 64\}$, which define the number of bitrate and rebuffering time samples used in the equation.

$$\text{MOS\_CLIENT} = a_0 + a_1 \times \sum_{n=1}^{N} q(R_n) + a_2 \times \sum_{n=1}^{N} T_n + a_3 \times \sum_{n=1}^{N-1} |q(R_{n+1}) - q(R_n)| \quad (6.7)$$

We chose not to use the authors' equation directly, as this equation requires that the value of μ provided in their proposal generalize the MOS results to our scenario. Checking the results obtained for μ using MOS_PSNR shows that the results are very different. Moreover, the authors do not present in the article the error produced by this equation (for example RMSE) using their μ. Therefore we decided to retrain the predictor. However for this, it is necessary to know the MOS value for each situation. Doing this process manually is costly, as it demands a human panel with enough members to obtain statistical significance, and to evaluate every second of the transmitted video. Instead, we used the offline estimator MOS_PSNR.

We trained the equations proposed by Yin et al. [2015b] using their log and linear formulation, and also using SVR as proposed by Mao et al. [2017]. Both epsilon-SVR and nu-SVR methods (Schölkopf et al. [2000]; Smola and Schölkopf [2004]) were tested, because the method was not specified in Mao et al. [2017]. However, these approaches provided RMSE bigger than one. Therefore, we tested new approaches using the same features. We consider a linear regression using GD and SVR using the following kernels: linear, polynomial, radial basis function (RBF), and Gaussian kernels.

The best result obtained in the search is the linear model using an RBF Nyström kernel (Williams and Seeger [2001]). In this model, the kernel is applied to the features, and the transformed features are used to train a linear model using gradient descent. This model obtained an RMSE equal to 0.728. Figure 6.5 shows a scatter plot where the X-axis shows the value predicted by the best model, and the Y-axis shows the reference value for the MOS (obtained using MOS_PSNR). Since we want the predicted value to be equal to the reference value, the ideal distribution for the points is a line, shown in red. Thus we see that the predictor shows a trend similar to the ideal distribution, but with a large variation around this line. Another important point is that the predictor in no case predicted a MOS equal to five. Furthermore, we see that for MOS equal to one, the prediction is quite varied, since the points are distributed almost as a continuous horizontal line for Y = 1.

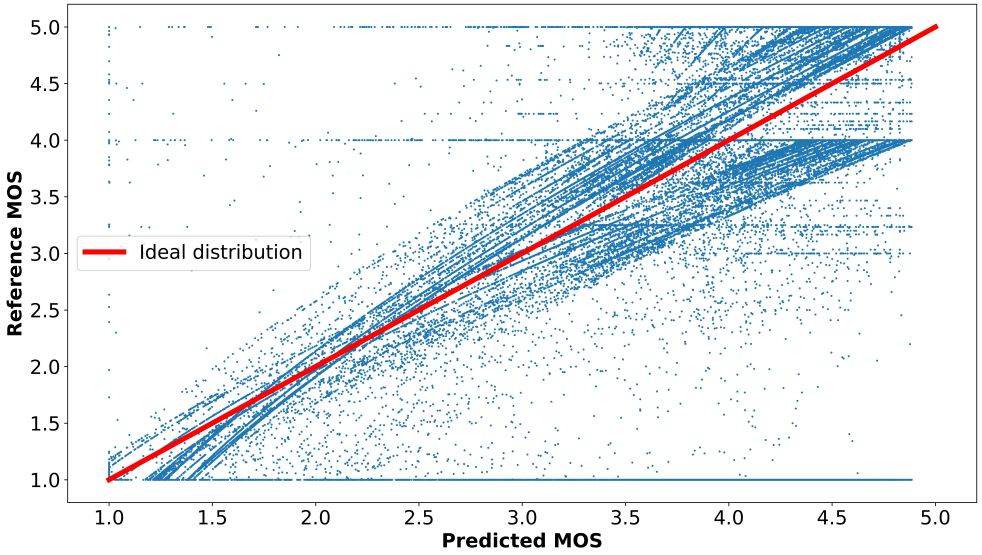

Figure 6.5: Distribution of expected points using MOS_CLIENT in relation to the reference value. $RMSE = 0.728$.

6.3.4 Control Loop Evaluation

This section shows the results of the experiments used to test the control loop. It uses four metrics: convergence time, fairness, MOS, and regret.

6.3.4.1 Experiment setup

We used the setup shown in Section 6.3.1. In each experiment, a process in the station continuously runs Mozilla Firefox version 69.0.2, which executes the video from a video server, in a continuous loop. Figure 5.4 shows the physical layout of our devices.

The experiments begin with the system having zero knowledge. We set the hyper-paramteres as shown in Table 6.3. The value of γ was selected to favor an immediate reward. We wanted the control loop to guarantee a high reward quickly, mainly because an improvement in network performance can be amplified as a result of DASH customer adaptability, which thus improves the immediate result for the client. Remember that the control loop runtime is 30 seconds, which is 25% of the video playtime, and DASH.js can switch immediately (even in the middle of a chunk) to a higher quality segment. Notice that this affects Q-network update (see lines 14 in Algorithm alg:DQL), but does not have an immediate effect over the Target-network.

In each run, the control loop is executed in a timely basis at thirty-seconds intervals. Therefore, state and reward information is aggregated with this granularity.

We have adopted a conservative value for the period between iterations: this period is double the maximum runtime of one iteration of the control loop. Table 6.5 shows the average, the confidence interval for the mean with 95% confidence, considering 120 one-hour runs, and the maximum value of one iteration of the control loop.

Table 6.5: Control loop runtime (in seconds)

Statistics	Mean	Lower bound	Upper bound	Max
Runtime (s)	4.697	4.578	4.816	16.0

We consider three baselines to evaluate the performance of the proposed controller:

1. *overlapping* channel baseline: In this baseline, the APs are configured on the same channel (channel 1, which contains the largest number of APs detected in the neighborhood) and the highest transmission power. Due to the physical configuration of our testbed, this ensures the highest level of co-interference generated by the stations.

2. *non-overlapping* channel baseline: In this baseline, the APs are configured in different and opposite channels (channels 1, and 11), and the highest transmission power. Because the channels do not overlap, one station's transmissions does not affect the other's. The full power configuration ensures that the station can achieve the highest possible bitrate, minus the interference of third party networks over which we have no control.

3. **ACS** baseline: This scenario is similar to the one described in Section 5.5.1.1. In each second the controller calculates the ACS based on Equation 5.6. The average of the interference factor is obtained at the end of the control loop runtime (see Section 6.3.4.1). The algorithm greedly selects the channel with the lowest average interference factor. The transmission power is set to the maximum value, and is maintained in this value during the entire experiment.

4. **ACS2**: This is a variation of the previous baseline. It applies to our setup, where there is only two APs. The controller computes the ACS, but if the greedy move selects the same channel, the controller also tests if the second best channel has an idle time greater than half the time of the first channel. If it is greater, the controller selects the second best channel with the largest margin, keeping the other AP in the first channel.

6.3.4.2 Results

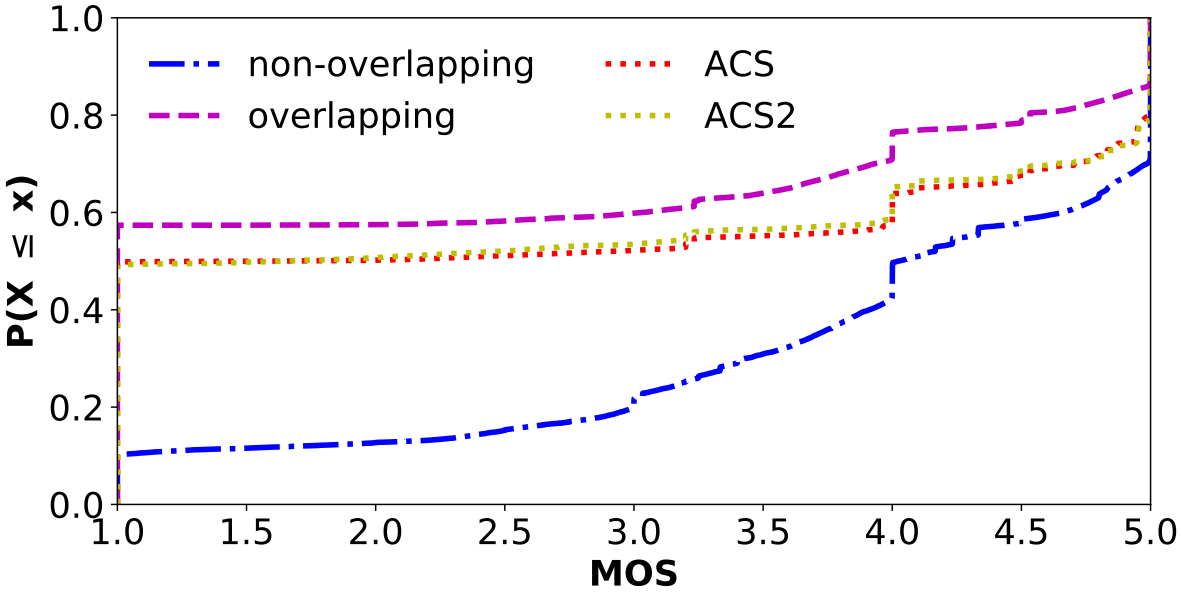

Figure 6.6: Video QoE – CDF of both baselines using MOS_PSNR

We can use PSNR to calculate the value of a MOS. This is done offline comparing the reference video, and the received video at the client, as shown in Section 6.2.2. Thus, Figure 6.6 shows the CDF of the MOS values obtained for those configurations. The figure shows the results obtained using *ACS*, and *ACS2*. The latter checks if the performance improves compared to the *ACS* baseline, due to reducing the conflict in channel selection. However, Figure 6.6 shows similar results for both. Therefore, in the rest of this chapter, we only show the *ACS* baseline.

Figure 6.7 shows the results obtained using the control loop, and the MOS_CLIENT predictor for the QoE metric. Figure 6.7 shows the CDF considering the baselines, and the control loop. We observe that the control loop achieves better results than the non-overlapping and the overlapping baselines. Notice that the predictor indicates worse results for the *non-overlapping* case when compared to the *overlapping* case. This is due to the variance presented in the MOS predictions made by the MOS_CLIENT predictor. Also, neither the baselines nor the control loop can reach the highest MOS, since the predictor never outputs $MOS > 4$.

Due to the high prediction errors using MOS_CLIENT, we also present the results using MOS_PSNR in Figure 6.8. Thus, the videos were processed after the execution to obtain MOS_PSNR values for the runs. The *non-overlapping* baseline obtains better results than the *overlapping* baseline, and yet the control loop gives better results than both. It is important to highlight that this is true even with

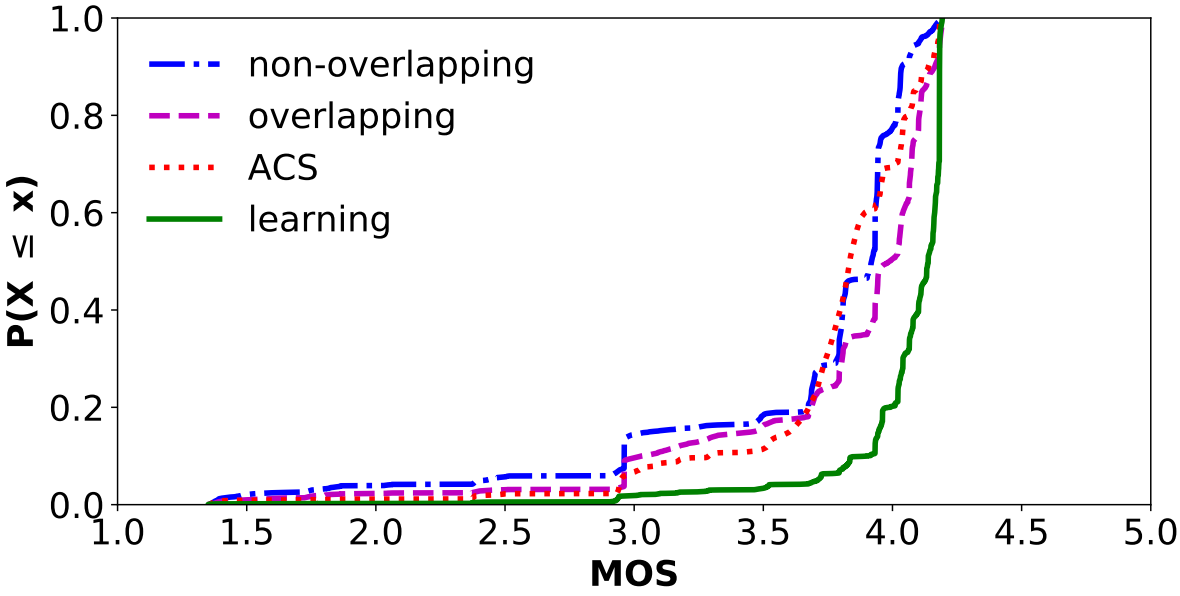

Figure 6.7: Video QoE – Comparison between the results the control loop and the baselines using the MOS_CLIENT predictor

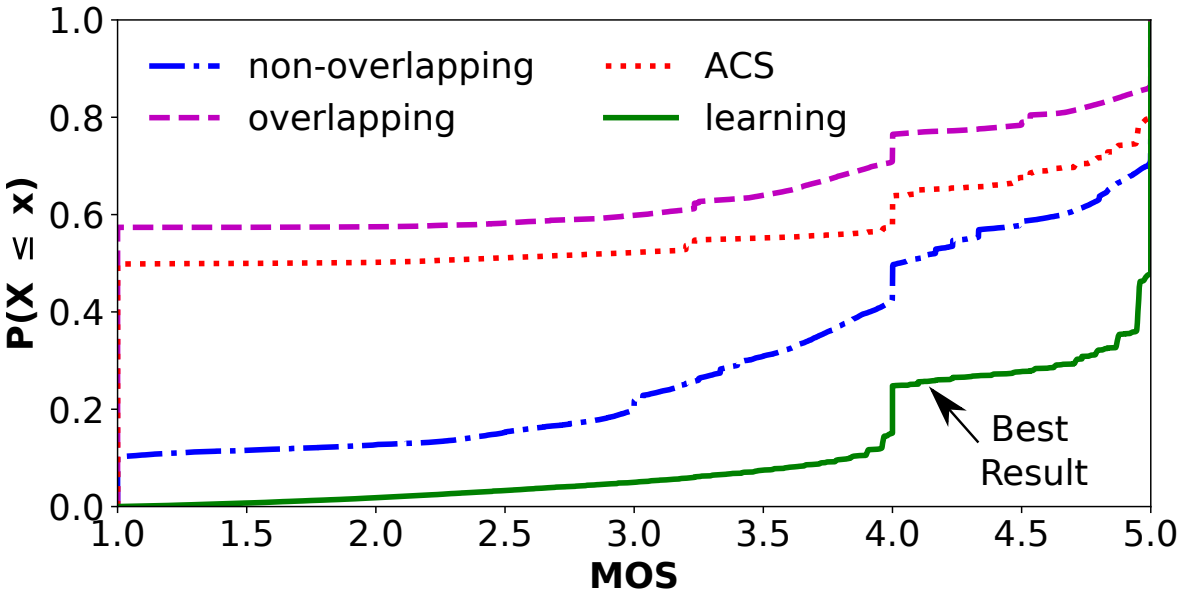

Figure 6.8: Video QoE – Comparison between the results using MOS_PSNR

the prediction errors, because, despite them, there is still a good correlation between the prediction value, and the actual MOS_PSNR value. We observed that about 50% of the control loop results reach the maximum MOS, while the *non-overlapping*, and the *overlapping* baselines only reach this value for 30% and 10% of their results, respectively.

Table 6.6 shows the regret obtained during the whole experiments using the control loop, and the baselines. Note that these results contain all intervals, including the

Table 6.6: Comparison of the regret obtained using the control loop, and the baselines.

Stats	Non-ovelapping	Overlapping	ACS	Learning
Mean	1.167 ± 0.024	2.602 ± 0.019	2.286 ± 0.029	0.414 ± 0.004
Std.Dev.	1.287	1.692	1.815	0.733
Improvement	64.5%	84.1%	81.9%	—

Note: 1) 95% confidence interval of the mean.

initial learning phase, because the control loop reaches the maximum MOS in about 3 intervals. Thus we also did not evaluate the performance during convergence, as we did in Section 5.5.3.1. The table also shows the confidence interval of the mean with 95% confidence.

We observe that the control loop has the lowest value for regret, meaning that it achieves the best overall result. The control loop improves the regret by 64%, 81%, and 84% if compared to the *overlapping*, *ACS*, and *non-overlapping* baselines, respectively. The results for *overlapping*, and *non-overlapping* baselines are expected, i.e., we expected non-overlapping to have lower regret than overlapping, because in the former, the APs are on non-interfering channels. However, we were surprised by the result obtained with the *ACS*, and *ACS2* baselines. We expected *ACS* to achieve a result at least similar to *non-overlapping*, but we notice that the results are close to the *overlapping* baseline. Analyzing the algorithm behavior, we expected that in many cases the result from the interference factor formula (see Equation 5.6) for the APs in our scenario may cause the algorithm to greedily select the same channel for both APs. However we noticed no significant difference between *ACS*, and *ACS2*.

Table 6.7: Comparison of the fairness index obtained during the experiment.

Stats	Non-overlapping	Overlapping	ACS	learning
Mean[1]	0.667 ± 0.008	0.518 ± 0.006	0.461 ± 0.010	**0.994** ± 0.001
Std.Dev.	0.296	0.411	0.421	0.055
Minimum	0.000	0.000	0.000	0.001
Maximum	1.000	1.000	1.000	1.000
Improvement	49.0%	91.9%	115.6%	—

Note: 1) 95% confidence interval of the mean.

Table 6.7 shows the Hossfeld fairness index values for the baselines, and the proposed control loop. The row with the mean shows the confidence interval with 95% confidence. For the mean values, the *overlapping* baseline obtained the worst value, while the best value is obtained with the control loop. The control loop improves the

fairness index by 49%, 91%, and 115% when compared, respectively, to the baselines *overlapping*, *non-overlapping*, and *ACS*.

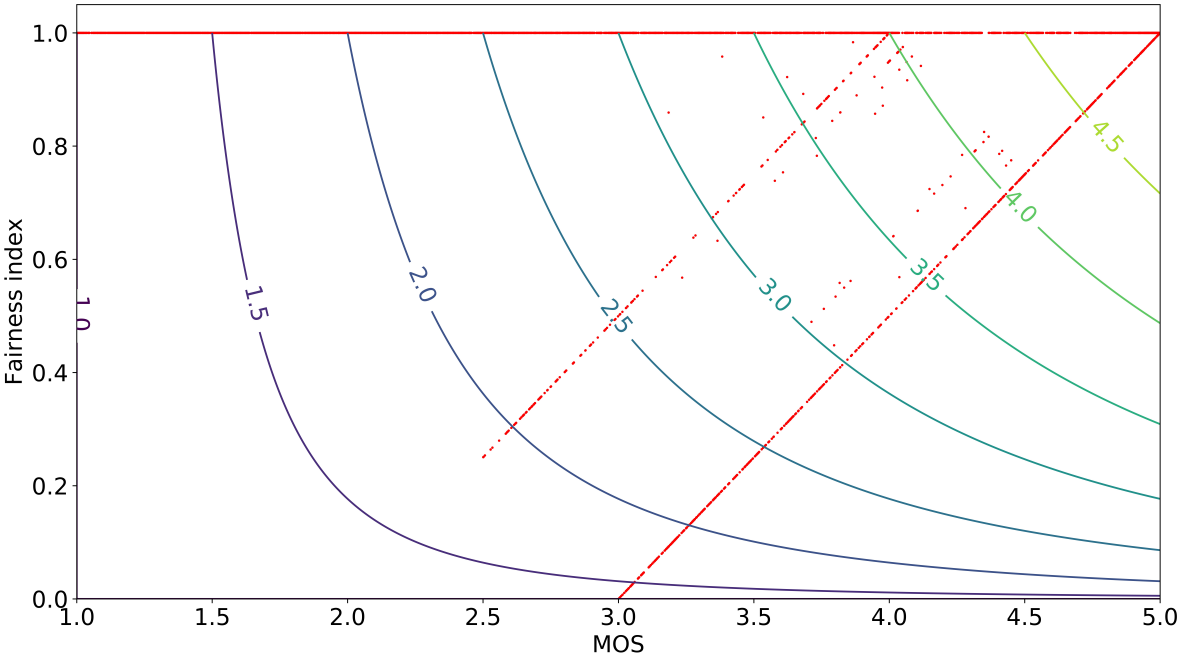

Figure 6.9: The results of fairness and MOS obtained during the learning experiment plotted over the reward function. The dots in red show the pair of MOS, and fairness obtained in the experiment. The number in the curves show the reward.

Figure 6.9 shows the reward of the control loop for each execution of the algorithm (dots in red) drawn over the reward function plot. The values are concentrated in a region at the top of the graph, where the fairness index is maximum. We also observed two patterns (they look like lines in red) that follow the improvement paths of both the MOS and the fairness index. This improvement follows approximately the gradient of the reward function.

Table 6.8: Time (in seconds) taken by the system to converge to the maximum MOS

Stats	Non-overlapping	Overlapping	ACS	learning
Number of Runs	92.0	120.0	120.0	120.0
Mean	185.5 ± 28.6	252.8 ± 28.1	167.3 ± 14.0	111.0 ± 13.0
Std.Dev.	137.9	209.5	122.2	73.9
Maximum	810.0	1,260.0	930.0	420.0
Improvement	40.5%	56.3%	34.0%	—

Figure 6.10 shows a histogram for the time to converge to the maximum MOS value grouped in bins of four execution periods of the control loop (or 120 seconds).

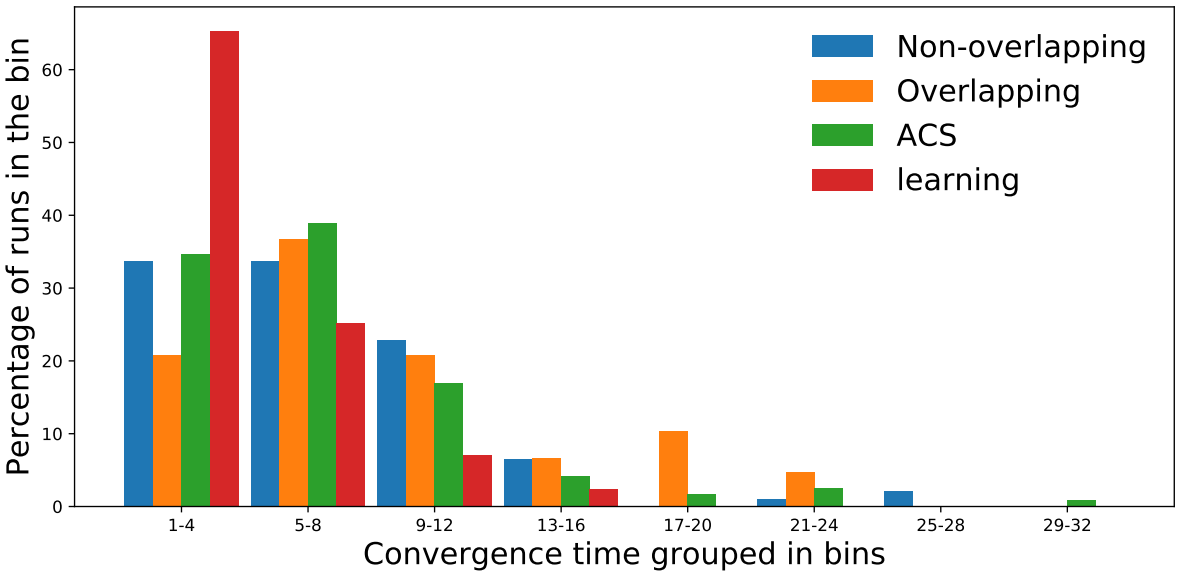

Figure 6.10: Video QoE – Histogram with the time to converge to the maximum MOS grouped in bins of 4 timesteps. The Y-axis shows the percentage of runs that appear in the bin. *Note*: The figure only shows bins up to 32 because the rest is negligible.

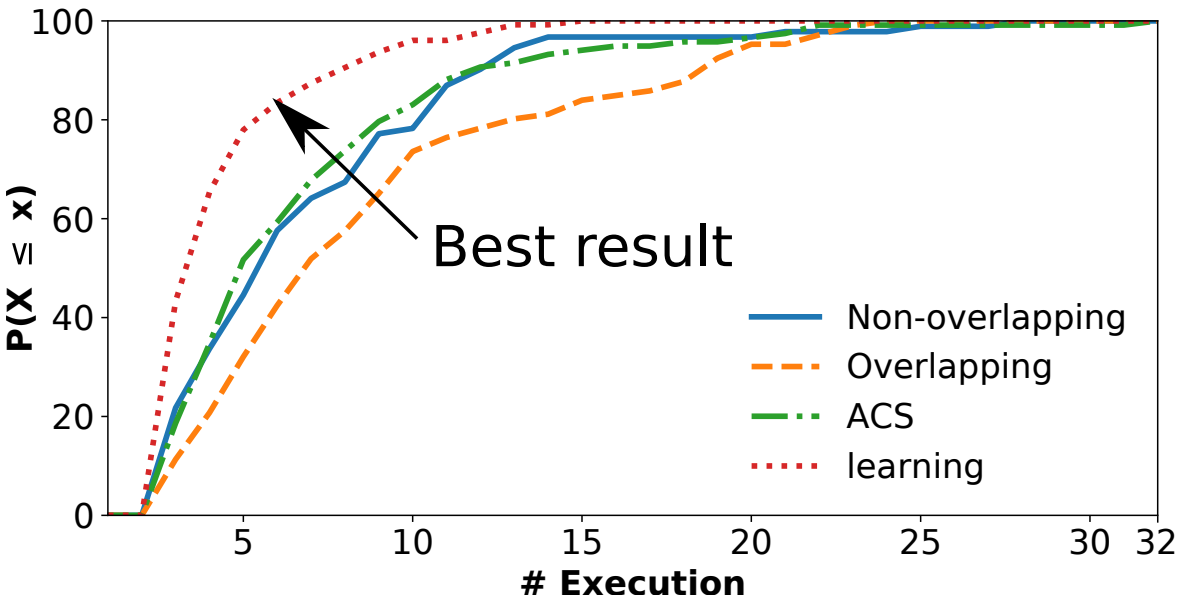

Figure 6.11: Video QoE – CDF with the time to converge to the maximum MOS in bins. The Y-axis shows the percentage of runs that appear in the bin. The X-axis shows the timesteps.

The X-axis shows up to the last bin that contains a run, while the Y-axis shows the percentage of runs the system has converged during each of the bins. We can see from the figure that *learning* gets its maximum peak in the first bin, unlike the baselines that are distributed in the other bins. From our data, the maximum values observed

for each case — *non-overlapping*, *overlapping*, *ACS*, and *learning* — were, respectively, 810, 1,260, 930, and 570 seconds. These results reinforce the fact that the control loop can get better average MOS than the others during all the experiment. We observe in our data that using the control loop, whenever the MOS falls below 5, the algorithm can return to the maximum MOS situation on average in 104.2 ± 2.7 seconds. That is, the algorithm can recover to full MOS in up to four interactions (on average) after the disturbance.

Figure 6.11 shows CDF of the convergence time for all runs of the control loop, and the baselines. The X-axis shows the timesteps required for convergence (120 timesteps equals one-hour run, which is the test maximum execution time). The Y-axis shows the probability of the method to converge in the timestep indicated in X. Thus, the further the curve is to the left, the better the results, *i.e.*, the faster the method converges. Then, the control loop shows better results than the others, converging in 50% of cases at least twice as fast.

6.3.5 Interpreting the Q-function

This section evaluates how the Q-function features contribute to the action selection taken by the agent. The main objective of this section is to shed light on the system behavior, seeking to better understand how the "black box" of the learned model generates decisions. This is an important topic in machine learning, since most complex models do not explain their predictions (Molnar et al. [2019]), and predictions are hard to interpret by humans. In this thesis, we consider that interpretability is the degree to which a human can understand the cause of a decision (Miller [2018]).

Section 6.3.5.1 will analyze two aspects related to the model: (1) identify which features the model considers most important; and (2) for a given prediction provided by the model, how each feature affected that specific prediction. We also built a surrogate model (Section 6.3.5.2), which is an approximation of the TCNN model, and it was used to help simplify the features used in the *learning* model's states.

6.3.5.1 Feature importance

This section quantifies the impact of features on the DDQL predictions. In the literature, this concept is called feature importance (Molnar et al. [2019]). This work employs permutation importance and partial dependence plots.

Permutation importance is calculated after the model is trained. Therefore it does not affect the model or change the predictions obtained to a certain value. The method randomly shuffles a single column of the validation data, leaving unchanged the target,

and all the other columns (Fisher et al. [2018]). Thus it measures the effect in an error measurement (typically the accuracy in classification, and mean square error (MSE) in regressions) in the shuffled data. The rationale here is that an important feature will generate a big change in the error, while less important ones will make a minor change (or even improve the result, when random changes cause the predictions on shuffled data to be more accurate). There is some randomness to the exact performance change from shuffling a column, which can be measured in permutation importance by repeating the process with multiple shuffles. Thus the method provides two results: one that measures the scale of the effect of the shuffle in the result, and an estimate of the error of this scale.

The libraries used to compute permutation importance are: eli5 (https://eli5.readthedocs.io/en/latest/index.html) version 0.9.0, mlxtend version 0.17.0, scipy version 1.3.1 (https://www.scipy.org/scipylib/index.html) and scikit-learn version 0.19.2 (https://scikit-learn.org/stable/documentation.html).

Figure 6.12 shows the results of permutation. The variables that have greater impact on the predictions are *tx_byte_avg_t1*, *tx_byte_avg_t0*, *rx_byte_avg_t1*, and *rx_byte_avg_t0*. This is an expected result: when the transmission capacity is higher, the DASH.js client is able to request higher quality chunks, which are therefore larger in size. Another important point observed in this result is that it confirms the need to use a temporal approach, as both the time value t_0 and the value in t_1 have a large influence on the prediction.

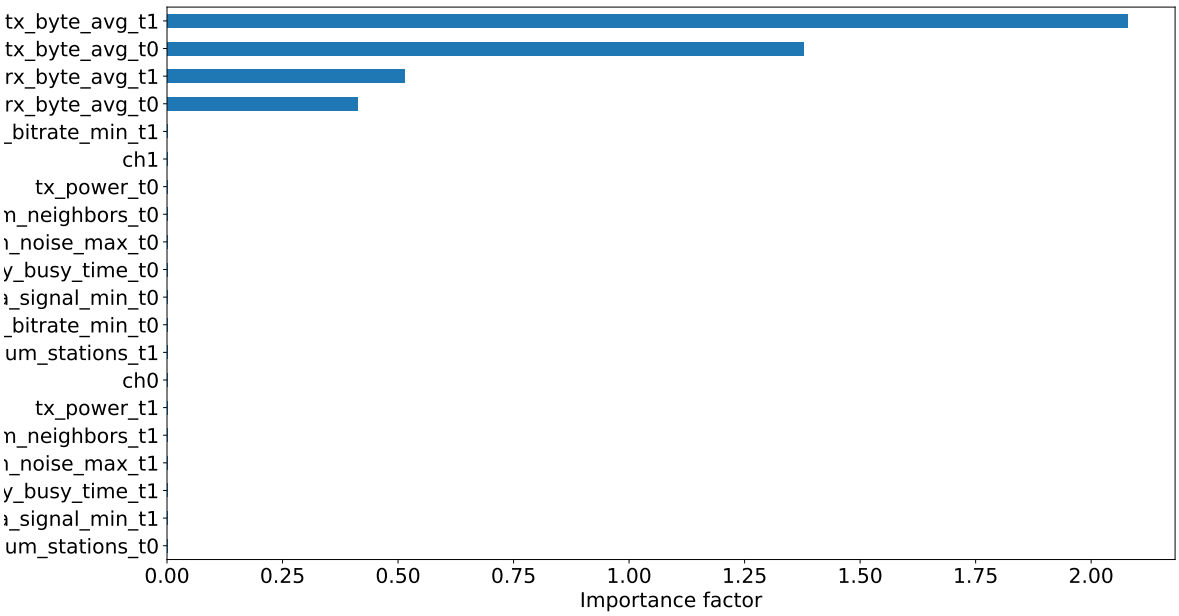

Figure 6.12: Feature importance permutation using mlxtend with correlation coefficient.

Some features used in the control loop should not influence the results: *num_stations*, and *num_neighbors*. Due to the evaluated scenario, the value of *num_stations* is fixed, as each AP has only one station connected. Also the number of neighboring APs (managed by the controller) relative to a specific AP is also fixed, *i.e.*, there are only two APs, and each AP only detects the other one. This number may, however, vary slightly (we have identified a standard deviation of 0.01 in our data), because during initialization one AP can start faster than the other, and thus identify that there are no controlled AP in the neighborhood.

We also used partial dependence plots (Friedman [2001]) to see how a feature affects the model's predictions. Partial dependence plots use the trained model to predict outcomes by repeatedly altering the value for one feature (keeping the others unchanged). Thus it shows the marginal effect of one or two features on the predicted outcome of our model (Molnar et al. [2019]). The technique is usually applied to only one feature or a maximum of two features, because one feature produces 2D plots, and two features produce 3D plots. Everything beyond that is quite tricky (Friedman [2001]). The library that was used to generate the plots is pdpbox (https://github.com/SauceCat/PDPbox) version 0.2.0.

Partial dependence shows how the variation on a selected feature changes the prediction from what it would be predicted at the baseline. The analysis also shows the level of confidence in the predicted value. This analysis produces one plot per feature. The feature affects the prediction if there is a visible variation when the feature value changes between its maximum and its minimum values. According to the method, there are some features that do not affected the prediction, as shown in Table 6.9. The main features are tx_byte_avg_t0, tx_byte_avg_t1, rx_byte_avg_t0, and rx_byte_avg_t1, which agrees with the result obtained using feature importance permutation. However this model The features *perc_phy_busy_time_t0*, and *sta_signal_min_t1* significantly influence forecasting only at lower baud rates $tx\_byte\_avg\_t_1 \leq 279\ kbps$.

It is interesting to note that two different forms of analysis highlight a common and small set of features that affect the predictions. Because of this result, in a future study we aim to simplify state representation to see if we can achieve similar results at a smaller computational and memory cost.

Table 6.9: Variation observed in the prediction caused by the feature variation

Feature	**Variation**
tx_byte_avg_t1	35.57
tx_byte_avg_t0	20.90
rx_byte_avg_t1	1.97
rx_byte_avg_t0	1.58
ch_noise_max_t0	0.85
rec_bitrate_min_t1	0.60
ch0	0.51
perc_phy_busy_time_t1	0.46
perc_phy_busy_time_t0	0.41
sta_signal_min_t0	0.36
rec_bitrate_min_t0	0.10
sta_signal_min_t1	0.06
ch_noise_max_t1	0.03

Note: Other features not shown did not affect the prediction.

6.3.5.2 Surrogate model

A surrogate model is used when one cannot directly understand how the real model works. A surrogate is a result model, i.e. it is an interpretable model trained to approximate the predictions of a black box model Friedman [2001]. Since it is simpler, the surrogate model should be easier to analyze.

The surrogate for our DQL state-value function was trained using a decision tree, and is shown in Figure 6.13. This model shows a correlation coefficient of 73%, when applied to the test set. It is not a very high correlation, but it is a price to pay for a simpler model, and as highlighted by Molnar et al. [2019], it is not clear what is the best cut-off correlation coefficient in order to be confident that the surrogate model is close enough to the black box model. However, we must point out that this model agrees with the methods shown in the previous section with regards to the variables that influence the control loop decision the most.

Notice that the whole branch when the condition in root node ($tx\_byte\_avg\_t1 \leq 279536.0$) is False, only depends on features based on the transmission, and reception of bytes by the APs. Thus we can say that when an AP reaches a certain flow threshold (in our case, 279 *kbps*), the control loop is only concerned with the amount of data carried in the wireless link. When the traffic is below the threshold, other features that appeared in the partial dependency analysis also influenced the results, for example, *perc_phy_busy_time_t0*. This shows that the control loop's decision model can be much simpler, focusing mainly on bandwidth

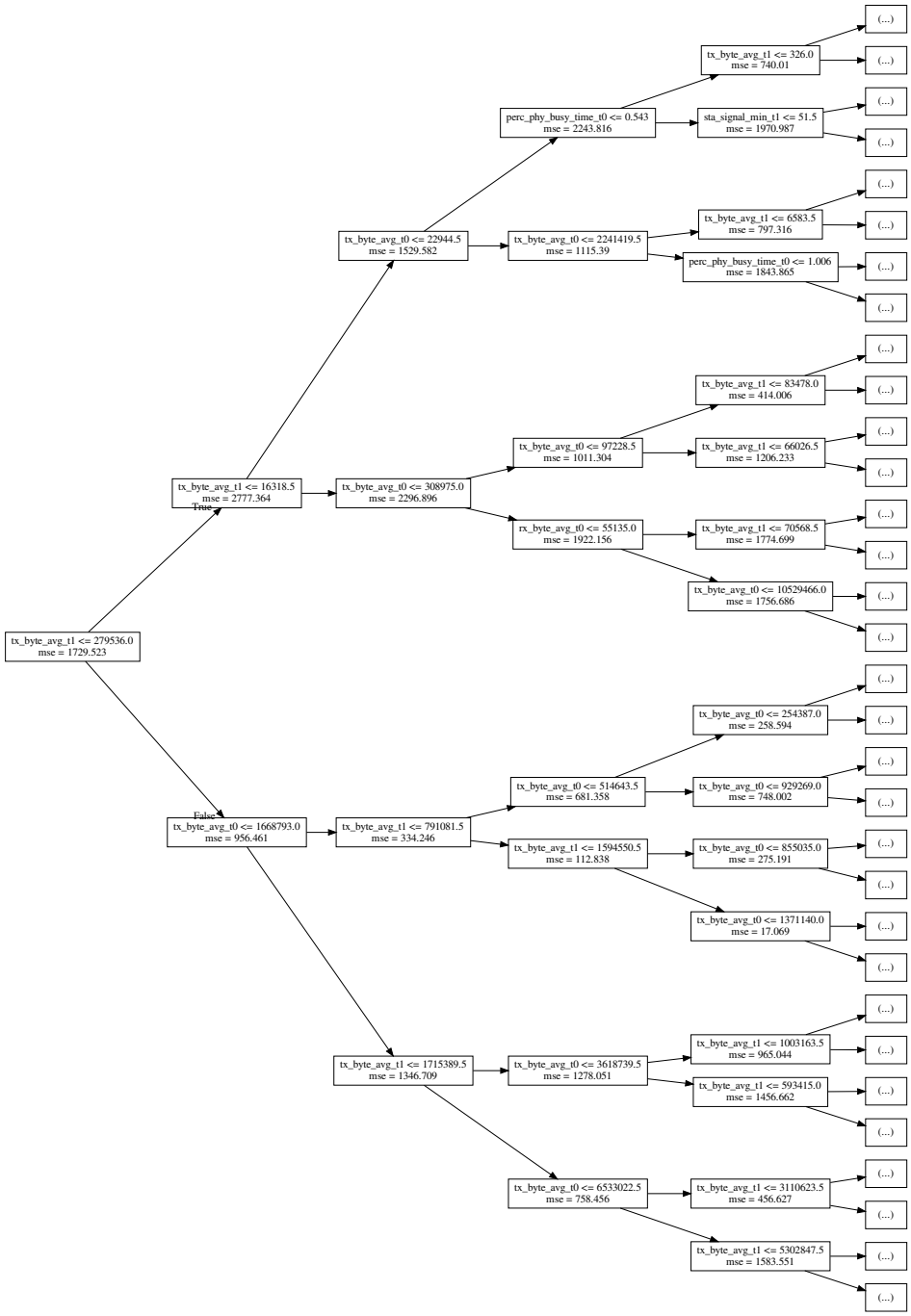

Figure 6.13: Surrogate model. Tree depth truncated to 4 levels.

usage, and a few network parameters such as busy time, signal, and noise. Section 6.3.6 shows this simplified model.

6.3.6 Simplifying the state representation

The results obtained with the full state representation (see Section 6.2.3) suggest that this representation could be simplified. In this section we show this simplified representation and discuss the results obtained.

State representation To verify if a simplified model would achieve similar performance compared to *learning*, we created a new prototype with the following features, which we called *simple*:

$$s_t^{(i)} = (perc\_phy\_busy\_time, sta\_signal\_min, rec\_bitrate\_min, \\ tx\_byte\_avg, rx\_byte\_avg)$$

In *simple*, the state is also represented by a tuple (s_{t-1}, s_t), which contains the current state and the previous state, where the state entry in the period t is the tuple above. Note that we use the two features that evaluate traffic — *tx_byte_avg_t0*, *tx_byte_avg_t1*, *rx_byte_avg_t0* and *rx_byte_avg_t1*. We consider beyond these four features only those that were identified in the partial dependence analysis, and in the surrogate model. Thus the final state representation in *simple* contains 10 features, while the complete representation in *learning* contains 40 features.

Runtime This feature reduction should impact in the DQL training, and evaluation time, because the network becomes smaller. Further, it should impact the information collection time, because fewer REST calls will be made by the controller to the APs. Table 6.10 shows the average runtime, and the confidence interval for the mean with 95% confidence for both control loops. *Simple* was implemented by creating an environment class derived from the class used in *learning*. The class accesses the APs, and also defines the state, action representation, and the calculates the reward. The class in *simple* only changes the state representation, discarding features that would not be used. Thus the prototype continues to request the same features from the APs. The table shows that the total execution time has increased in *simple* by 22 ms. This increment is due to the calls in the hierarchy of classes, and time needed to drop the unused features from the data structure that holds the current state. No reduction in the execution time of the neural network due to the decrease of the number of features was noticed.

Table 6.10: Control loop runtime (in seconds)

Control loop	Mean	Lower Bound	Upper Bound
simple	4.719	4.601	4.837
learning	4.697	4.578	4.816

MOS Figure 6.14 shows the CDF of the MOS results obtained during the whole execution of both control loops. It shows that *simple* slightly improves the results obtained with *learning* by 0,7% on average. This is a small amount, but, by using the Student's t-test for the mean, with the same variance, one can tell that the MOS obtained for both control loops are statistically different with 95% confidence. This improvement may be due to an improvement in the quality of decisions made by the agent, *i.e.*, the features used in *learning* generate noise that impairs the controller's performance.

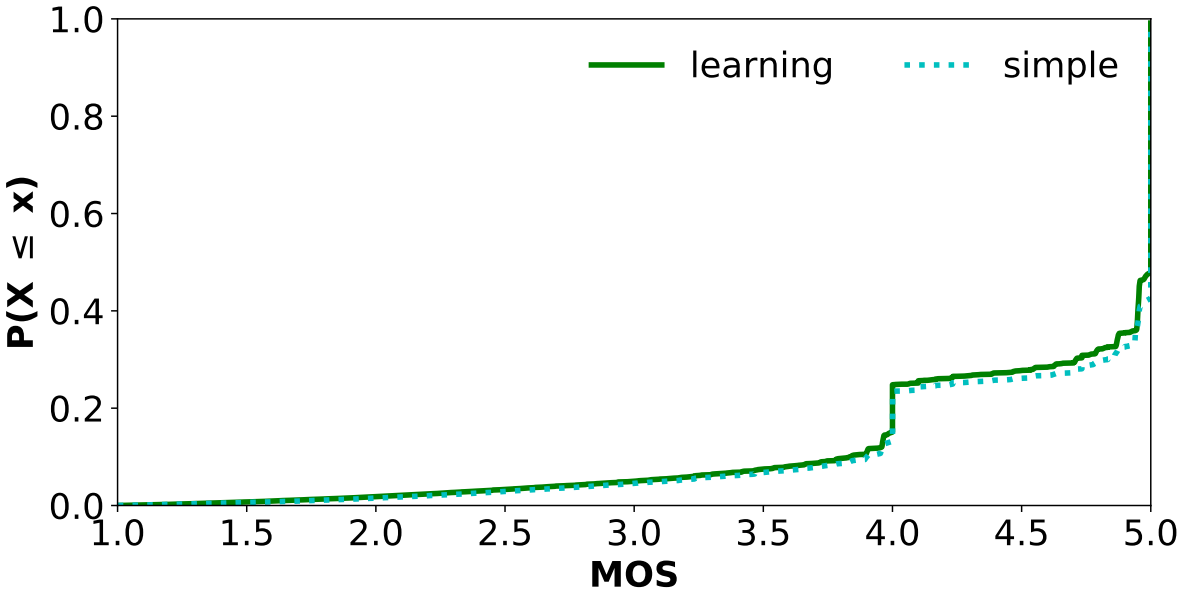

Figure 6.14: Video QoE – Comparison between the control loop results using MOS_PSNR

The results obtained for the fairness are statistically equal with 95% confidence.

Convergence Figure 6.15 shows the CDFs comparing both control loops against time to converge to maximum MOS at each execution step (30 seconds). In the first steps of implementation we observed that *learning* performs a little better, but, in the figure, this distinction is not clear, except in the range where the probability is between 70% and 85%. From this point on, *simple* gets a little better. Using the test of means, we

found that the average convergence times are statistically equal with 95% confidence for both control loops.

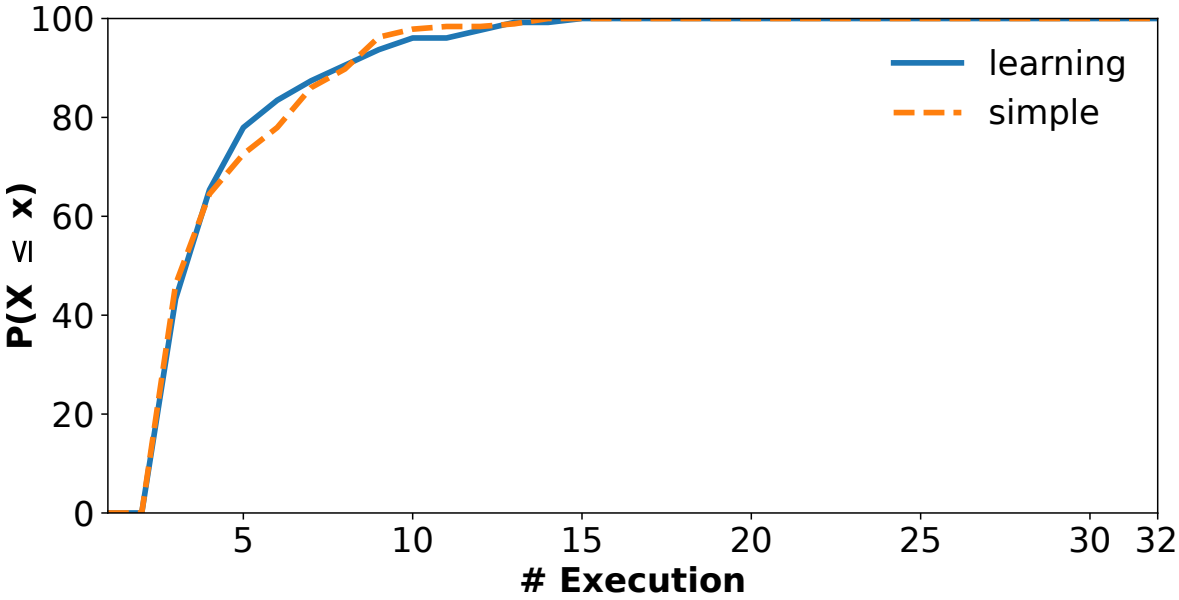

Figure 6.15: Video QoE – CDF with the time to converge to the maximum MOS in bins comparing both control loops. The Y-axis shows the percentage of runs that appear in the bin. The X-axis shows.

6.3.7 Discussion

Experiments show that for the scenario developed in this chapter, the control loop is able to improve not only what customers perceive, but also improve the fairness index, making both stations perform more evenly. The MOS metric used does not distinguish competing video flows from the same station, however we believe that a more complex scenario, that is, with more stations connected to the APs, should be explored in future work.

The results indicated that several features used to represent the state in the more complete control loop did not influence the predicted result. This may be due in part to the simple scenario used to evaluate the video transmissions. According to these results, the surrogate model relies mainly on four features: *tx_byte_avg*, *rx_byte_avg*, *perc_phy_busy_time*, and *sta_signal_min*, agreeing with the feature importance analysis. We used this information to simplify the state representation, and obtained similar results. However, based on the results shown in this chapter, one cannot say that a more complex setup would not need more features to maintain good performance. Thus there is still one follow up: modify the experiment setup to

make it more complex (for example, more concurrent stations). That would check if the simplicity of the evaluation scenario was the cause for the overly simplistic model. This study is left for future work.

6.4 Summary

This chapter details the proposed intelligent control loop for video applications using QoE. This control loop is evaluated in a central control scenario where the APs are controlled by a single controller. We compare the performance of this system when the QoE uses video metrics obtained from the client. Table 6.11 shows the summary of the performance metrics comparing the control loop, and the baselines. The control loop improves the regret up to 84% when compared to the baselines, respectively. The control loop also improves the fairness index up to 115% when compared to the baselines. Further, the control loop also reduces the MOS when compared to the three baselines.

Table 6.11: Summary of performance metrics improvement obtained by the **learning** control loop.

Method	**Improvement**		
	Regret	**Fairness**	**Convergence time**
Non-overlapping	64.5%	49.0%	40.5%
Overlapping	84.1%	91.9%	56.3%
ACS	81.9%	115.6%	34.0%

The use of interpretability methods allowed, in our case, to evaluate that the representation of states could be simplified, using much less features than the initial proposal in 6.2.3. Our results show that the performance of the system with fewer features (*simple*) is similar to that obtained with the initial proposal (*learning*). However, our studies do not allow us to evaluate whether this result occurs because the problem is simple, thus it can be modeled with few features, or because the scenario is simple, not requiring more information to deal with the stations' demands.

The next chapter presents the conclusions of the thesis. It highlights the main lines of future work opened with this thesis, lists the papers already published during the PhD, as well as work submitted, and under review.

Chapter 7

Conclusion and Future Work

The dynamics of data transmission and the scope of wireless network present several challenges not found in wired networks. Automatic management mechanisms are needed to rapidly adapt the network parameters to the traffic demands in order to improve the client satisfaction.

This thesis presented control mechanisms for wireless networks, specifically for WLANs using SDN, that allow the automatic configuration of these networks according to the demands of their users, aiming to improve their overall satisfaction. Mechanisms to improve two basic applications on the Internet, web browsing and video, were proposed. We evaluated machine learning techniques that are capable of online learning. This thesis focuses on model-free methods, thus reinforcement learning approaches were applied to discover the best actions to configure the wireless network.

Experiments using Q-learning and UCB1 were performed in an Internet browsing environment, and we use DQL in a video streaming scenario. The results show that the use of learning methods provides a lower regret than the baselines. Remember that the regret measures how much the agent is losing by not following the optimal policy (which, in our case, provides the maximum MOS in all situations). We have seen from the results for web traffic control shown in Table 5.7 and video streaming control in Table 6.11 that applying RL methods in the control loop generates an improvement in user satisfaction, represented by reduced regret. It should be noted that using the reward, which considers QoE, represents a guide to the proposed methods. It makes the control loop to take into account the user satisfaction, thus the agent seeks to maximize user satisfaction. Therefore, identifying a good QoE predictor is important for a good outcome. For example, in the case of web browsing, not only should the predictor be able to indicate the value of QoE, but the system must be able to correctly (and rapidly) classify the web site being requested by the user.

To deploy the web control loop in more realistic settings, we proposed a classifier to identify the site class of the web traffic according to the examples of the classes described by Hora et al. [2016]. This classifier uses offline training of a semi-supervised machine learning algorithm. The classifier was evaluated, and showed an accuracy of about 87%.

Experiments using DQL were performed in a video streaming scenario. This second control loop maximizes a reward function based on the QoE of video flows. Further, the control loop improves the regret by up to 84% in the best case. It also improves, in the best case, the fairness index by up to 115%.

There are two types of future work related to the web browsing experiment, namely improving the practicality of the system, and improving the machine learning algorithms. First we will advance the system to operate on flows other than web browsing. Regarding improvements in the method, there are many options to be pursued, for example, (a) using function approximation (also covered in the next chapter), (b) using a pre-trained system to respond quicker to customer requests and test the model generalization, (c) adjusting the exploration phase on-line, (d) using some black box function optimization, such as Bayesian Optimization (Brochu et al. [2010]; Snoek et al. [2012]), (e) using an actor-critic approach to improve the controller's decisions, and (f) evaluate if the knowledge can be transfered, *e.g.*, deploying the controller with pre-trained weights on a different testbed.

The results show that a control loop can improve the user satisfaction, *i.e.*, the architecture proposed in Section 4 can be used to improve QoE for wireless network users. We also see that instances of this architecture can be built using various southbound APIs. It also appears that the reward function can be used as a plug-and-play black box in the control system. Thus, an improvement in the reward function, for example by using a QoE metric that better represents the user's perception, can be easily integrated into the system.

While this thesis was written, we had faced some challenges, which we believe are common in the design of RL-based systems, and which still constitute an important research topic. The literature lacks a guideline to help define the states, and the reward function for wireless network problems using RL. However, we believe that an important point was addressed in Chapter 6, which is the refinement of the system state representation, which can make the control loop make better (or faster) decisions. In Chapter 5, UCB1 uses a simpler representation, and shows better results than Q-learning. Q-learning uses a matrix representation for the Q-values, and since the system explores a very limited amount of the state and action space during the experiment, this matrix becomes very sparse. This sparsity undermines the system's decisions, because

minor changes in the state or action places the system in a unknown situation (the position of the matrix was never updated). In Chapter 6, an approximation function is adopted for the Q-value. It addresses the sparsity problem, which is natural due to the exploration-exploitation dilemma in a production system, and given the amount of features that influence the state of the system. Chapter 6 shows a surrogate model for this function, indicating that the decision process is apparently simple. This impacts the quality of the decision positively, because the controller needs less data to reach a good policy. However, our results do not show whether the state representation should be simple because the problem is simple or because our evaluation scenario was too simple. To answer this question, we would need to use more complex scenarios, which was not done in this thesis.

A technical problem in the control loop's implementation stems from the lack of a good framework capable of providing communication with real devices. This led to the proposition and implementation of our own framework, which is capable of interacting with Linux-based IEEE 802.11 APs. However, our proposal must still be improved to allow other device types, and also to allow the framework to run on other platforms.

7.1 Future work

We have identified several research opportunities derived from the work done in this thesis. These are points related to the complexity of the scenarios, the improvement of wireless network control, and aspects related to the application of other machine learning techniques.

Complexity of the scenarios: The scenarios presented in this thesis, despite testing various network configurations, use a small number of devices. Thus, conditions of high competition between the stations connected to the controlled APs were not evaluated. Using more stations would evaluate how control performance is affected in terms of the processing time of a larger volume of data. Longer processing times can even affect control loop decision periods, influencing decision quality. In addition, more concurrently competing stations may affect how easy it is for the controller to allocate the APs to channels.

Reward function: The control loop depends on the reward function to set the system goal. This thesis uses MOS as the basis for the reward, however as we observed in experiments with web browsing and video streaming, the use of this metric is not direct. For example, we have seen that averaging the MOS can lead to an unbalanced

situation where one agent receives a very high reward and the other a very low reward. We experiment with a penalty on the average reward to encourage agents to achieve a more balanced situation. However, this penalty adds an additional hyperparameter.

We believe that a method to automatically derive the reward constitutes a new line of research. One of the promises of ML is that data can be used to learn things better than human design. Thus, since reward function design is difficult, why not use data to learn better reward functions? Imitation learning and reverse reinforcement learning could be used for better reward functions, which would be implicitly defined by human demonstrations or human classifications.

Concept Drift: Chapter 4 shows our proposal for the control layer operation using RL. In this proposal, we highlight the need to detect concept drift. The occurrence of a change may make the value learned in Q-function no longer appropriate for the new environment configuration. However, concept drift may occur in different ways, so different detection, and handling schemes may be required for each situation. Recurring changes and long-term trends are considered systematic, and can be explicitly identified and manipulated. When the drift is detected, the system may decide what to do. It can restart training by prioritize exploration. The system may retrain the current Q-function, or train an ensemble (a new Q-function is created with differences that occur). If the system contains several pre-trained models, the system can select one that best handles the new concept. Evaluating approaches that can meet online learning requirements may be pursued in future work.

Use planning as a fine tuning step: Dyna-Q (Sutton [1991]) is a classical example of this approach, and AlphaGo is one of most successful cases. Our experiments required simple models, however, the scenarios were also simple. We believe that more complex scenarios (more devices, more distinct flows, and requirements, mobile users, etc) are subject to variations requiring greater adaptability of the agent. The literature shows that planning, defines by Sutton and Barto [2018] as any computational process that takes a model as input and produces or improves a policy, can improve the system response. This model can be trained online, thus requiring no previous knowledge of the system, which is one of our premises. This approach should be addressed in a future work.

If enough data is available, a supervised method can be used to create the system's model. Although this approach sounds simple, in a dynamic system, the model's update is a concern. A static (or older) model can negatively influence the response quality due to its degree of adherence to the previous system operation, disregarding new

environmental conditions. Online learning is vulnerable to drifting, *i.e.*, errors can be cumulative over time, leading to a large deviation between the actual environmental behavior, and the learned model. According to Sutton and Barto [2018], planning is a costly operation, thus it cannot be done in every decision step. Moreover, the agent should balance when to plan, improving the system response, and the impact on the QoE of increased system response time. Perhaps some of the planning can be done in parallel by updating the parameters of a separate Q-function, using the same process applied to the Q-network and target-network in DQL.

Transfer learning: In this topic we want to highlight two approaches. The first one is to use the controller-learned value function for a generic agent to initialize a new AP on the network. This experiment was not performed, and allows to verify the generalizability of the learned function and thus reduce the time that the AP needs to provide good results for the user. The second approach requires a new testbed, so one can test if the value function learned can be applied to another network configuration, i.e., an AP using this function converges faster than another one with a random initialization. These two aspects can be addressed in future work.

7.2 Publications

During the development of this thesis, we have contributed with some scientific papers listed below:

- **MOURA, Henrique**; MACEDO, Daniel; VIEIRA, Marcos. Wireless Control using Reinforcement Learning for Practical Web QoE. Accepted by Elsevier Computer Communications on January, 2020.
- **MOURA, Henrique**; ALVES, Alisson; BORGES, Jonas; MACEDO, Daniel; VIEIRA, Marcos. Ethanol: a Software-Defined Wireless Networking Architecture for IEEE 802.11 Networks. In: Elsevier Computer Communications, 2019.
- **MOURA, Henrique**; NUNES, Matheus; MIRANDA Júnior, Gilson; MACEDO, Daniel. Estimating Quality of Service on Wi-Fi Stations Using Recurrent Neural Networks. In: PIMRC 2019.
- **MOURA, Henrique**; MACEDO, Daniel; VIEIRA, Marcos. Automatic Quality of Experience Management for WLAN Networks using Multi-Armed Bandit. In: IFIP/IEEE International Symposium on Integrated Network Management – IM, 2019.

- GUIMARAES, J. C. T.; **MOURA, Henrique**; VIEIRA, M. A. M.; VIEIRA, L. F. M.; MACEDO, D. F. Alocação dinâmica de largura de banda para redes sem fio infraestruturadas In: Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC 2019, 2019, Gramado/RS.

- MIRANDA JUNIOR, G; **MOURA, Henrique**; NUNES, M. H. N.; CORREIA, L. H.; MACEDO, D. F. Estimativa de Qualidade de Serviço em Estações Wi-Fi Utilizando Redes Neurais Recorrentes In: Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC 2019, 2019, Gramado/RS.

- **MOURA, Henrique**; ALVES, A. R.; SOTO, J. H.; REIS, L. H. C.; Alex Vieira; MACEDO, D. F.; NOGUEIRA, J. M. S. Caracterização de Interrupções na Internet usando RIPE Atlas In: XXXVI Simpósio Brasileiro de Redes de Computadores, 2018, Campos do Jordão. Caracterização de Interrupções na Internet usando RIPE Atlas. Porto Alegre: Sociedade Brasileira de Computação (SBC), 2018.

- GUIMARAES, J. C. T.; **MOURA, Henrique**; BORGES, J. R. A.; VIEIRA, M. A. M.; VIEIRA, L. F. M.; MACEDO, D. F. Dynamic Bandwidth Allocation for Home and SOHO Wireless Networks In: 2018 IEEE Symposium on Computers and Communications (ISCC), 2018, Natal. 22th IEEE Symposium on Computers and Communications. IEEE, 2018. (short paper)

- SILVA, E. B. E.; **MOURA, Henrique**; FANELLI, G.; MIRANDA JUNIOR, M. R.; MACEDO, D. F.; VIEIRA, L. F. M.; VIEIRA, M. A. M. Comparison of Data Center Traffic Division Policies Using SDN In: IEEE/IFIP Network Operations and Management Symposium, 2018, Taiwan. IEEE/IFIP Network Operations and Management Symposium. , 2018.

- ALVES, A. R.; **MOURA, Henrique**; BORGES, J. R. A.; MOTA, V. F. S.; REIS, L. H. C.; MACEDO, D. F.; VIEIRA, M. A. M. HomeNetRescue: An SDN Service for Troubleshooting Home Networks In: IEEE/IFIP Network Operations and Management Symposium, 2018, Taiwan. IEEE/IFIP Network Operations and Management Symposium. , 2018.

- **MOURA, Henrique**; SILVA, E. B. E.; MACEDO, D. F. BWPING-UDP – Avaliando o Desempenho de Redes Sem Fio In: XXXV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, 2017, Belém. Anais do SBRC 2017 - Trilha Principal e Salão de Ferramentas. Porto Alegre: Sociedade Brasileira da Computação (SBC), 2017. p.1118 - 1124

- SILVA, E. B. E.; **MOURA, Henrique**; MACEDO, D. F.; VIEIRA, L. F. M.; VIEIRA, M. A. M. Comparação de Políticas de Divisão de Tráfego em Data Center Empregando SDN In: XXXV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC 2017, 2017, Belém. ANAIS SBRC 2017 - Trilha Principal e Salão de Ferramentas. Porto Alegre: Sociedade Brasileira de Computação (SBC), 2017. p.73 - 86

- **MOURA, Henrique**; MACEDO, D. F. Controle de roteamento em redes mesh via SDN-usando heurísticas para Unsplittable Flow Problem In: XLIX Simpósio Brasileiro de Pesquisa Operacional, 2017, Blumenau. XLIX Simpósio Brasileiro de Pesquisa Operacional - Trabalhos Completos - Metaheurísticas. SOBRAPO, 2017. p.1891 - 1902

- ALVES, A. R.; **Moura, Henrique D.**; REIS, L. H. C.; GUIMARAES, J. C. T.; BORGES, J. R. A.; SILVA, P. S.; MACEDO, D. F.; VIEIRA, M. A. M. Um Serviço SDN para Detecção e Solução de Problemas em Redes Domésticas In: XXII Workshop de Gerência e Operação de Redes e Serviços (WGRS), 2017, Belém. Anais do WGRS 2017. Porto Alegre: Sociedade Brasileira de Computação (SBC), 2017. p.96 - 109

- MOTA, V. F. S.; **MOURA, Henrique**; SILVA, V. F.; MACEDO, D. F.; GHAMRI-DOUDANE, Y.; NOGUEIRA, J. M. S. Analisando a Capacidade de Descarregamento de Redes Moveis por meio de Redes Oportunísticas In: Simpósio Brasileiro de Redes de Computadores (SBRC), 2016, Salvador. Trilha Principal. , 2016.

One journal submission is under review, and is listed below:

- **MOURA, Henrique**; MIRANDA Júnior, Gilson; GONÇALVES, André; MACEDO, Daniel; VIEIRA, Marcos. Improving Wireless Network Control using Reinforcement Learning – A survey. Submitted to IEEE Communications Surveys and Tutorials, in April, 2019.

Bibliography

Aguiar, E. S., Pinheiro, B. A., Figueiredo, J., Cerqueira, E., Abelem, A. J. G., and Gomes, R. L. (2011). Trends and Challenges for Quality of Service and Quality of Experience for Wireless Mesh Networks. In *Wireless Mesh Networks*, page 23. InTech.

Aharony, N., Zehavi, T., and Engel, Y. (2005). Learning wireless network association control with gaussian process temporal difference methods. In *Proceedings of OPNETWORK*.

Alberg, J. and Lipton, Z. C. (2017). Improving factor-based quantitative investing by forecasting company fundamentals. In *31st Conference on Neural Information Processing Systems*, page 5.

Alpaydin, E. (2009). *Introduction to machine learning*. MIT press.

Alsheikh, M. A., Lin, S., Niyato, D., and Tan, H.-P. (2014). Machine learning in wireless sensor networks: Algorithms, strategies, and applications. *IEEE Communications Surveys & Tutorials*, 16(4):1996--2018.

Alves, A. R., Duarte, H. M., Borges, J. R. A., Mota, V. F. S., Cantelli, L. H., Macedo, D. F., and Vieira, M. A. M. (2018). HomeNetRescue: an SDN Service for Troubleshooting Home Networks. *IEEE/IFIP Network Operations and Management Symposium (NOMS)*.

Alves, A. R., Moura, H. D., Reis, L. H. C., Guimaraes, J. C. T., Borges, J. R. A., Silva, P. S., Macedo, D. F., and Vieira, M. A. M. (2017). Um Serviço SDN para Detecção e Solução de Problemas em Redes Domésticas. *XXXV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - WGRS 2017*.

Amour, L., Mushtaq, M. S., Souihi, S., and Mellouk, A. (2017). Qoe-based framework to optimize user perceived video quality. In *2017 IEEE 42nd Conference on Local Computer Networks*, pages 599–602. ISSN 0742-1303.

Anegekuh, L., Sun, L., Jammeh, E., Mkwawa, I.-H., and Ifeachor, E. (2015). Content-based video quality prediction for HEVC encoded videos streamed over packet networks. *IEEE Transactions on Multimedia*, 17(8):1323--1334.

Aref, M. A., Jayaweera, S. K., and Machuzak, S. (2017). Multi-agent reinforcement learning based cognitive anti-jamming. In *2017 IEEE Wireless Communications and Networking Conference*, pages 1--6. IEEE.

Atallah, R. F., Assi, C. M., and Yu, J. Y. (2017). A reinforcement learning technique for optimizing downlink scheduling in an energy-limited vehicular network. *IEEE Transactions on Vehicular Technology*, 66(6):4592--4601.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multi-armed bandit problem. *Machine learning*, 47(2-3):235--256.

Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. E. (1995). Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 322--331. IEEE.

Auer, P. and Ortner, R. (2010). Ucb revisited: Improved regret bounds for the stochastic multi-armed bandit problem. *Periodica Mathematica Hungarica*, 61(1-2):55--65.

Bai, S., Kolter, J. Z., and Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*.

Baidas, M. W. (2014). Cooperation in wireless networks: a game-theoretic framework with reinforcement learning. *IET Communications*, 8(5):740--753.

Baltoglou, G., Karapistoli, E., and Chatzimisios, P. (2012). IPTV QoS and QoE measurements in wired and wireless networks. In *2012 IEEE Global Communications Conference*, pages 1757–1762. ISSN 1930-529X.

Baraković, S. and Skorin-Kapov, L. (2013). Survey and challenges of QoE management issues in wireless networks. *Journal of Computer Networks and Communications*, 2013.

Barco, R., Lazaro, P., and Munoz, P. (2012). A unified framework for self-healing in wireless networks. *IEEE Communications Magazine*, 50(12):134–142. ISSN 0163-6804.

Berger-Tal, O., Nathan, J., Meron, E., and Saltz, D. (2014). The exploration-exploitation dilemma: a multidisciplinary framework. *PloS one*, 9(4):e95693.

Bergstra, J. A. and Middelburg, C. (2003). ITU-T recommendation G.107: The e-model, a computational model for use in transmission planning.

Besson, L. and Kaufmann, E. (2018). What Doubling Tricks Can and Can't Do for Multi-Armed Bandits. *arXiv preprint arXiv:1803.06971*.

Bianchi, G. (2000). Performance analysis of the IEEE 802.11 distributed coordination function. *IEEE Journal on Selected Areas in Communications*, 18(3):535--547.

Biswas, S., Bicket, J., Wong, E., Musaloiu-e, R., Bhartia, A., and Aguayo, D. (2015). Large-scale measurements of wireless network behavior. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 153--165. ACM.

Boutaba, R., Salahuddin, M. A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., and Caicedo, O. M. (2018). A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1):16.

Bowling, M. and Veloso, M. (2002). Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215--250.

BroadBand Now (2018). Terrestrial Fixed Wireless Internet in the United States.

Brochu, E., Cora, V. M., and De Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*.

Busoniu, L., Babuska, R., and De Schutter, B. (2006). Multi-agent reinforcement learning: A survey. In *2006 9th International Conference on Control, Automation, Robotics and Vision*, pages 1--6. IEEE.

Buşoniu, L., Babuška, R., and De Schutter, B. (2010). Multi-agent reinforcement learning: An overview. In *Innovations in multi-agent systems and applications-1*, pages 183--221. Springer.

Buşoniu, L., de Bruin, T., Tolić, D., Kober, J., and Palunko, I. (2018). Reinforcement learning for control: Performance, stability, and deep approximators. *Annual Reviews in Control*.

Calyam, P., Kalash, A., Krishnamurthy, A., and Renkes, G. (2009). A human-and-network aware encoding adaptation scheme for remote desktop access. In *2009 IEEE International Workshop on Multimedia Signal Processing*, pages 1–6. ISSN .

Camps-Mur, D., Garcia-Saavedra, A., and Serrano, P. (2013). Device-to-device communications with wi-fi direct: overview and experimentation. *IEEE wireless communications*, 20(3):96--104.

Cesa-Bianchi, N., Gentile, C., Lugosi, G., and Neu, G. (2017). Boltzmann exploration done right. In *Advances in Neural Information Processing Systems*, pages 6284--6293.

Cesa-Bianchi, N. and Lugosi, G. (2006). *Prediction, learning, and games*. Cambridge university press.

Changuel, N., Sayadi, B., and Kieffer, M. (2012). Online learning for QoE-based video streaming to mobile receivers. In *2012 IEEE Globecom Workshops*, pages 1319--1324. IEEE.

Chapelle, O., Sindhwani, V., and Keerthi, S. S. (2008). Optimization techniques for semi-supervised support vector machines. *Journal of Machine Learning Research*, 9:203--233.

Chapelle, O. and Zien, A. (2005). Semi-Supervised Classification by Low Density Separation. In *AISTATS*, pages 57--64. Citeseer.

Chen, D. and Varshney, P. K. (2004). QoS Support in Wireless Sensor Networks: A Survey. In *International conference on wireless networks*, volume 233, pages 1--7.

Chen, K. and Duan, R. (2011). C-RAN –the road towards green RAN. Technical report, China Mobile. https://pdfs.semanticscholar.org/eaa3/ca62c9d5653e4f2318aed9ddb8992a505d3c.pdf.

Chenji, H., Haas, Z. J., and Xue, P. (2015). Low complexity qoe-aware bandwidth allocation for wireless content delivery. In *MILCOM 2015 - 2015 IEEE Military Communications Conference*, pages 419–425. ISSN .

Christophides, V., Teixeira, R., and Rossi, D. (2018). Narrowing the Gap Between QoS Metrics and Web QoE Using Above-the-fold Metrics. In *Passive and Active Measurement: 19th International Conference, PAM 2018, Berlin, Germany, March 26–27, 2018, Proceedings*, volume 10771, page 31. Springer.

Cisco (2017). Cisco Visual Networking Index: Forecast and Methodology, 2016 – 2021. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html#_Toc484813971.

Claeys, M., Latre, S., Famaey, J., and De Turck, F. (2014a). Design and evaluation of a self-learning HTTP adaptive video streaming client. *IEEE communications letters*, 18(4):716--719.

Claeys, M., Latré, S., Famaey, J., Wu, T., Van Leekwijck, W., and De Turck, F. (2013). Design of a Q-learning-based client quality selection algorithm for HTTP adaptive video streaming. In *Proceedings of the 2013 Workshop on Adaptive and Learning Agents, Saint Paul (Minn.), USA*, pages 30--37.

Claeys, M., Latré, S., Famaey, J., Wu, T., Van Leekwijck, W., and De Turck, F. (2014b). Design and optimisation of a (FA) Q-learning-based HTTP adaptive streaming client. *Connection Science*, 26(1):25--43.

Clark, D. D., Partridge, C., Ramming, J. C., and Wroclawski, J. T. (2003). A knowledge plane for the internet. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 3--10. ACM.

Clinckx, N. and Baffalio, Y. (2014). Optimize network OPEX and CAPEX while enhancing the quality of service. Technical report, EY.

Comsa, I., Trestian, R., and Ghinea, G. (2018). 360° mulsemedia experience over next generation wireless networks - a reinforcement learning approach. In *2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–6. ISSN 2472-7814.

Coutinho, N., Matos, R., Marques, C., Reis, A., Sargento, S., Chakareski, J., and Kassler, A. (2015). Dynamic dual-reinforcement-learning routing strategies for quality of experience-aware wireless mesh networking. *Comput. Netw.*, 88(C):269--285. ISSN 1389-1286.

Crawley, E. S., Nair, R., Rajagopalan, B., and Sandick, H. (1998). A Framework for QoS-based Routing in the Internet. RFC 2386, RFC Editor. http://www.rfc-editor.org/rfc/rfc2386.txt.

Dalal, A. C. (2011). User-perceived quality assessment of streaming media using reduced feature sets. *ACM Transactions on Internet Technology*, 11(2):8.

Dalal, A. C., Bouchard, A. K., Cantor, S., Guo, Y., and Johnson, A. (2012). Assessing QoE of on-demand TCP video streams in real time. In *2012 IEEE International Conference on Communications*, pages 1165–1170. ISSN 1938-1883.

Daneshgaran, F., Laddomada, M., Mesiti, F., and Mondin, M. (2008). Unsaturated throughput analysis of IEEE 802.11 in presence of non ideal transmission channel and capture effects. *IEEE Transactions on Wireless Communications*, 7(4):1276–1286.

De Vriendt, J., De Vleeschauwer, D., and Robinson, D. (2013). Model for estimating QoE of video delivered using HTTP adaptive streaming. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 1288--1293. IEEE.

Demirbilek, E. and Grégoire, J.-C. (2016). Inrs audiovisual quality dataset. In *Proceedings of the 2016 ACM on Multimedia Conference*, MM '16, pages 167--171.

Demirbilek, E. and Grégoire, J.-C. (2017). Machine Learning–Based Parametric Audiovisual Quality Prediction Models for Real-Time Communications. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 13(2):16.

DiCioccio, L., Teixeira, R., and Rosenberg, C. (2011). Characterizing home networks with homenet profiler. *UPMC Sorbonne Universits, Tech. Rep. CP-PRL-2011-09-0001*.

Dorf, R. C. and Bishop, R. H. (2011). *Modern control systems*. Pearson.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. *CoRR*, abs/1604.06778.

Dubin, R., Dvir, A., Pele, O., and Hadar, O. (2017). I Know What You Saw Last Minute—Encrypted HTTP Adaptive Video Streaming Title Classification. *IEEE Transactions on Information Forensics and Security*, 12(12):3039--3049.

Egger, S., Hossfeld, T., Schatz, R., and Fiedler, M. (2012). Waiting times in quality of experience for web based services. In *2012 Fourth International Workshop on Quality of Multimedia Experience*, pages 86--96. IEEE.

Egilmez, H. E., Civanlar, S., and Tekalp, A. M. (2013). An optimization framework for QoS-enabled adaptive video streaming over OpenFlow networks. *IEEE Transactions on Multimedia*, 15(3):710--715.

Ehsan, S. and Hamdaoui, B. (2012). A survey on energy-efficient routing techniques with QoS assurances for wireless multimedia sensor networks. *IEEE Communications Surveys & Tutorials*, 14(2):265--278.

Engel, Y., Mannor, S., and Meir, R. (2005). Reinforcement learning with Gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 201--208. ACM.

Engelberg, S. (2015). *A mathematical introduction to control theory*, volume 4. World Scientific Publishing Company.

Engmann, S. and Cousineau, D. (2011). Comparing distributions: the two-sample Anderson-Darling test as an alternative to the Kolmogorov-Smirnoff test. *Journal of applied quantitative methods*, 6(3):1--17.

Ericsson AB (2018). 5G subscriptions to reach half a billion in 2022: Ericsson Mobility Report. https://www.ericsson.com/TET/trafficView/loadBasicEditor.ericsson. Accessed: 2018-07-24.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. (2018). Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*.

Fiedler, M., Hossfeld, T., and Tran-Gia, P. (2010). A generic quantitative relationship between quality of experience and quality of service. *IEEE Network*, 24(2).

Fisher, A., Rudin, C., and Dominici, F. (2018). Model class reliance: Variable importance measures for any machine learning model class, from the "rashomon" perspective. *arXiv preprint arXiv:1801.01489*.

Floodlight-dev (2016). How to Collect Switch Statistics (and Compute Bandwidth Utilization). https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/21856267/How+to+Collect+Switch+Statistics+and+Compute+Bandwidth+Utilization. Accessed: 2018-07-04.

Franklin, G. F., Powell, J. D., and Emami-Naeini, A. (2014). *Feedback control of dynamic systems*. Prentice Hall Press.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189--1232.

Gadaleta, M., Chiariotti, F., Rossi, M., and Zanella, A. (2017). D-DASH: A Deep Q-Learning Framework for DASH Video Streaming. *IEEE Transactions on Cognitive Communications and Networking*, 3(4):703--718.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):44.

Ghahfarokhi, B. S. and Movahhedinia, N. (2013). A personalized QoE-aware handover decision based on distributed reinforcement learning. *Wireless networks*, 19(8):1807--1828.

Gieseke, F., Airola, A., Pahikkala, T., and Kramer, O. (2012). Sparse Quasi-Newton Optimization for Semi-supervised Support Vector Machines. In *International Conference on Pattern Recognition Applications and Methods*, pages 45--54.

Gittins, J., Glazebrook, K., and Weber, R. (2011). *Multi-armed bandit allocation indices*. John Wiley & Sons.

Goli, Y. D. and Ambika, R. (2018). Network traffic classification techniques-a review. In *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS)*, pages 219–222. ISSN .

Gomes, R. L., Júnior, J. J., Abelém, A. G., and Júnior, W. M. (2009). QoE and QoS support on Wireless Mesh Networks. In *Proceedings of the XV Brazilian Symposium on Multimedia and the Web*, page 2. ACM.

Guimaraes, J. C., Moura, H., Borges, J., Vieira, M. A. M., Vieira, L. F. M., and Macedo, D. F. (2018). Dynamic Bandwidth Allocation for Home and Soho Wireless Networks. In *2018 IEEE Symposium on Computers and Communications*, Natal, Brazil.

Gummeson, J., Ganesan, D., Corner, M. D., and Shenoy, P. (2010). An adaptive link layer for heterogeneous multi-radio mobile sensor networks. *IEEE Journal on Selected Areas in Communications*, 28(7).

Habachi, O., Hu, Y., Van der Schaar, M., Hayel, Y., and Wu, F. (2012). MOS-based congestion control for conversational services in wireless environments. *IEEE Journal on Selected Areas in Communications*, 30(7):1225--1236.

Harel, M., Mannor, S., El-Yaniv, R., and Crammer, K. (2014). Concept drift detection through resampling. In *International Conference on Machine Learning*, pages 1009--1017.

Harishankar, M., Pilaka, S., Sharma, P., Srinivasan, N., Joe-Wong, C., and Tague, P. (2019). Procuring Spontaneous Session-Level Resource Guarantees for Real-Time Applications: An Auction Approach. *IEEE Journal on Selected Areas in Communications*, 37(7):1534--1548.

Hasselt, H. V. (2010). Double Q-learning. In *Advances in Neural Information Processing Systems*, pages 2613--2621.

He, J. and Pung, H. K. (2006). Performance modelling and evaluation of IEEE 802.11 distributed coordination function in multihop wireless networks. *Computer Communications*, 29(9):1300--1308. ISSN 0140-3664.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026--1034.

He, X., Wang, K., Huang, H., Miyazaki, T., Wang, Y., and Guo, S. (2018). Green resource allocation based on deep reinforcement learning in content-centric IoT. *IEEE Transactions on Emerging Topics in Computing*.

He, Y., Zhang, Z., Yu, F. R., Zhao, N., Yin, H., Leung, V. C., and Zhang, Y. (2017). Deep-reinforcement-learning-based optimization for cache-enabled opportunistic interference alignment wireless networks. *IEEE Transactions on Vehicular Technology*, 66(11):10433--10445.

Herbaut, N., Négru, D., Chen, Y., Frangoudis, P. A., and Ksentini, A. (2016). Content delivery networks as a virtual network function: A win-win isp-cdn collaboration. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1--6. IEEE.

Hochstadt, A., Newman, S., Greenberfg, R., and Aflalo, K. (2018). Internet trends 2018. stats & facts in the u.s. and worldwide. https://www.vpnmentor.com/blog/vital-internet-trends/. Accessed: 2018-07-20.

Hora, D. N. d., Teixeira, R., van Doorselaer, K., and van Oost, K. (2016). Predicting the Effect of Home Wi-Fi Quality on Web QoE. In *Proceedings of the 2016 Workshop on QoE-based Analysis and Management of Data Communication Networks*, Internet-QoE '16, pages 13--18, New York, NY, USA. ACM.

Hoßfeld, T. and Binzenhöfer, A. (2008). Analysis of Skype VoIP traffic in UMTS: End-to-end QoS and QoE measurements. *Computer Networks*, 52(3):650--666.

Hoßfeld, T., Heegaard, P. E., Skorin-Kapov, L., and Varela, M. (2017). No silver bullet: QoE metrics, QoE fairness, and user diversity in the context of QoE management. In *2017 Ninth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1--6. IEEE.

Hoßfeld, T., Skorin-Kapov, L., Heegaard, P. E., and Varela, M. (2016). Definition of QoE fairness in shared systems. *IEEE Communications Letters*, 21(1):184--187.

Hoßfeld, T., Skorin-Kapov, L., Heegaard, P. E., and Varela, M. (2018). A new QoE fairness index for QoE management. *Quality and User Experience*, 3(1):4. ISSN 2366-0147.

Huynh-Thu, Q. and Ghanbari, M. (2012). The accuracy of PSNR in predicting video quality for different video scenes and frame rates. *Telecommunication Systems*, 49(1):35--48.

Issa, O., Speranza, F., Li, W., and Liu, H. (2012). Estimation of time varying QoE for high definition IPTV distribution. In *2012 IEEE Consumer Communications and Networking Conference*, pages 326–330. ISSN 2331-9852.

ITU (2018). World Telecommunication/ICT Development Report and database. http://data.worldbank.org/indicator/IT.CEL.SETS/. Accessed: 2018-07-24.

ITU-T (2007). Definition of quality of experience (QoE), liaison statement. Recommendation P.10/G.100, Amd 1, ITU Telecommunication Standardization Sector and OF ITU, Geneva.

ITU-T Recommendation P.10/G.100 (2016). Vocabulary for performance and quality of service. Amendment 5: New definitions for inclusion in Recommendation ITU-T P.10/G.100.

Jaber, M., Imran, M. A., Tafazolli, R., and Tukmanov, A. (2016). A multiple attribute user-centric backhaul provisioning scheme using distributed SON. In *2016 IEEE Global Communications Conference*, pages 1--6. IEEE.

Jiang, W., Zhang-Shen, R., Rexford, J., and Chiang, M. (2009). Cooperative content distribution and traffic engineering in an isp network. In *ACM SIGMETRICS Performance Evaluation Review*, volume 37, pages 239--250. ACM.

Joshi, H. (2018). Technology, Media and Telecommunications Predictions 2018 - India edition. Technical report, Deloitte Touche Tohmatsu Limited, a.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237--285.

Kaisers, M. and Tuyls, K. (2010). Frequency adjusted multi-agent Q-learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 309--316. International Foundation for Autonomous Agents and Multiagent Systems.

Kalchbrenner, N., Espeholt, L., Simonyan, K., Oord, A. v. d., Graves, A., and Kavukcuoglu, K. (2016). Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*.

Kamboh, U. R., Yang, Q., and Qin, M. (2017). Impact of Self-Organizing Networks Deployment on Wireless Service Provider Businesses in China. *International Journal of Communications, Network and System Sciences*, 10(05):78.

Kazemi, R., Vesilo, R., Dutkiewicz, E., and Liu, R. P. (2012). Reinforcement learning in power control games for internetwork interference mitigation in wireless body area networks. In *2012 International Symposium on Communications and Information Technologies*, pages 256--262. IEEE.

Kelleher, J. D., Mac Namee, B., and D'Arcy, A. (2015). *Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies*. MIT Press.

Khan, A., Sun, L., and Ifeachor, E. (2009a). Content-Based Video Quality Prediction for MPEG4 Video Streaming over Wireless Networks. *Journal of Multimedia*, 4(4).

Khan, A., Sun, L., and Ifeachor, E. (2009b). Content Classification Based on Objective Video Quality Evaluation for MPEG4 Video Streaming over Wireless Networks. In *Proceedings of the World Congress on Engineering*, volume 1, pages 1--3.

Khan, A., Sun, L., Jammeh, E., and Ifeachor, E. (2010). Quality of experience-driven adaptation scheme for video applications over wireless networks. *IET communications*, 4(11):1337--1347.

Khan, M. I. and Rinner, B. (2012). Resource coordination in wireless sensor networks by cooperative reinforcement learning. In *2012 IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 895--900. IEEE.

Khan, S., Duhovnikov, S., Steinbach, E., and Kellerer, W. (2007). MOS-based multiuser multiapplication cross-layer optimization for mobile multimedia communication. *Advances in Multimedia*, 2007(1):6--6.

Khorsandroo, S., Noor, R. M., and Khorsandroo, S. (2012). The role of psychophysics laws in quality of experience assessment: a video streaming case study. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, pages 446--452. ACM.

Kim, H. J., Choi, S. G., Kim, H. S., and Choi, S. G. (2010). A study on a QoS/QoE correlation model for QoE evaluation on IPTV service. In *2010 The 12th International Conference on Advanced Communication Technology*, volume 2, pages 1377--1382. IEEE.

Kim, H. J., Yun, D. G., Kim, H.-S., Cho, K. S., and Choi, S. G. (2012). QoE assessment model for video streaming service using QoS parameters in wired-wireless network. In *2012 14th International Conference on Advanced Communication Technology (ICACT)*, pages 459--464. IEEE.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR*.

Kirk, D. E. (2012). *Optimal control theory: an introduction*. Courier Corporation.

Klaue, J., Rathke, B., and Wolisz, A. (2003). Evalvid–a framework for video transmission and quality evaluation. In *International conference on modelling techniques and tools for computer performance evaluation*, pages 255--272. Springer.

Kok, J. R. and Vlassis, N. (2006). Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 7(Sep):1789--1828.

Kuleshov, V. and Precup, D. (2014). Algorithms for multi-armed bandit problems. *arXiv preprint arXiv:1402.6028*.

Lai, T. L. and Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4--22.

Li, W. and Moore, A. W. (2007). A machine learning approach for efficient traffic classification. In *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2007. MASCOTS'07. 15th International Symposium on*, pages 310--317. IEEE.

Littman, M. L. (2001). Friend-or-foe Q-learning in general-sum games. In *International Conference on Machine Learning*, volume 1, pages 322--328.

Liu, J., Tao, X., and Lu, J. (2018). QoE-Oriented Rate Adaptation for DASH with enhanced Deep Q-Learning. *IEEE Access*, 7:8454--8469.

Liu, Y., He, W., Wang, Y., and Yang, H. (2019). Network-assisted neural adaptive naked-eye 3d video streaming over wireless networks. *IEEE Access*, 7:141363--141373.

Macedo, D. F., Guedes, D., Vieira, L. F. M., Vieira, M. A. M., and Nogueira, M. (2015). Programmable Networks – From Software-Defined Radio to Software-Defined Networking. *IEEE Communications Surveys & Tutorials*, 17(2):1102--1125.

Machado, V. A., Silva, C. N., Oliveira, R. S., Melo, A. M., Silva, M., Francês, C. R., Costa, J. C., Vijaykumar, N. L., and Hirata, C. M. (2011). A new proposal to provide estimation of QoS and QoE over WiMAX networks: An approach based on computational intelligence and discrete-event simulation. In *2011 IEEE Latin-American Conference on Communications*, pages 1--6. IEEE.

Maes, F., Wehenkel, L., and Ernst, D. (2012). Learning to play k-armed bandit problems. In *Proceedings of the 4th International Conference on Agents and Artificial Intelligence (2012)*.

Mahmoud, Q. (2007). *Cognitive networks: towards self-aware networks*. John Wiley & Sons.

Malik, A., Qadir, J., Ahmad, B., Yau, K.-L. A., and Ullah, U. (2015). QoS in IEEE 802.11-based wireless networks: a contemporary review. *Journal of Network and Computer Applications*, 55:24--46.

Mao, H., Netravali, R., and Alizadeh, M. (2017). Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 197--210. ACM.

Marchetti, N., Prasad, N. R., Johansson, J., and Cai, T. (2010). Self-organizing networks: State-of-the-art, challenges and perspectives. In *2010 8th International Conference on Communications*, pages 503--508. IEEE.

Matos, R., Coutinho, N., Marques, C., Sargento, S., Chakareski, J., and Kassler, A. (2012). Quality of experience-based routing in multi-service wireless mesh networks. In *2012 IEEE International Conference on Communications*, pages 7060--7065. IEEE.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69--74.

Miller, T. (2018). Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*.

Mishra, A., Shrivastava, V., Agrawal, D., Banerjee, S., and Ganguly, S. (2006). Distributed channel management in uncoordinated wireless environments. In *Proceedings of the 12th annual international conference on Mobile computing and networking*, pages 170--181. ACM.

Mismar, F. B. and Evans, B. L. (2018). Q-learning algorithm for volte closed loop power control in indoor small cells. In *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, pages 1485--1489. IEEE.

Mitra, D., Wang, Q., and Hong, A. (2016). Social welfare and policy impact on emerging isp-content provider relationships. In *The 44th Research Conference on Communication, Information and Internet Policy*. TPRC.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928--1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529--533.

Mohamed, S. and Rubino, G. (2014). On the Accuracy of PSNR and PSQA video quality measures. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, pages 1--5.

Mohammadi, F. S. and Kwasinski, A. (2018). QoE-Driven Integrated Heterogeneous Traffic Resource Allocation Based on Cooperative Learning for 5G Cognitive Radio Networks. In *2018 IEEE 5G World Forum (5GWF)*, pages 244--249. IEEE.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012). *Foundations of machine learning*. MIT press.

Mok, R. K. P., Luo, X., Chan, E. W. W., and Chang, R. K. C. (2012). QDASH: A QoE-aware DASH System. In *Proceedings of the 3rd Multimedia Systems Conference*, MMSys '12, pages 11--22.

Molnar, C. et al. (2019). Interpretable machine learning: A guide for making black box models explainable. *E-book at< https://christophm. github. io/interpretable-ml-book/>, version dated*.

Morocho-Cayamcela, M. E., Lee, H., and Lim, W. (2019). Machine learning for 5g/b5g mobile and wireless communications: Potential, limitations, and future directions. *IEEE Access*, 7:137184--137206.

Moura, H., Alves, A. R., Borges, J. R., Macedo, D. F., and Vieira, M. A. (2019a). Ethanol: A Software-Defined Wireless Networking architecture for IEEE 802.11 networks. *Computer Communications*.

Moura, H., Bessa, G. V. C., Vieira, M. A. M., and Macedo, D. F. (2015). Ethanol: Software Defined Networking for 802.11 Wireless Networks. *IFIP/IEEE International Symposium on Integrated Network Management*.

Moura, H., Nunes, M., Gilson Miranda, J., Macedo, D. F., and Correia, L. H. A. (2019b). Estimating Quality of Service on Wi-Fi Stations Using Recurrent Neural Networks. In *2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC): Track 3: Mobile and Wireless Networks (IEEE PIMRC 2019 - Track 3)*, Istanbul, Turkey.

Moura, H. D., Macedo, D. F., and Vieira, M. A. (2019c). Automatic quality of experience management for wlan networks using multi-armed bandit. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 279--288. IEEE.

Moura, H. D., Macedo, D. F., and Vieira, M. A. (2020). Wireless Control using Reinforcement Learning for Practical Web QoE. *Computer Communications*.

Mushtaq, M. S., Augustin, B., and Mellouk, A. (2012). Empirical study based on machine learning approach to assess the QoS/QoE correlation. In *2012 17th European Conference on Networks and Optical Communications*, pages 1--7. IEEE.

Nasrabadi, A., Shirsavar, M., Ebrahimi, A., and Ghanbari, M. (2014). Investigating the PSNR calculation methods for video sequences with source and channel distortions. In *2014 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting*, pages 1--4. IEEE.

Ng, A. Y., Jordan, M. I., and Weiss, Y. (2002). On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849--856.

Nguyen, T. T. and Armitage, G. (2008). A survey of techniques for internet traffic classification using machine learning. *IEEE communications surveys & tutorials*, 10(4):56--76.

Ni, Q., Romdhani, L., and Turletti, T. (2004). A survey of qos enhancements for ieee 802.11 wireless lan. *Wireless Communications and Mobile Computing*, 4(5):547--566.

Nilsson, N. J. (1996). Introduction to machine learning: An early draft of a proposed textbook.

Nishida, K. (2008). *Learning and detecting concept drift*. PhD thesis, Hokkaido University.

Olsson, M., Cavdar, C., Frenger, P., Tombaz, S., Sabella, D., and Jantti, R. (2013). 5GrEEn: Towards Green 5G mobile networks. In *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications*, pages 212--216. IEEE.

Oord, A. v. d., Kalchbrenner, N., and Kavukcuoglu, K. (2016). Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*.

Opara-Martins, J., Sahandi, R., and Tian, F. (2016). Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective. *Journal of Cloud Computing*, 5(1):4.

Orsolic, I., Pevec, D., Suznjevic, M., and Skorin-Kapov, L. (2017). A machine learning approach to classifying YouTube QoE based on encrypted network traffic. *Multimedia tools and applications*, 76(21):22267--22301.

Papadopouli, M., Charonyktakis, P., Plakia, M., and Tsamardinos, I. (2015). On user-centric modular QoE prediction for VoIP based on Machine-learning algorithms. *IEEE Transactions on mobile computing*, 1:1--1.

Paudel, I., Pokhrel, J., Wehbi, B., Cavalli, A., and Jouaber, B. (2014). Estimation of video QoE from MAC parameters in wireless network: A Random Neural Network approach. In *2014 14th International Symposium on Communications and Information Technologies*, pages 51--55. IEEE.

Petrangeli, S., Claeys, M., Latré, S., Famaey, J., and De Turck, F. (2014). A multi-agent q-learning-based framework for achieving fairness in http adaptive streaming. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1--9. IEEE.

Piamrat, K., Viho, C., Bonnin, J.-M., and Ksentini, A. (2009). Quality of experience measurements for video streaming over wireless networks. In *Information Technology: New Generations, 2009. ITNG'09. Sixth International Conference on*, pages 1184--1189. IEEE.

Pokhrel, J., Wehbi, B., Morais, A., Cavalli, A., and Allilaire, E. (2013). Estimation of QoE of video traffic using a fuzzy expert system. In *2013 IEEE 10th Consumer Communications and Networking Conference*, pages 224–229. ISSN 2331-9852.

Priyadarshana, W. and Sofronov, G. (2015). Multiple break-points detection in array CGH data via the cross-entropy method. *IEEE/ACM transactions on computational biology and bioinformatics*, 12(2):487--498.

Ravi, S. and Shah, D. (2014). A Survey of Adaptive Streaming over HTTP. *IOSR Journal of Electronics and Communication Engineering*, 9(2).

Recommendation, I.-T. (2001). Perceptual evaluation of speech quality (PESQ): An objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs. *Rec. ITU-T P. 862*.

Reichl, P., Egger, S., Schatz, R., and D'Alconzo, A. (2010). The logarithmic nature of QoE and the role of the Weber-Fechner law in QoE assessment. In *2010 IEEE International Conference on Communications*, pages 1--5. IEEE.

Reis, A. B., Chakareski, J., Kassler, A., and Sargento, S. (2010). Distortion optimized multi-service scheduling for next-generation wireless mesh networks. In *INFOCOM IEEE Conference on Computer Communications Workshops, 2010*, pages 1--6. IEEE.

Sayed-Mouchaweh, M. and Lughofer, E. (2012). *Learning in non-stationary environments: methods and applications*. Springer Science & Business Media.

Schölkopf, B., Smola, A. J., Williamson, R. C., and Bartlett, P. L. (2000). New support vector algorithms. *Neural computation*, 12(5):1207--1245.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Schwartz, H. M. (2014). *Multi-agent machine learning: A reinforcement approach*. John Wiley & Sons.

Seah, M. W. M., Tham, C.-K., Srinivasan, V., and Xin, A. (2007). Achieving coverage through distributed reinforcement learning in wireless sensor networks. In *Intelligent Sensors, Sensor Networks and Information, 2007. ISSNIP 2007. 3rd International Conference on*, pages 425--430. IEEE.

Seddiki, M. S., Shahbaz, M., Donovan, S., Grover, S., Park, M., Feamster, N., and Song, Y.-Q. (2014). FlowQoS: QoS for the rest of us. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 207--208. ACM.

Serral-Gracià, R., Cerqueira, E., Curado, M., Yannuzzi, M., Monteiro, E., and Masip-Bruin, X. (2010). An overview of quality of experience measurement challenges for video applications in IP networks. In *International Conference on Wired/Wireless Internet Communications*, pages 252--263. Springer.

Shaikh, J., Fiedler, M., and Collange, D. (2010). Quality of experience from user and network perspectives. *Annals of Telecommunications*, 65(1-2):47--57.

Shirazi, G. N., Kong, P.-Y., and Tham, C.-K. (2009). Cooperative retransmissions using Markov decision process with reinforcement learning. In *Personal, Indoor and Mobile Radio Communications, 2009 IEEE 20th International Symposium on*, pages 652--656. IEEE.

Smola, A. J. and Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and computing*, 14(3):199--222.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951--2959.

Soldani, D., Li, M., and Cuny, R. (2007). *QoS and QoE management in UMTS cellular systems*. John Wiley & Sons.

Song, J., Yang, F., Zhou, Y., Wan, S., and Wu, H. R. (2016). QoE evaluation of multimedia services based on audiovisual quality and user interest. *IEEE Transactions on Multimedia*, 18(3):444--457.

Souidi, M., Souihi, S., Hoceini, S., and Mellouk, A. (2015). An adaptive real time mechanism for IaaS cloud provider selection based on QoE aspects. In *2015 IEEE International Conference on Communications*, pages 6809–6814. ISSN 1550-3607.

Spiteri, K., Sitaraman, R., and Sparacio, D. (2019). From theory to practice: Improving bitrate adaptation in the dash reference player. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 15(2s):67.

Spiteri, K., Urgaonkar, R., and Sitaraman, R. K. (2016). BOLA: near-optimal bitrate adaptation for online videos. *CoRR*, abs/1601.06748.

Statista (2018). Number of mobile (cellular) subscriptions worldwide from 1993 to 2017 (in millions). https://www.statista.com/statistics/262950/global-mobile-subscriptions-since-1993/. Accessed: 2018-07-24.

Stefan Winkler, R. C. (2003). Video quality evaluation for internet streaming applications. *Proc.SPIE*, 5007:5007 – 5007 – 12.

Stefano, B., Francesco, D. P., Claudio, G. G., Salvatore, M., Martina, P., Antonio, P., Lorenzo, R. C., and Vincenzo, S. (2015). A multi-agent reinforcement learning based approach to Quality of Experience control in Future Internet networks. In *2015 34th Chinese Control Conference*, pages 6495--6500. IEEE.

Stooke, A. and Abbeel, P. (2018). Accelerated methods for deep reinforcement learning. *arXiv preprint arXiv:1803.02811*.

Sugiyama, M. (2015). *Statistical reinforcement learning: modern machine learning approaches*. Chapman and Hall/CRC.

Sugiyama, M. and Kawanabe, M. (2012). *Machine learning in non-stationary environments: Introduction to covariate shift adaptation*. MIT press.

Sun, Y., Yin, X., Jiang, J., Sekar, V., Lin, F., Wang, N., Liu, T., and Sinopoli, B. (2016). CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 272--285. ACM.

Sundaresan, S., Feamster, N., and Teixeira, R. (2016). Home network or access link? locating last-mile downstream throughput bottlenecks. In *International Conference on Passive and Active Network Measurement*, pages 111--123. Springer.

Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160--163.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press, Cambridge, Massachusetts, 3 edition.

Szepesvári, C. (2010). Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1--103.

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330--337.

Testolin, A., Zanforlin, M., De Grazia, M. D. F., Munaretto, D., Zanella, A., Zorzi, M., and Zorzi, M. (2014). A machine learning approach to QoE-based video admission control and resource allocation in wireless systems. In *2014 13th Annual Mediterranean Ad Hoc Networking Workshop*, pages 31--38. Citeseer.

Tokic, M. and Palm, G. (2011). Value-difference based exploration: adaptive control between epsilon-greedy and softmax. In *Annual Conference on Artificial Intelligence*, pages 335--346. Springer.

Tran, H. A., Mellouk, A., Hoceini, S., and Augustin, B. (2012). Global state-dependent QoE based routing. In *2012 IEEE International Conference on Communications*, pages 131--135. IEEE.

Tsamardinos, I., Rakhshani, A., and Lagani, V. (2015). Performance-estimation properties of cross-validation-based protocols with simultaneous hyper-parameter optimization. *International Journal on Artificial Intelligence Tools*, 24(05):1540023.

Valente Klaine, P., Imran, M. A., Onireti, O., and Souza, R. D. (2017). A survey of machine learning techniques applied to self organizing cellular networks. *IEEE Communications Surveys and Tutorials*.

van der Hooft, J., Petrangeli, S., Wauters, T., Huysegems, R., Alface, P. R., Bostoen, T., and De Turck, F. (2016). HTTP/2-based adaptive streaming of HEVC video over 4G/LTE networks. *IEEE Communications Letters*, 20(11):2177--2180.

Verma, G. and Dean, F. (2015). QoS support in wireless sensor networks. *Journal of Network Communications and Emerging Technologies*, 5(2).

Vermorel, J. and Mohri, M. (2005). Multi-armed bandit algorithms and empirical evaluation. In *European conference on machine learning*, pages 437--448. Springer.

Wang, Y., Zhou, W., and Zhang, P. (2017). *QoE management in wireless networks*. Springer.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, King's College, Cambridge.

Williams, C. K. and Seeger, M. (2001). Using the Nyström method to speed up kernel machines. In *Advances in neural information processing systems*, pages 682--688.

Wolpert, D. H., Macready, W. G., et al. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67--82.

WOWZA (2012). VRT Presents First Live Public Trial of MPEG-DASH During London Olympics. https://www.wowza.com/news/vrt-presents-first-live-public-trial-of-mpeg-dash-during-london-olympics. Accessed: 2018-08-20.

Wu, Q., Li, Z., and Xie, G. (2013). Codingcache: multipath-aware CCN cache with network coding. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*, pages 41--42. ACM.

Wu, Y., Hu, F., Kumar, S., Zhu, Y., Talari, A., Rahnavard, N., and Matyjas, J. D. (2014). A learning-based QoE-driven spectrum handoff scheme for multimedia transmissions over cognitive radio networks. *IEEE Journal on Selected Areas in Communications*, 32(11):2134--2148.

Xing, M., Xiang, S., and Cai, L. (2014). A real-time adaptive algorithm for video streaming over multiple wireless access networks. *IEEE Journal on Selected Areas in communications*, 32(4):795--805.

Yamagishi, K. and Hayashi, T. (2008). Parametric packet-layer model for monitoring video quality of IPTV services. In *2008, IEEE International Conference on Communications*, pages 110--114. IEEE.

Yang, E. and Gu, D. (2004). Multiagent reinforcement learning for multi-robot systems: A survey. Technical report, tech. rep.

Yau, K.-L., Komisarczuk, P., and Teal, P. D. (2010). Achieving efficient and optimal joint action in distributed cognitive radio networks using Payoff Propagation. In *Communications, 2010 IEEE International Conference on Communications*, pages 1--6. IEEE.

Yin, J., Mao, Y., Leng, S., Wang, X., and Fu, H. (2015a). QoE provisioning by random access in next-generation wireless networks. In *2015 IEEE Global Communications Conference*, pages 1--7. IEEE.

Yin, X., Jindal, A., Sekar, V., and Sinopoli, B. (2015b). A control-theoretic approach for dynamic adaptive video streaming over HTTP. *ACM SIGCOMM Computer Communication Review*, 45(4):325--338.

Yogeswaran, M. and Ponnambalam, S. (2012). Reinforcement learning: exploration–exploitation dilemma in multi-agent foraging task. *Opsearch*, 49(3):223--236.

You, F., Zhang, W., and Xiao, J. (2009). Packet loss pattern and parametric video quality model for IPTV. In *2009 Eigth IEEE/ACIS International Conference on Computer and Information Science*, pages 824--828. IEEE.

Yousaf, F. Z., Mämmelä, O., and Mannersalo, P. (2014). Reinforcement learning method for QoE-aware optimization of content delivery. In *2014 IEEE Wireless Communications and Networking Conference*, pages 3390--3395. IEEE.

Yu, X., Yang, J., and Zhang, J.-p. (2012). A transductive support vector machine algorithm based on spectral clustering. *AASRI Procedia*, 1:384--388.

Zelnik-Manor, L. and Perona, P. (2005). Self-tuning spectral clustering. In *Advances in neural information processing systems*, pages 1601--1608.

Zhang, K. and Pan, W. (2006). The two facets of the exploration-exploitation dilemma. In *Intelligent Agent Technology, 2006. IAT'06. IEEE/WIC/ACM International Conference on*, pages 371--380. IEEE.

Zhang, X., Chen, H., Zhao, Y., Ma, Z., Xu, Y., Huang, H., Yin, H., and Wu, D. O. (2019). Improving Cloud Gaming Experience through Mobile Edge Computing. *IEEE Wireless Communications*.

Zhu, H., Wang, H., Luo, X., and Qian, H. (2018). An online learning approach to wireless computation offloading. In *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 678--682. IEEE.

Zineb, A. B., Ayadi, M., and Tabbane, S. (2015). Cognitive radio networks management using an ANFIS approach with QoS/QoE mapping scheme. In *2015 International Symposium on Networks, Computers and Communications*, pages 1--6. IEEE.