A Deep Reinforcement Learning approach for Optimization and Task-offloading of Mobile Edge Computing in Virtual Radio Access Networks

Author: Jordi BIOSCA CARO
Supervisors: Nicola CALABRETTA, Salvatore SPADARO, and Bitao PAN

A thesis submitted in fulfillment of the requirements for the master’s degree of Master’s degree in Telecommunications Engineering in the Faculty of Electrical Engineering Eindhoven University of Technology & Technical University of Catalonia

October 15, 2020
Abstract

Mobile systems are increasing in number and, in the future, exponential growth is expected with the deployment of new technologies like 5G and Internet of Things. Requirements from those technologies lead to an improvement from the existent techniques to new sophisticated ones. A key role in future developments, which are already applied in research and industry, are Software Defined Networks (SDN) and Network Function Virtualization (NFV). Therefore, we present a solution for mobile edge computing (MEC) using a deep reinforcement learning (DRL) algorithm to optimize and offload tasks in a scenario of a virtual radio access network (VRANs). Final chapters show results obtained from experiments where the learning agent improves its reward through time benefiting the amount of bandwidth used in the network. Finally, a chapter discussing about the conclusions arise with interesting future work which could potentially lead to better results.
Acknowledgements

On one hand, I would like to thank professor Nicola Calabretta to allow me to participate and work along with his colleagues at the Electro-Optical group at Eindhoven University of Technology (TU/e). Being able to develop my master thesis on the laboratories of TU/e has been very enriching and it is an experience I will never forget. Moreover, I would like to thank my co-supervisor from Technical University of Catalonia, Salvatore Spadaro, you have been a crucial part to make this possible, you were the first I contacted and the one that introduced me to Nicola and his research group, without you any of this would have been possible. Your help not just in the beginning but during and until the very last step, the different meetings we had, you were always there helping. Without a doubt you are a great professional but also a fantastic human being. Also, I have to give credit to the PhD student from TU/e, Bitao Pan. You were the want that was always in the lab open to any questions, doubts, concerns or to give a hand when I needed. Unfortunately, someone tried to break all your work from years but I am sure that you will overcome this situation and will look at the bright side, I wish you all the best and I hope to see you again in the future.

On the other hand, in this period of a global pandemic being outside my home was not an easy journey, thanks to the strength and love from all my family, friends and my partner I have been able to enjoy every piece of time I spent in the Netherlands. This experience has enriched me a lot professionally but also personally, I have met people I will always carry in my heart whatever my future holds...
# Contents

**Abstract** iii  
**Acknowledgements** v  
**1 Introduction** 1  
**2 State-of-the-art** 3  
2.1 5G-based networks ........................................... 3  
2.1.1 Mobile Edge Computing .................................... 3  
2.1.2 Virtual Radio Access Networks ............................ 5  
2.2 Deep Reinforcement Learning ................................. 6  
2.2.1 Markov Decision Processes ................................. 7  
2.2.1.1 Markov Process .......................................... 7  
2.2.1.2 Markov Reward Process ................................. 7  
2.2.1.3 Markov Decision Process ............................... 8  
2.2.1.4 Bellman Expectation Equation ......................... 8  
2.2.1.5 Optimal Policy ........................................... 10  
2.2.1.6 Bellman Optimal Equation ............................. 10  
2.2.2 Reinforcement Learning ................................. 11  
2.2.2.1 From MDPs to Reinforcement Learning ................ 11  
2.2.3 Deep Q-learning ........................................... 13  
2.2.3.1 Advanced Deep Q-learning model ..................... 14  
**3 Related Work** 17  
**4 System description and assumptions** 25  
4.1 System Model .................................................. 25  
4.2 Solution approach ............................................ 26  
**5 Implemented Solution** 29  
5.1 Reinforcement Learning Framework .......................... 29  
5.1.1 State space .................................................. 29  
5.1.2 Action space ............................................... 31  
5.1.3 Reward function ........................................... 31  
5.2 Algorithm definition .......................................... 32  
5.2.1 Scenario initialisation ..................................... 32  
5.2.2 Metrics collection ......................................... 32  
5.2.2.1 Data collection .......................................... 33  
5.2.2.2 Topology collection .................................... 33  
5.2.3 Metrics evaluation ........................................ 33  
5.2.3.1 State ...................................................... 33  
5.2.3.2 Reward ................................................... 33
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.4</td>
<td>Deep Learning Agent</td>
<td>33</td>
</tr>
<tr>
<td>5.2.4.1</td>
<td>Neural Network</td>
<td>34</td>
</tr>
<tr>
<td>5.2.4.2</td>
<td>Decision Making</td>
<td>34</td>
</tr>
<tr>
<td>5.2.4.3</td>
<td>Action Implementation</td>
<td>34</td>
</tr>
<tr>
<td>5.2.5</td>
<td>Iterative process</td>
<td>34</td>
</tr>
<tr>
<td>5.3</td>
<td>Traffic and stress scripts</td>
<td>34</td>
</tr>
<tr>
<td>6</td>
<td>Performance Evaluation</td>
<td>37</td>
</tr>
<tr>
<td>6.1</td>
<td>Fine-tuning parameters</td>
<td>37</td>
</tr>
<tr>
<td>6.2</td>
<td>Results</td>
<td>38</td>
</tr>
<tr>
<td>7</td>
<td>Conclusions and Future Work</td>
<td>43</td>
</tr>
</tbody>
</table>

Bibliography 45
List of Figures

1.1 Representation of the approach referred to this project. ........................................ 1
2.1 Architecture of MEC systems [8]. ................................................................. 5
2.2 Backup diagrams ............................................................................................. 9
2.3 2 step look-ahead diagram. ............................................................................... 9
2.4 Reinforcement Learning scheme. ...................................................................... 12
2.5 Deep Q-learning. ............................................................................................. 14
3.1 Structure of single-user MEC system [21]. .................................................... 17
3.2 Illustration of resource management for network slicing [13]. ................. 18
3.3 System architecture of TIDE [22]. ................................................................. 19
3.4 Illustration of a multimedia multi-service QoS optimization for resource allocation for MEC system [23]. ........................................... 20
3.5 Framework for resource management with RL-related information [11]. 21
3.6 Wireless powered MEC network [24]. ........................................................... 21
3.7 Different approaches for Deep-NFVOrch ...................................................... 23
4.1 System model defined ...................................................................................... 25
4.2 Scaled solution for virtual radio access networks ........................................... 26
4.3 Virtual RAN schematic by nodes. ................................................................. 27
5.1 Example of blocking wavelength for a specific link. .................................... 30
5.2 Implemented solution with all the steps taken by the algorithm. ............ 35
6.1 Topologies most used for the agent during training. .................................. 39
6.2 Reward by timestep for both tests. .............................................................. 40
6.3 Bandwidth received for each of the nodes during the 2nd test. ............... 41
6.4 Number of errors per second during 2nd test. .......................................... 42
List of Tables

4.1 Available wavelengths for connections between nodes. . . . . . . . . . . 27
5.1 Matrix illustration of the topology of the network that defines the state. 29
5.2 Matrix illustration of possible values for all the different metrics used
to define the state. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
5.3 State space matrix for the reinforcement algorithm. . . . . . . . . . . 31
6.1 Parameters fine-tuned for the tests launched. . . . . . . . . . . . . . . . . . . 38
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAN</td>
<td>Radio Access Network</td>
</tr>
<tr>
<td>BS</td>
<td>Base Station</td>
</tr>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>SDN</td>
<td>Software Defined Network</td>
</tr>
<tr>
<td>RAN</td>
<td>Radio Access Network</td>
</tr>
<tr>
<td>VRAN</td>
<td>Virtual Radio Access Network</td>
</tr>
<tr>
<td>RAT</td>
<td>Radio Access Technology</td>
</tr>
<tr>
<td>NFV</td>
<td>Network Function Virtualization</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>3GPP</td>
<td>Third-Generation Partnership Project</td>
</tr>
<tr>
<td>vNF-SC</td>
<td>virtual Network Function Service Chaining</td>
</tr>
<tr>
<td>DCI-EON</td>
<td>Data Center Interconnection Elastic Optical Network</td>
</tr>
<tr>
<td>RL</td>
<td>Reinforcement Learning</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>DRL</td>
<td>Deep Reinforcement Learning</td>
</tr>
<tr>
<td>MDP</td>
<td>Markov Decision Process</td>
</tr>
<tr>
<td>TD</td>
<td>Temporal Difference</td>
</tr>
<tr>
<td>DQL</td>
<td>Deep Q-Learning</td>
</tr>
<tr>
<td>DQN</td>
<td>Deep Q Network</td>
</tr>
<tr>
<td>NN</td>
<td>Neural Network</td>
</tr>
<tr>
<td>DNN</td>
<td>Deep Neural Network</td>
</tr>
<tr>
<td>DDQL</td>
<td>Double Deep Q-Learning</td>
</tr>
<tr>
<td>DDQN</td>
<td>Double Deep Q-learning Network</td>
</tr>
<tr>
<td>MEC</td>
<td>Mobile Edge Computing</td>
</tr>
<tr>
<td>SE</td>
<td>Spectrum Efficiency</td>
</tr>
<tr>
<td>QoE</td>
<td>Quality of Experience</td>
</tr>
<tr>
<td>SFC</td>
<td>Service Function Chain</td>
</tr>
<tr>
<td>CPU</td>
<td>Computation Processing Unit</td>
</tr>
<tr>
<td>UE</td>
<td>User Equipment</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>WD</td>
<td>Wireless Device</td>
</tr>
<tr>
<td>SOA</td>
<td>Semiconductor Optical Amplifier</td>
</tr>
<tr>
<td>ROADM</td>
<td>Reconfigurable Optical Add-Drop Multiplexer</td>
</tr>
<tr>
<td>PON</td>
<td>Passive Optical Network</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Future networks are expected to satisfy a fast growth of mobile data demand on the next years according to [1]. Nowadays, mobile users are highly increasing in number not only because the fact the personal devices are rising, but also for the appearance of new devices such the ones from Internet of Things (IoT). Moreover, the arrival of the next generation of mobile networks along with the growing proliferation of data generation and consumption at the network edge, has led to a massive use of edge computing. Edge computing can be defined as any technique that allows computation and storage to be executed at the network end [2]. The network architecture of edge computing is a hierarchical model with each layer offering different combinations of computing/networking resources and performance trade-offs [3].

![Figure 1.1: Representation of the approach referred to this project.](image)

The main reason on the introduction of edge computing comes from the concept of cloud computing. Cloud computing has introduced new ways to store and process data, create and offer services, and operate complex systems. In particular, mobile edge computing (MEC) is the network architecture concept that enables cloud computing capabilities at the edge of the cellular network. The main idea behind MEC is that by running applications and performing tasks closer to the mobile customer, network congestion is reduced, and applications have a better performance. This technology is implemented to run at the cellular base stations or other edge nodes to enable flexible deployment and services for customers. Moreover, MEC also allows cellular operators to open their radio access network (RAN) to authorize third parties implement their functionalities. Actually, the RAN edge offers a service environment with ultra-low latency and high bandwidth as well as direct access to real-time network information [4]. The mobile subscriber’s experience can be enriched through efficient network and service operations, based on insight into the radio and network conditions.
Chapter 1. Introduction

This thesis aims to demonstrate how a machine learning algorithm is able to optimize and off-load tasks in a scenario of a virtual radio access network along with mobile edge computing servers at each of the RANs. On next chapters, details on how we approach (Fig. 1.1) to our solution are presented. First, there is a theoretical explanation on how 5G networks are moving towards mobile edge computing (MEC) servers and virtual radio access networks (Virtual RAN). Then, an introduction to deep reinforcement learning algorithms, allow a better understanding on how the thesis approaches to the final solution. Finally, conclusions and potential future work are presented.
Chapter 2

State-of-the-art

This chapter is an explanation on the state-of-the-art techniques that are going to be used during this thesis. First, a brief introduction to 5G-based networks is necessarily to understand the benefits and challenges that will arise from moving computational resources to the edge and virtualizing the network. Then, there is an introduction to deep reinforcement learning that covers from its basic concept to the advanced model used for the final solution.

2.1 5G-based networks

The arrival of the next generation of mobile networks has rendered the demands of more evolved and scalable technologies as explained in [5–7]. According to Third-Generation Partnership Project (3GPP) and the IEEE Computer Society, network slicing, software defined networks (SDN), network function virtualization (NFV), Internet of things (IoT), Cloud technologies, machine learning (ML), and Big Data are among the most important technologies to achieve the requirements of 5G, like the increasing demands from the end users. The new "5G Networks" are expected to be flexible, scalable, and cost-efficient thus improving the network capacity with the limited available resources while confronting the geographical and temporal variation of traffic demand. Techniques implemented for the new 5G networks will tackle the problem of cloud computing and to address those issues mobile edge computing, elastic optical networks, virtual radio access networks will play a key role. Detail on these techniques is included in this on the following chapter.

In order to accomplish the requirements for the new 5G networks this thesis presents a solution that exploits benefits from different technologies by taking computational resources to the edge and from the virtualization of the radio access networks. Below, there is introduced the concept of mobile edge computing and virtual radio access networks.

2.1.1 Mobile Edge Computing

Edge computing is an exciting new approach to network architecture that helps organizations break beyond the limitations imposed by traditional cloud-based networks. Traditional cloud computing networks are highly centralized, with data being gathered on the outermost edges and transmitted back to the main servers for processing. However, all of that has changed and companies are optimizing their networks and relocating more processing function closer to where data is gathered at the network edge, where it can be analyzed and applied in real time much closer to the intended users. The speed and flexibility afforded by this approach to handling data creates an exciting range of benefits for organizations which are summarized on the following benefits:
• **Speed:** the ability to increase the network performance by reducing latency is the core business for many companies; from healthcare institutions to businesses that provide data-driven services to customers. Being able to process data in nearby edge data centers means that information does not have to travel nearly as far as it would under a traditional cloud architecture.

• **Security:** edge computing distributes processing, storage, and applications across a wide range of devices and data centers, which makes it difficult for any single disruption to take down the network. It also reduces the amount of data actually at risk because there is less data to be intercepted during transit.

• **Scalability:** by combining collocation services with regional edge computing data centers, organizations can expand their edge network reach quickly and cost-effectively. The flexibility of not having to rely upon a centralized infrastructure allow them to adapt quickly to evolving markets and scale their data and computing needs more efficiently.

• **Versatility:** the scalability of edge computing also makes it very versatile. Edge computing devices are always on, and always generating data for future analysis. The unstructured information gathered by edge networks can either be processed locally to deliver quick services or delivered back to the core of the network where powerful analytics and machine learning programs will dissect it to identify trends and notable data points.

• **Reliability:** given the security advantages provided by edge computing, it should not come as surprise that it offers better reliability as well. With edge data centers positioned closed to end users, there is less chance of a network problem in a distant location affecting local customers. Since so many edge computing devices and edge data centers are connected to the network, it becomes more difficult for any failure to shut down service entirely.

Therefore, the step from edge computing to mobile edge computing relates on the addition of mobile devices to edge computing networks. As it stands, MEC is a network architecture that enables cloud computing capabilities and an IT services environment at the edge of the cellular network and a key emerging technology of 5G together with Network Functions Virtualization and Software Defined networking [8]. Thanks to the advantages mentioned previously and applying them for mobile services a wide range of applications, such as gaming, augmented reality apps, 3D modelling, social networking, health monitoring, surveillance networks, or connected vehicles take benefit to improve its services.

As it can be seen in Figure 2.1 edge devices include all type of devices (both mobile phones and IoT devices) connected to the network. Edge cloud is the less resourceful cloud deployed in each of the mobile base station which also has the responsibility of traditional network traffic control (both forwarding and filtering) and hosting various mobile edge applications. Finally, public cloud is the cloud infrastructure hosted in the Internet.

The key element of MEC is the IT application server which is integrated at the RAN element. This server provides computing resources, storage capacity, connectivity, and access to user traffic and radio and network information [9]. And as we talk about radio access networks a new approach to virtualize the network to improve is presented in the following section.
2.1.2 Virtual Radio Access Networks

Nowadays, network operators are using networking technologies such as SDN and NFV to virtualize their RAN to meet the growing demands that require scalability, increased management efficiencies, and cost-effective operations. Virtualization can be applied to different aspects of the RAN, through spectrum virtualization, hardware sharing, virtualization of multiple radio access technologies (RATs), and virtualization of computing resources [10]. Spectrum virtualization allows the available spectrum to be utilized more efficiently by permitting multiple network operators to share the same spectrum. Hardware and network sharing is of particular relevance for small cells in order to avoid massive over-provisioning. Virtualization of multiple RATs allows simplified management of different RATs, each dedicated to different services and offering a different quality of service (QoS). Virtualization of computing resources is a new option that builds upon the idea of co-locating the processing resources of multiple BSs at a central processing center.

Actually, the approach of operators used in Radio Access Networks (RANs), considering only busy hours, is no longer acceptable; temporal and geographical variations of traffic, in addition with the lack of network capacity put them in a complex situation. On this topic, many researchers have already developed new ideas and concepts for RANs. They meet all the existing requirements integrating Software Defined Network (SDN) and Network Function Virtualization (NFV) in mobile networking.

To extend the benefits of virtualization to radio access networks (RANs) many companies are working to leverage RAN architectures like virtual RAN in preparation for the 5G and IoT era. Virtual RAN applies the principles of Network Functions Virtualization (NFV) by virtualizing network functions, providing a greater degree of flexibility in the RAN in return. A Virtual RAN consists of a centralized pool of baseband units (BBUs), virtualized RAN control functions and service delivery optimization. With a Virtual RAN, baseband units are moved away from the base station and to a datacenter. As a result, functions of the BBUs can be implemented with virtual machines in a centralized data center. This provides intelligent scaling of computing resources, while decreasing energy consumption and capital expenditure (CAPEX).

The virtualization of the RAN is expected to help carriers prepare for 5G networks, which further increase bandwidth requirements. For example, a 5G virtual
base station can improve system capacity and spectral efficiency by drawing from a pool of BBUs that share signaling among cells. A virtual RAN can also help simplify the deployment of novel features and algorithms, which optimize resource usage and enhance the end-user experience.

According to Ericsson, virtualized RANs (VRANs) arise a good number of benefits. First, a fully virtualized RAN implies that there is just one single uniform hardware platform across the core network, RAN and edge. This simplifies the management of the complete network, reducing operations and maintenance costs. Secondly, the network functions are separated from the processing hardware which means RAN network functions from multiple vendors could run on the same hardware, benefiting the flexibility for the service provider. Also, VRAN represent an opportunity to embrace actual solutions in cloud technologies, for non-RAN-specific functions. Another benefit, a VRAN holds the promise of increased flexibility as functionality and capacity could be more easily deployed where and when required. Lastly, a widely adopted open platform could also lower barriers for cross-domain innovation, facilitating the development of new use cases and services.

Moreover, the centralized architecture of a 5G virtual base station enables control plane and data plane splitting to be easily implemented. Additionally, by decoupling network functions from proprietary hardware, a virtual RAN can enable a level of adaptability in networks that operators need for the commercialization of 5G.

2.2 Deep Reinforcement Learning

Deep reinforcement learning is the technique used to solve the problem of optimization and task offloading in this thesis. DRL learns from experience which makes it suitable for the solution of our use case, we do not hold any dataset with relevant information of our network or information from nodes, and we are also not able to obtain this kind of dataset. Therefore, supervised or unsupervised learning solutions are not feasible and reinforcement learning becomes the only option for approaching our solution. However, pure reinforcement learning has some limitations when it comes to large environments or complex solutions. Thanks to a neural network within the agent in a reinforcement learning setup it is possible to estimate the solution which leads to a deep reinforcement learning approach. Details on how DRL performs will be given in the following sections, but summarising one could define DRL as an agent in charge of the decision making according to two different incomes. Those parameters feed the agent, according to the new state and reward after the last action performed by the agent. Different actions performed by the agent generate a reward and a new state from the environment that feed the neural network of the agent that decides the following action. This process iterates until the agent learns the maximum reward for all the different actions that, in our case, seeks for the optimization and offloading of tasks within the network nodes from the setup explained at Chapter 4.

In this section, it is introduced fundamental knowledge for the understanding of the final solution of the thesis. Firstly, an explanation about Markov decisions processes will take us on how reinforcement learning (RL) algorithms take advantage of complex problems. RL is an important branch of machine learning and usually focuses on solving control problems with delayed rewards over time [11]. Different from traditional machine learning algorithms that require large historical datasets, RL can start from scratch and gradually achieve human-level decision
2.2. Deep Reinforcement Learning

Later, we will discuss about deep reinforcement learning (DRL) which is a branch of reinforcement learning with capabilities of deep learning to improve its efficiency and performance in terms of convergence to the final solution. Finally, the advanced DRL model used in the final solution of the project is introduced.

2.2.1 Markov Decision Processes

Markov Decision Processes (MDP) is a mathematical framework to describe a fully observable environment where the outputs are half random and half under control of the agent, which is the decision maker. A theoretical overview about MDPs is given step by step in the following subsections.

2.2.1.1 Markov Process

First thing we should do is to define a Markov State (a state is a random variable): a state \( S_t \) is Markov if and only if:

\[
P(S_{t+1}|S_t) = P(S_{t+1}|S_1, ..., S_t)
\]  

(2.1)

A Markov state contains all the useful information about the past, so that the probability of the next state is the same conditioning on the current state or on the entire state history. In a Markov state \( s \), we can define the transition probability to a successor state \( s' \) as

\[
P_{SS'} = P(S_{t+1} = s'|S_t = s)
\]

(2.2)

We can then use a matrix for the probabilities of all possible successor states; we call this matrix the transition probability matrix.

We can now define what is a Markov Process: a sequence of random states with Markov property. More formally, it is a tuple \((S, P)\), where \( S \) is a finite set of states and \( P \) is a transition probability matrix.

2.2.1.2 Markov Reward Process

A Markov Reward Process adds the notion of reward, a scalar signal \( R_t \) that the agent receives for each state is visits. Formally, a Markov Reward Process is a tuple \((S, P, R, \gamma)\) where \( R \) is a reward function and \( \gamma \in [0, 1] \) is a discount factor.

We are now able to define the notion of return: the return \( G_t \) is the total discounted reward from time-step \( t \), in formula:

\[
G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}
\]

(2.3)

The discount factor allows to evaluate the present value of future rewards. If \( \gamma \) is close to 0 then the process weights more near-future than far-future rewards. If \( \gamma \) is close to 1 the process is more far-sighted: the special case \( \gamma = 1 \) is called undiscounted case, and some hypothesis must hold on the process to avoid infinite returns.

Starting the notion of return we can define the state value function: a state value function \( v(s) \) is the expected return starting from state \( s \)

\[
v(s) = E(G_t|S_t = s)
\]

(2.4)
It can be shown that this function can be decomposed in 2 components, and immediate reward and a discounted value of the successor state:

\[ v(s) = \mathbb{E}(R_{t+1} + \gamma v(S_{t+1} | S_t = s)) = R_s + \gamma \sum_{s' \in S} P_{ss'} v(s') \] (2.5)

Writing this in matricial form, we see that this is a linear equation, and can be solved to find \( v \):

\[ v = R + \gamma P v \] (2.6)

\[ v = (I - \gamma P)^{-1} R \] (2.7)

Solving this linear problem with a direct method is not usually convenient with respect to computational complexity, so iterative methods are preferred instead.

### 2.2.1.3 Markov Decision Process

We can finally give the definition of a Markov Decision Process: a Markov Decision Process (MDP) is a tuple \((S, A, P, R, \gamma)\), where \(A\) is a finite set of actions and \(P\) and \(R\) depend now on the action chosen at time \(t\):

\[ P^a = P(S_{t+1} = s' | S_t = s, A_t = a) \] (2.8)

\[ R^a = \mathbb{E}(R_{t+1} | S_t = s, A_t = a) \] (2.9)

The action taken from the agent in a state \(s\) at a time \(t\) is stochastic, and we can define its probability. A policy is a probability distribution on actions conditioning on states:

\[ \pi(a|s) = P(A_t = a | S_t = s), \] (2.10)

so in any given state \(s\) we are asking which is the probability the agent will take the action \(a\).

This leads to the key definition of value function for a MDP:

- a state-value function is the expected return starting from state \(s\) and following policy \(\pi\) thereafter:
  \[ v_\pi(s) = \mathbb{E}_\pi(G_t | S_t = s) \] (2.11)

- an action-value function is the expected return starting from state \(s\), taking action \(a\) and following policy \(\pi\) thereafter:
  \[ q_\pi(s, a) = \mathbb{E}_\pi(G_t | S_t = s, A_t = a) \] (2.12)

### 2.2.1.4 Bellman Expectation Equation

The state-value function can be decomposed as we did in Eq. (2.5).

\[ v_\pi(s) = \mathbb{E}_\pi(R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s) \] (2.13)

and similarly for the action-value function:

\[ q_\pi(s, a) = \mathbb{E}_\pi(R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a) \] (2.14)
2.2. Deep Reinforcement Learning

Using so called backup diagrams from Figure 2.2, we can derive additional formulations; suppose the agent is in a state $s$ and performs a look-ahead to evaluate the current state like in diagram 2.2a. From state $s$ different actions are possible, and the probability of each of them is given by the policy function, so we can write

$$v_\pi(s) = \sum_{a \in A} \pi(a|s)q_\pi(s, a)$$

(2.15)

Suppose now the agent has already taken the action $a$, and we want to evaluate what happens after that (which depends on the environment) as in the diagram 2.2b.

![State-Action Diagrams](image)

We have an immediate reward $R^s_a$ for having taken action $a$ from state $s$ and then the future value depends on the state the agent arrives following the dynamics of the environment:

$$q_\pi(s, a) = R^s_a + \gamma \sum_{s' \in S} P^a_{ss'} v_\pi(s')$$

(2.16)

Now if you put together the previous two diagrams, we obtain the situation described in the diagram 2.2c from which immediately derives the formula:

$$v_\pi(s) = \sum_{a \in A} \pi(a|s)(R^s_a + \gamma \sum_{s' \in S} P^a_{ss'} v_\pi(s'))$$

(2.17)

If we instead start from the agent already having taken the action $a$ and perform a 2 step look-ahead, we have the following diagram 2.3. From which follow the formula:

$$q_\pi(s, a) = R^s_a + \gamma \sum_{s' \in S} P^a_{ss'} \sum_{a' \in A} \pi(a'|s')q_\pi(s', a')$$

(2.18)

The last two equations (2.17) and (2.18) show an iterative relationship for $v_\pi$ and $q_\pi$ with themselves.

![2 step look-ahead diagram](image)
2.2.1.5 Optimal Policy

The value function tells us how good is for an agent to be in a specific state or to make a specific action. The goal we would be interested in however is finding the best possible policy among the possible ones. This is defined as the optimal value function for a MDP:

- an optimal state-value function is the maximum state-value function over all policies:
  \[ v^*(s) = \max_\pi v_\pi(s) \]  
  (2.19)

- an optimal action-value function is the maximum action-value function over all policies:
  \[ q^*(s, a) = \max_\pi q_\pi(s, a) \]  
  (2.20)

Once we know the optimal action-value function, the optimal policy \( \pi^* \) derives from simply taking the action that corresponds to the maximum value of the optimal action-value function in a given state \( s \):

\[ \pi^*(s|a) = \begin{cases} 
1 & \text{if } a = \arg\max_{a \in A} q^*(s, a) \\
0 & \text{otherwise}
\end{cases} \]  
(2.21)

2.2.1.6 Bellman Optimal Equation

The following relationship holds for optimal state-value and optimal action-value functions:

\[ v^*(s) = \max_{a \in A} q^*(s, a) \]  
(2.22)

In equation (2.15) we were averaging the action-value function over the possible actions using the policy as probability distribution: in this case, given that the agent follows the optimal policy, we are taking the max of the action-value function over possible actions. Equation (2.16) instead does not change, as the behaviour of the environment is not under the agent control:

\[ q^*(s, a) = R_s + \gamma \sum_{s' \in S} P_{ss'} v^*(s') \]  
(2.23)

Putting together these 2 formulas, we obtain:

\[ v^*(s) = \max_{a \in A} R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} v^*(s') \]  
(2.24)

Starting from the expression (2.18) instead, in which the agent has already taken the action \( a \), we have that

\[ q^*(s, a) = R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} \max_{a' \in A} q^*(s', a') \]  
(2.25)

where in the right addendum we replace the average of the action-value function over possible actions with a maximum. Equations (2.24) and (2.25) are the starting point to find the optimal policy; they are non-linear equations and don’t allow in general for closed form solutions, so iterative methods are needed.
2.2. Deep Reinforcement Learning

2.2.2 Reinforcement Learning

Reinforcement Learning, is an important branch of machine learning, is an effective tool and widely used in the literature to address MDPs.

Unlike MDP, Reinforcement Learning (RL) algorithms attempt to derive optimal policies without explicit model of the environment’s dynamics as explained in [12]. In this case, the underlying transmission probability $p(s_{t+1}|s_t, a_t)$ is unknown and even non-stationary. Thus, the RL agent will learn from actual interactions with the environment and adapting its behaviour upon experiencing the outcome of its actions, so as to maximize the expected discounted rewards. Similarly in [13], RL learns how to interact with the environment to achieve maximum cumulative return and it has been applied to fields like robotics, self-driving and video games for years now.

2.2.2.1 From MDPs to Reinforcement Learning

Mathematically, RL follows the concept of Markov Decision Process, although while MDP is a generalized framework for modelling decision-making problems in cases where the result is partially random and affected by the applied decision. As seen previously in section 2.2.1.3 an MDP can be formulated by a 5-tuple as $(S, A, P, R, \gamma)$ where $S$ and $A$ denote a finite state space and action set respectively. $P$ indicated the probability that the action $a \in A$ under state $s \in S$ at slot $t$ to state $s' \in S$ at slot $t+1$. $R$ is an immediate reward after performing the action $a$ under state $s$, while $\gamma \in [0, 1]$ is a discount factor to reflect the diminishing importance of current reward on future ones. Usually, the goal of an MDP is to find a policy $a = \pi(s)$ that determines the selected action $a$ under state $s$, so as to maximize the value function, which is typically defined as the expected discounted cumulative reward by the Bellman equation (2.24).

What RL intends is to obtain this optimal policy $\pi_s$ under unknown circumstances and partially random dynamics. RL does not have knowledge of whether it has converged to its goal, it needs to balance between exploiting and exploring. These two concepts are key for RL algorithms, when exploiting an action, we are weighting more those ones that obtained more reward on previous iterations while exploration allows the environment to try new random actions independently of its past reward. The aim of the algorithm is to find the best equilibrium between exploring new potential actions and exploiting the already learnt experience. According to [13] we can classify RL algorithms according to different criteria:

- **Model-based vs Model-free**: in model-based algorithms the agent tries to learn the model of how the environment works from the observations and then plan a solution for the model. Once the model is more or less accurate, it uses a planning algorithm with this learned model. However, model-free algorithms do not learn how to model the environment. Instead, the agent estimates the value function and derives the optimal policy by choosing the action yielding the largest value in the current state. A good example of model-free algorithm is Q-learning which is explained in section 2.2.2.2.

- **Monte-Carlo vs Temporal-Difference Update**: for Monte-Carlo update the agent updates its estimation for a state-action pair by calculating the mean return from a collection of episodes. A TD update approximates the estimation by comparing estimates at two consecutive episodes.
Chapter 2. State-of-the-art

- **On-policy vs Off-policy**: before updating the value function, the agent also needs to sample and learn the environment by performing some non-optimal policy. If the update is irrelevant to the sampling policy, the agent is called to perform an off-policy update. On-policy algorithms are when the agent updates the value function by taking into account the sampling policy.

Basically, in a reinforcement learning process the agent first observes its current state, and then takes an action, and receives its immediate reward together with its new state as illustrated in Figure 2.4.

2.2.2.2 Q-learning

In RL, Q-learning is the most effective method and widely used in the literature. However, as explained in [14] there are different extensions of Q-learning for advanced MDP models. The basic Q-learning, SARSA and Q-learning for Markov games. In this thesis, we will focus on the basic Q-learning explained below.

Back to an MDP, we aim to find an optimal policy \( \pi^* : \mathcal{S} \rightarrow \mathcal{A} \) for the agent to minimize the overall cost for the system. Accordingly, we first define value function \( V^\pi(s) : \mathcal{S} \rightarrow \mathbb{R} \) that represents the expected value obtained by following policy \( \pi \) from each state \( s \in \mathcal{S} \). The value function \( V^\pi \) for policy \( \pi \) quantifies the goodness of the policy through an infinite horizon and discounted MDP that can be expressed as follows:

\[
V^\pi(s) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) | s_0 = s \right] = \mathbb{E}_\pi \left[ r_t(s_t, a_t) + \gamma V^\pi(s_{t+1}) | s_0 = s \right] \quad (2.26)
\]

Since we aim to find the optimal policy \( \pi^* \), an optimal action at each state can be found through the optimal value function expressed by \( V^*(s) = \max_a \{ \mathbb{E}_\pi[r_t(s_t, a_t) + \gamma V^\pi(s_{t+1})] \} \).

If we denote \( Q^*(s, a) \triangleq r_t(s_t, a_t) + \gamma \mathbb{E}_\pi[V^\pi(s_{t+1})] \) as the optimal Q-function for all state-action pairs, then the optimal value function can be written by \( V^*(s) = \max_a \{ Q^*(s, a) \} \). Now, the problem is reduced to find optimal values of Q-function, i.e., \( Q^*(s, a) \), for all state-action pairs, and this can be done through iterative processes. In particular, the Q-function is updated to the following rule:

\[
Q_{t+1}(s, a) = Q_t(s, a) + \alpha_t \left[ r_t(s, a) + \gamma \max_{a'} Q_t(s, a') - Q_t(s, a) \right] \quad (2.27)
\]
2.2. Deep Reinforcement Learning

The core idea behind this update is to find the Temporal Difference (TD) between the predicted Q-value, i.e., \( r_t(s,a) + \gamma \max_{a'} Q_t(s',a') \) and its current value, i.e., \( Q_t(s,a) \). In the equation (2.27), \( \alpha_t \) is the learning rate used to determine the impact of new information to the existing Q-value. The learning rate can be chosen constant or it can be adjusted dynamically during the learning process. However, it must satisfy Assumption (1) to guarantee the convergence for the Q-learning algorithm.

**Assumption 1.** The step size \( \alpha_t \) is deterministic, nonnegative and satisfies the following conditions: \( \alpha_t \in [0, 1] \), \( \sum_{t=0}^{\infty} \alpha_t = \infty \), and \( \sum_{t=0}^{\infty} (\alpha_t)^2 < \infty \)

The step size adaptation \( \alpha_t = 1/t \) is one of the most common examples used in RL. More discussions for selecting appropriate step size can be found in [15].

Once either all Q-values converge or a certain amount of iterations is reached, the algorithm terminates. The algorithm then yields the optimal policy indicating an action to be taken at each state such that \( Q^*(s,a) \) is maximized for all states in the state space, i.e., \( \pi^*(s) = \arg\max_a Q^*(s,a) \). Under the assumption of the step size (i.e., Assumption 1), it is proved in [16] that the Q-learning algorithm converges to the optimum action-values with probability one.

2.2.2.2.1 \( \epsilon \)-greedy policy

Is an approach to train the Q-network by exploring random action with probability \( \epsilon \) and exploiting an action that maximizes the expected long-term reward with probability \( 1 - \epsilon \).

2.2.3 Deep Q-learning

The Q-learning is powerful and simple, its main weakness is the lack of generality and scalability. That is, as the states rise, the Q-table may exponentially increase, which makes Q-learning inapplicable in large-scale networks. To tackle this issue, the deep Q network (DQN) introduces a neural network to estimate the Q-value function. Thanks to the neural networks, DQN can be applied to the large-scale problem. It is noticeable that neither Q-learning nor DQN can cope with continuous problems. Hence, deep deterministic policy gradient (DDPG), as the representative advanced DRL technique, employs the actor-critic model and is capable of dealing with continuous control problems like workload balancing, task offloading, resource allocation, traffic optimization, and energy scheduling [11].

The Q-learning algorithm can efficiently obtain an optimal policy when the state space and action space are small. However, in practice, with complicated system models, these spaces are usually large. As a result, the Q-learning algorithm may not be able to find the optimal policy. Thus, Deep Q-learning (DQL) algorithm is introduced to overcome this shortcoming. Intuitively, the DQL algorithm implements a Deep Q-Network (DQN), i.e., a Deep Neural Network (DNN), instead of the Q-table to derive an approximate value of \( Q^*(s,a) \) as shown in Figure 2.5.

As stated in [17], the average reward obtained by reinforcement learning algorithms may not be stable or even divergent when a nonlinear function approximation is used. This stems from the fact that a small change of Q-value may greatly affect the policy. Thus, the data distribution and the correlations between the Q-values and the target values \( R + \gamma \max_{a'} Q(s',a') \) are varied. To address this issue, two mechanisms, i.e., experience replay and target Q-network, can be used.
• **Experience replay mechanism**: the algorithm initializes a replay memory D, with information of the transitions \((s_t, a_t, r_t, s_{t+1})\). i.e., experiences, generated randomly, e.g., using \(\epsilon\)-greedy policy. Then, the algorithm selects samples, i.e., minibatches of transitions from D to train the DNN. The Q-values obtained by the trained DNN will be used to obtain new experiences, i.e., transition, and these experiences will be then stored in the memory pool D. This mechanism allows the DNN trained more efficiently by using the experience replay, the transitions are more independent and identically distributed, and this the correlation s between observation can be removed.

• **Fixed Target Q-network**: in the training process, the Q-value will be shifted. Thus, the value estimations can be out of control if a constantly shifting set of values is used to update the Q-network. This leads to the destabilization of the algorithm. To address this issue, the target Q-network is used to update frequently but slowly the primary Q-networks’ values. In this way, the correlations between the target and estimated Q-values are significantly reduced, thereby stabilizing the algorithm.

The DQL algorithm with experience replay and fixed target Q-network is presented in this thesis. DQL inherits and promotes advantages of both reinforcement and deep learning techniques, and it has a wide range of applications in practice such as game development, transportation, and robotics.

### 2.2.3.1 Advanced Deep Q-learning model

There are many different models that take profit of the advantages from Q-learning and neural networks. In this section, one of them is presented and specifically, Double Deep Q-learning (DDQN) is explained in detail since is the one used for the solution of this thesis.

#### 2.2.3.1.1 Double Deep Q-learning

In some stochastic environments, the Q-learning performs poorly due to the large over-estimations of action values [18]. These over-estimations result from a positive bias that is introduced because Q-learning uses the maximum action value as an approximation for the maximum expected action value as shown in Eq. (2.27). The reason is that the same samples are used to decide which action is the best, i.e., with highest expected reward, and the same samples are also used to estimate that action-value. Thus, to overcome the over-estimation problem of the Q-learning algorithm, the authors in [19] introduce a solution using two Q-value function, i.e., \(Q_1\) and
Q_2$, to simultaneously select and evaluate action values through the loss function as follows:

$$r_j + \gamma Q_2(s_{j+1}, \arg\max_{a_j+1} Q_1(s_{j+1}, a_j+1; \theta_1); \theta_2) - Q_1(s_j, a_j; \theta_1)^2.$$

Note that the selection of an action, in the arg max, is still due to the online weights $\theta_1$. This means that, as in Q-learning, we are still estimating the value of the greedy policy according to the current values, as defined by $\theta_1$. However, the second set of weights $\theta_2$ is used to evaluate fairly the value of this policy. This second set of weights can be updated symmetrically by switching the roles of $\theta_1$ and $\theta_2$. Inspired by this idea, the authors in [19] then develop Double Deep Q-Learning (DDQL) [20] using a Double Deep Q-learning Network (DDQN) with the loss function updated as follows:

$$r_j + \gamma \hat{Q}(s_{j+1}, \arg\max_{a_j+1} Q(s_{j+1}, a_j+1; \theta'); \theta') - Q(s_j, a_j; \theta)^2 \quad (2.28)$$

Unlike double Q-learning, the weights of the second network $\theta_2$ are replaced with the weights of the target networks $\theta'$ for the evaluation of the current greedy policy as shown in Eq. (2.28). The update to the target network stays unchanged from DQN and remains a periodic copy of the online network.
Chapter 3

Related Work

In this chapter it is introduced the state of the art research on Reinforcement Learning techniques applied to different networking scenarios. Actually, reinforcement learning is getting attention of many institutions for, its applications to model real scenarios and optimizing behaviours. Most interesting areas of research are in robotics, video-gaming, business strategy, networks, health-care, chemistry and personalized recommendations. These are a widely variety of scenarios where reinforcement learning can be applied, but this is a topic with a lot of potential and probably in the future more areas of knowledge will start developing their reinforcement learning techniques.

Below, some similar related work is explained. Attention is focused on those examples where RL, or its deep variation (DRL), is applied to networking or similar applications.

Firstly, in [21] they consider a single-user MEC system containing a mobile device and a MEC server. The aim is to resolve a task offloading problem on the mobile device. i.e., minimize the mean slowdown of tasks in the buffer queue and the mean energy consuming of the mobile device. They propose an online DRL method to schedule tasks to execution unit or transmission unit independently.

The structure of the system model is composed by a single-user MEC system, i.e., a mobile device and a MEC server which communicate through wireless transmission channel. The mobile device is mainly composed of a task buffer queue, a task scheduling unit, a transmission unit module, and a local execution unit as shown in Figure 3.1.

From the four aspects mentioned on the system model the algorithm is designed to minimize the mean slowdown of tasks in the queue and energy consumption in mobile device by optimizing task offloading policy, and reducing the energy consumption of mobile devices while speeding up the processing speed of tasks in the

![Figure 3.1: Structure of single-user MEC system [21].](image)
queue. According to section 2.2 it is needed to define which are the state and action space together with the reward function for the model.

The state space consists of the resource occupancy status of local execution unit and transmission unit in mobile device and the resource requirement status of tasks in task buffer queue. Action space is defined as a scheduling task from task queue to the local execution unit or to the transmission unit. Finally, according to the objective of minimizing the mean slowdown and energy consumption, the reward function takes into account the values of all unfinished tasks in a mobile device, tasks waiting to be scheduled and tasks to be processed in execution unit or transmission unit. Moreover, as they use a deep reinforcement learning approximation, they add to the agent a neural network composed by an input layer, two hidden layers and an output layer.

Another article related talks about how Network Slicing is a challenging issue that looks forward to intelligent innovations to make resource management consistent with users’ activities per slice. In [13] they propose a DRL algorithm to solve some typical resource management for network slicing scenarios, which include radio resource slicing and priority-based core network slicing. An illustration of resource management for network slicing can be seen at Figure 3.2.

On one hand, their approach for radio resource slicing tries to give a bandwidth sharing solution to maximize the long-term reward. State space consists on the number of arrived packets in each slice within a specific time window, while the algorithm decides what bandwidth to allocate to each slice according to reward function that weights the sum of Spectrum Efficiency (SE) and Quality of Experience (QoE) satisfaction ratio. On the other hand, they simulate a scenario with 3 service function chains (SFCs) with the same basic capability but working at different computation processing units (CPUs). The reward function that feeds the agent weights the sum
of average time in 3 SFCs, along with the information of the state space which includes the priority and timestamp of last arrived five flows in each service function chain (SFC). Agent action consists on allocate SFC for the flow at the current timestamp.

According to [22], they propose an intelligent network control architecture based on DRL that can dynamically optimize routing strategies in an SDN network. The architecture is called TIDE (Time-relevant Deep Reinforcement Learning control) and it is implemented and validated on a real network environment.

As shown in Figure 3.3, the skeleton is composed by three different planes: data plane, control plane and AI plane.

- control plane: connects the data plane with the AI plane. Through the connection with the data plane, the controller can collect the network status such as the flow table statistics, resource availability, etc. With the AI interface the controller sends the global view of the network state input to the AI plane, and receives the dynamic strategy from the AI plane.

- data plane: where all the routers, hosts and connections of the network can be found. Informs the control plane about all the metrics required and updates the topology when required by the agent of the AI plane.

- AI plane: a smart agent uses the global view of the network and network performance as input, which is used for routing strategy and then evaluates the performance of such policy according to the instant reward collected from the network.

In the following publication [23], a DQN algorithm is adopted to make decisions to allocate radio resource dynamically compared with round-robin and priority-based scheduling algorithms.

In the system model (3.4a), the MEC server can provide various multimedia applications for multiple user equipment’s (UEs). The data packets from multimedia applications are mapped into several QoS flows, according to their different QoS characteristics. Then, the data packets in different QoS flows are processed in the
Chapter 3. Related Work

(A) Multimedia multi-service in MEC system.

(B) RAN resource allocation.

Figure 3.4: Illustration of a multimedia multi-service QoS optimization for resource allocation for MEC system [23].

gNB and delivered to UEs. The paper considers that M UEs can request N different multimedia applications from the MEC server at the network edge and the aim is to adjust resource allocation to optimize the system performance. To do so, they implement a DQN framework with state space representing the system features including the waiting time of the packets in the buffers, the association between packets and QoS flows. Action space, decide how to allocate every resource segment (3.4b) at time $t$. Finally, the reward function evaluates the packet delay and packet loss according to the QoS evaluation value.

At [11], a model-free DRL approach to efficiently manage the resources at the network edge is proposed. The agent implements a mobility-aware data processing service migration management. Inspired by the power and success of RL they propose a solution into edge computing and resource management.

The framework (3.5), consists of three modules: network hyper-visor, RL-based controller and action executor.

- network hyper-visor: responsible for monitoring and aggregating the states of the elements in the edge computing environment.
- RL-based controller: capable of making decisions based on the real-time environment state and its accumulated experience.
- action executor: communicates with the RL-based controller to obtain the control decisions, and execute the derived action.

As explained in the previous sections, the reinforcement learning algorithm needs the state and action space, and the reward function. For their solution, the state is a tuple of the current location of the virtual machine (VM) and the number of users associated to each base station. Action space consists on migrating from one base station to another. Then, a function of the communication cost for the data transmission and VM migration during time slot $t$ defines the reward of their model.

The authors from the publication [24], propose a solution for a wireless powered MEC network with one AP and multiple wireless devices (WDs) as shown in Figure 3.6.
The solution presents a deep reinforcement learning-based online offloading (DROO) framework to maximize the weighted sum of the computation rates of all the WDs. Compared with other existing learning-based methods they introduce novel contributions like 1) the DROO learns from the past offloading experience under various wireless fading conditions, and automatically improves its action generating policy, 2) decomposes the problem into an offloading decision sub-problem and a resource allocation sub-problem, 3) only needs to select from few candidate actions each time, thus is computationally feasible and efficient in large-scale networks with high-dimensional space, and 4) it gradually decreases the number of convex resource allocation sub-problems to be solved in a time frame.

The state space contains information of the wireless channel gains while the action space is an indicator variable which decides if computation task is offloaded or computed locally. Related work similar to the articles summarised above can also be found at [2, 25–27].
Similarly, deep reinforcement learning techniques are being applied at elastic optical network (EONs) which have been widely considered for inter/intra-datacenter (DC) networks solutions. They provide an agile optical layer to adapt to the highly dynamic traffic among DCs.

At [28] a DRL module based on asynchronous advantage actor critic (A3C) used as an observer in Deep-NFVOrch is designed to improve adaptability. When a service cycle (Fig. i.e., including both pre-deployment and provisioning phases) ends, the observer collects instant performance metrics and uses them as the feedback to update length of the next service cycle, the DL-based request predictor, and its own DNN, for getting better performance metrics next time. The DRL-based observer leverages an asynchronous training scheme to ensure superior learning capability for online operations. The DL-based request predictor (DL-Predictor) forecasts future vNF-SC requests in the next $\Delta T_n$ according to historical requests, and then the prediction is sent to the vNF-SC pre-deployment module, which conducts lightpath establishment and vNF instantiation in the IDC-EON accordingly. Next, the system moves to the provisioning phase, which takes most of the time in $\Delta T_n$, and it collects the actual incoming vNF-SC requests to serve. Finally, when the provisioning phase is about to end, the DRL-based observer (DRL-Observer) collects instant performance metrics from the vNF-SC provisioning module and uses them to evaluate the provisioning in this service cycle. DRL-Observer is trained to update its own DNN and $\Delta T$ intelligently based on the performance evaluation. It then forwards the new $\Delta T$ (i.e., $\Delta T_{n+1}$) to DL-Predictor and the system timer and lets the system move to the next service cycle. Their simulations indicate that Deep-NFVOrch can realize adaptive vNF-SC provisioning and simultaneously balance the system performance on resource usage, network reconfiguration overhead, and blocking probability, under highly dynamic vNF-SC requests. They model an inter-datacenter elastic optical network (IDC-EON) as a graph with two sets of DCs and fiber links, respectively.

Authors at [29] tackle the problem of adaptability and cost-effective virtual network function service chaining (vNF-SC) in a datacenter interconnection based on elastic optical network (DCI-EON). Their solution optimizes the previous design of a deep reinforcement learning based adaptive service framework, named Deep-NFVOrch. Specially, Deep-NFVOrch works in service cycles and tries to reduce the setup latency of vNF-SC by invoking request prediction and pre-deployment at the beginning of each service cycle. To address this latency issue, they propose a new service framework that first predicts future vNF-SC requests based on historical information, then deploys the required vNFs and lightpaths in the DCI-EON in advance and thus need to steer the application traffic through the vNFs in sequence only when the requests actually come in. In the pre-deployment phase, a deep learning module predicts the requests that might arrive in this service cycle, deploy the required vNFs and lightpaths in advance based on the prediction, and tear down the unused vNFs and lightpaths in the DCI-EON. Then, the provisioning phase and steer application traffic through the required vNFs to form a vNF-SC upon receiving each new vNF-SC request. To the end, the total latency of provisioning a vNF-SC request is shortened significantly, since the setup latencies of lightpaths and vNFs are removed from it, and the traffic steering can usually be finished within the hundreds of milliseconds in a software-defined networking based environment. Therefore, as seen in Fig., the selection of $\Delta T_n$’s value should be careful and adaptive to properly balance the tradeoff in a time-varying network environment. At the beginning of each service cycle, the DRL-Observer collects instant performance metrics regarding
the service provisioning in the previous cycle and feed them into its DNN to determine the duration of the current cycle and update the DL-based vNF-SC requests predictor and its own DNN.

Moreover, authors at [30–34] present their work to resolve the aforementioned issues by leveraging deep reinforcement learning in elastic optical networks. Including new techniques of deep reinforcement learning like multi-agent solutions.
Chapter 4

System description and assumptions

In this chapter, one can find detailed explanation on the system model, and how we escalate from the general model to the laboratory setup where simulations for this thesis have been tested.

4.1 System Model

Our proposed solution aims to model a system where radio access networks are connected through optical fibers and built upon a virtual network. The thesis aims to apply techniques of optimization and task-offloading to a virtual radio access network where each of the nodes contain a mobile edge computing server. The optimization part labels those topologies that exploit the maximum available wavelength, minimum number of packet errors, or packet loss; while task-offloading weights the performance parameters such as memory available, CPU usage, or temperature of the cores. Decision making relies on one of the nodes of the network which has control on the wavelength assignment and it is performed through a deep reinforcement learning algorithm.

As described in Figure 4.1, a VRAN may be composed by $n \in [1, N]$ radio access networks. All those RANs are equipped with one mobile edge computing server which help users process data, create and offer services, and operate complex systems. We differentiate two different RANs according to their functionality: (1) primary node (Node $N$) which acts as gateway to the Internet or higher layers of cloud computing servers and (2) secondary nodes (Node 1, ..., Node $N - 1$) which are connected to the primary node through an optical fiber that contains $\lambda_m$ wavelengths.
with \( m \in [1, M] \). Secondary nodes have similar capabilities as primary node, they help users in their area and are connected to the Internet/Cloud services through another node which acts as gateway. However, just primary node has a direct connection to the rest of nodes and is the one in charge on distributing the load if required.

Moreover, the solution proposed is potentially scalable. Instead of defining physical radio access networks as nodes, we could define a node as a VRAN composed by different nodes with the previously mentioned assignment of primary or secondary RANs. Although, this time primary VRAN should be able to weight the performance of secondary nodes globally, an addition of metrics might be done in order to optimize the VRANs-network. This proposed solution can be seen at Figure 4.2. Primary VRAN is connected to the external services while the other VRANs act as secondary nodes of the model.

![Figure 4.2: Scaled solution for virtual radio access networks](image)

### 4.2 Solution approach

As we approach from the general model to our solution it is first introduced the setup provided by the ECO group at TU/e - Eindhoven University of Technology. Basically, it consists of an optical metro access ring network with four nodes as illustrated in [35]. Each node is composed of a low cost semiconductor optical amplifier (SOA) based 2-degree re-configurable optical add-drop multiplexer (ROADM) with the function of switching and amplification, a FPGA based optical-electrical interface for traffic slicing and network status monitoring, a powerful server as the edge computing, a centralized SDN controller and NFV orchestrator. The metro access network employing disaggregated low cost 2-degree add-drop wavelength blocker (WBL) switch based on SOAs, which provide power amplification. The SOA gates inside the ROADM can be turning on and off by the FPGA based O/E/O interface to make each single wavelength pass/stop or drop/continue. Note that each of the incoming wavelengths are dropped to the current node no matter if it is blocked by the node or not, the node checks only the data that belongs to the node based on the destination information, similar to the mechanism in passive optical network (PON) technology. Furthermore, it is only possible to add traffic on a wavelength only if it is free or was blocked by the node itself. For that matter, the SDN controller is responsible of assigning wavelength connections between nodes. Besides driving SOAs, the FPGA interface is also working as traffic classifier, monitor and BVT controller. The incoming traffic is sliced by the FPGA based interface according to its type. Traffic of different VNF chains will be destined and sent to different nodes.
4.2. Solution approach

according to the flow map information from OpenFlow control packets. All the connections between metro nodes have 10km of fibre with four available wavelengths. The control system includes ONOS [36] based SDN controller, Netconf and OpenFlow based SDN agent and FPGA control logic for implementing the configuration commands. Each FPGA is equipped with four 10Gbps SFP+ transceivers (ITU Ch 21, 23, 25, 27), i.e., four different wavelengths.

Therefore, our virtual radio access network (VRAN) contains four nodes with the already mentioned characteristics. Every node is equipped with a powerful server that will act as a mobile edge computing (MEC) server, and 10Gbps transceivers at four different ITU channels. As is can be seen in Fig. 4.3, the ring metro access network is modelled in a manner that allows a Virtual RAN scenario.

However, connection between secondary and primary nodes use the wavelength assignment from Table 4.1. Where there is one different wavelength to interconnect Node4 with the rest\(^1\). Therefore, primary node is going to be Node4 with Node1, Node2, and Node3 acting as secondary as it can be seen in the Fig. 4.3. We assume that all nodes will act as RANs which obtain traffic from their area that could potentially be from many different applications, such as, mobile communications, connected vehicles, security systems, health monitoring, etc. Secondary nodes will send traffic to Node4 mimicking the behaviour of requirements to connect services to which primary node has access or to offload tasks if needed according to their performance parameters. Then, Node4 is going to address the traffic generated from its own network, will distribute generated traffic from/to Node1, Node2, and Node3, and will act as gateway to Internet/Cloud services.

Moreover, the SDN controller is installed in Node1 of the network, thus wavelength assignment and collection of metrics will be performed at this node.

\(^1\)This is due to the restrictions from the setup

<table>
<thead>
<tr>
<th></th>
<th>Node1</th>
<th>Node2</th>
<th>Node3</th>
<th>Node4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>(\lambda_3)</td>
</tr>
<tr>
<td>Node2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>(\lambda_1)</td>
</tr>
<tr>
<td>Node3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>(\lambda_2)</td>
</tr>
<tr>
<td>Node4</td>
<td>(\lambda_3)</td>
<td>(\lambda_1)</td>
<td>(\lambda_2)</td>
<td>-</td>
</tr>
</tbody>
</table>

**Table 4.1:** Available wavelengths for connections between nodes.
Chapter 5

Implemented Solution

This section presents the framework for the reinforcement learning solution and how is adapted to our case scenario. Later, details on the steps of the algorithm and the process of every iteration is explained and an schematic of the different steps of the implemented solution can be found (Fig 5.2).

5.1 Reinforcement Learning Framework

Once the system model is defined we could already discuss the definition of our reinforcement learning approach. As a model-free approach, our RL-based algorithm does not require any prior knowledge on the network dynamics and statistics. Based on the explanations of the previous section we have 4 nodes that act as RAN, with one primary node connected to all the services (internet/cloud and rest of the nodes) and three secondaries only connected to the primary. Moreover, Node1 is going to be our intelligent node that will decide which topology maximizes the different parameters that we will see on following sections because is where the SDN controller is installed. In this section, we will define the state and action space, and also the reward function.

5.1.1 State space

We approach our state space as a vector with information of the topology of the network, and thresholds assigned to parameters like free CPU, and RAM available. The topology of the network can be expressed as a matrix (Table 5.1) where a node has assigned a value from 0 to 15 for the first, second and third node right after it. This number represents an hexadecimal value which gives information on the wavelength used for the connection between nodes, i.e., number 4 (0100) means that the connection uses $\lambda_3$ or similarly number 7 (0111) makes use of $\lambda_1$, $\lambda_2$ and $\lambda_3$.

Moreover, complementing this matrix there is another one with threshold values for different relevant metrics as it can be seen on Table 5.2, where there is information about the CPU, and RAM for each node. The fact, that the state value function

<table>
<thead>
<tr>
<th>Node</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0...15</td>
<td>0...15</td>
<td>0...15</td>
</tr>
<tr>
<td>2</td>
<td>0...15</td>
<td>0...15</td>
<td>0...15</td>
</tr>
<tr>
<td>3</td>
<td>0...15</td>
<td>0...15</td>
<td>0...15</td>
</tr>
<tr>
<td>4</td>
<td>0...15</td>
<td>0...15</td>
<td>0...15</td>
</tr>
</tbody>
</table>

Table 5.1: Matrix illustration of the topology of the network that defines the state.
contains information about the CPU usage and the available memory is an improvement compared to other work on this topic. Our solution seeks not just a network optimization in terms of bandwidth but also about performance of each node.

However, the first state space is a matrix defined with a total of $16^{12} \times 4^8$ state possibilities, which prevents the algorithm to perform due to out of memory issues. It would require a lot of time for training and it actually breaks the code due to an out of memory error. That is the reason why a reduction on the number of possible states would be required. Regarding the first part of the state matrix where the topology of the network is defined it is possible to reduce the number of possibilities. Basically, in our setup (explained at section 4.2) we just have three wavelengths to assign in total. Moreover, once a wavelength is established to a link between nodes it blocks the wavelength to be used for the rest of the nodes. For instance, one of the possible topologies that could have been configured by the agent can be seen in Fig. 5.1. Different wavelengths are assigned to connect different nodes, problem is on the top-right link, where $\lambda_4$ is used by the orange and red node. Therefore, we have seen with a simple example that not all the $16^{12}$ possibilities are feasible which allows us to find a way to reduce the state space. Together with our approach of a Virtual RAN scenario we designed most of the topologies where one node was connected with the rest using different wavelengths and ended up with a total of 257 available topologies. From these different states by definition some of them would not work due to the wavelength assigned to the connection but this was an interesting challenge for our algorithm to check its performance not just for best topologies but also for non-realistic ones.

Although, this was a significance reduction on the state space memory problems would still arise due to the complementary matrix. For this matter, the number
5.1. Reinforcement Learning Framework

<table>
<thead>
<tr>
<th>Node</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>CPU</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0...15</td>
<td>0...15</td>
<td>0...15</td>
<td>0...2</td>
<td>0...2</td>
</tr>
<tr>
<td>2</td>
<td>0...15</td>
<td>0...15</td>
<td>0...15</td>
<td>0...2</td>
<td>0...2</td>
</tr>
<tr>
<td>3</td>
<td>0...15</td>
<td>0...15</td>
<td>0...15</td>
<td>0...2</td>
<td>0...2</td>
</tr>
<tr>
<td>4</td>
<td>0...15</td>
<td>0...15</td>
<td>0...15</td>
<td>0...2</td>
<td>0...2</td>
</tr>
</tbody>
</table>

Table 5.3: State space matrix for the reinforcement algorithm.

of thresholds was reduced for CPU and RAM to three possibilities. Therefore, a number 2 on the matrix would mean that the CPU/RAM at that node is stressed according to its maximum threshold specified beforehand and 0 would mean no stressed node. Finally, our state space was defined as it can be seen in Table 5.3 by a matrix of 5\times4 with the constraints already mentioned.

All these assumptions leave the state space with a total of $257 \times 3^8 = 1.686.177$ possibilities.

5.1.2 Action space

After the state space is defined, we have to set how action space looks like. This is a relevant matter for the reinforcement algorithm, agent is going to act accordingly to the action space we define which will lead to a new state reinforced by a reward value. This is a significant decision, as our interest is to find the optimal topology at any time $t$ the action performed by the agent should include information on the new topology, i.e., the agent should be able to change the topology. Therefore, this crucial decision simplifies our range of possibilities. As seen in the previous section we have a matrix (Table 5.1) of 4\times3 with a range of 0 to 15 depending on the wavelength assigned. Actually, this could cause a memory problem to the algorithm (as seen in Section 5.1.1, thus we modify the action space to all the possible topologies available at the state space which gives us a total of 257 actions.

5.1.3 Reward function

The reward function represents the prize or punishment to which the environment feeds the agent together with its current state. The value of the reward gives an insight on how the environment has reacted given an action, it represents how good or bad the previous action affected the environment. After some training steps we clearly want to see how the reward keeps increasing and stabilizing on a certain value until its convergence.

In our scenario we want to maximize mainly the bandwidth, thus we are going to reward more all those network interfaces that use their peak of available bandwidth while low values of bandwidth will have less effect on the reward. But this is not as easy at seems because maximum bandwidth for all network interfaces is not the only value that optimizes our network. We are also interested on having low values for CPU load or memory free. Related to the previous mentioned metrics, environment is going to reward more those nodes that have a lower load which can affect to the learning of the agent. For instance, from all the 4 nodes there are three which are highly loaded in terms of memory, this information is contained both in the new state and reward obtained from the environment at time $t$. Once, the agent has been trained and its fed with these new values, it will evaluate and decide which is the action to take. According to the scenario mentioned the state with information...
on the saturation of three nodes will receive less reward, therefore the agent will
decide whether at time $t + 1$ it is necessary to change topology or keep the same
one. The expected action would be to decide a topology where the “free” node has
more dedicated wavelengths so in terms of edge computing its load will increase
compared to the others that will see how their load reduces.

Another scenario that could occur is when one of the fibers between links is bro-
den. Since the agent is continuously learning at certain point after some iterations
it will realise about this imperfection. Therefore, new topologies where the men-
tioned link has lower weight in terms of the reward received will be skipped while
the others will be highlighted.

Mentioned below there are the metrics used which have an influence on the final
reward. Note, that reward values are not given in this section since it is not the
purpose, on Chapter 5 more on this parameters is explained.

- **Free CPU**: evaluation on how free the average of all CPUs at each one of the
  nodes is. Note, that each node may have up to 18 CPUs.

- **RAM available**: obtains a reward according the value of memory available at
  all the nodes.

- **Network Interface Bandwidth**: analysis on the total bandwidth at each of the
  network interfaces for all nodes, as close as the bandwidth used gets to the
  maximum the higher the reward is.

- **Packet loss**: takes into account the difference between the total bandwidth
  send and received through the network. According to the difference more or
  less reward is given.

- **Temperature**: evaluates what is the average temperature of all the CPUs at
  each node.

- **Number of errors**: information regarding the number of errors per network
  interface at all the nodes. More errors imply less reward.

### 5.2 Algorithm definition

Thanks to the assumptions from previous sections we discuss here on how the so-
lution has been implemented, its functionality, scalability, performance, and robust-
ness.

Firstly, we represent from the starting point until the very end the solution imple-
mented. Meanwhile, details about the starting point until the very end the solution imple-
mented. Meanwhile, details about the characteristics of the code, software decisions,
metrics evaluation, among others are given.

#### 5.2.1 Scenario initialisation

Starting point of the algorithm is having the setup ready and prepared for the tasks
that will be taken during all the process. Mainly, get permission to the SDN con-
troller to change the topology of the network at any given time.

#### 5.2.2 Metrics collection

Secondly, it is necessary to collect the interesting metrics for all servers and the topol-
ogy of the network. This will give us knowledge about all the parameters that have
been already mentioned: link-assignment of the wavelengths, CPU usage, free memory, available bandwidth, CPUs temperature, etc.

5.2.2.1 Data collection

For this implementation we used the open-source software Netdata [4]. Netdata is a distributed, real-time, performance and health monitoring software for systems and applications, it provides insights in real-time about everything happening on the system and it runs on browser manner. Dashboards with all kind of information about the running system are presented on a very interactive and clear fashion. Moreover, it operates according to the memory requirements, using only idle CPU cycles, by default it contains certain plugins that collect system metrics, but its behaviour is extensible by using its plugin API.

Every second, Netdata collects 1.938 metrics, presented in 299 charts. Additionally to all dashboards Netdata also presents the metrics on json format. Taking profit of this, the first implementation was creating a data collector, which was in charge of collecting all the metrics at any time \( t \). The collector is configurable, and one can decide which metrics are going to be collected before launch time.

5.2.2.2 Topology collection

Since we use ONOS as SDN controller, the topology of the network can be modified from the files found at the first node (where ONOS is installed). The procedure is similar, the ONOS files which define the topology of the network are written in json, thus another collector is implemented to obtain and update at any time \( t \) the wavelengths assigned to all the links.

5.2.3 Metrics evaluation

Third implementation consists on evaluating the metrics obtained from the data collector. Parsing the data, we get the actual status of the environment, containing information of the necessarily metrics to define the state and reward that are going to feed the deep learning agent.

5.2.3.1 State

As mentioned at Section 5.1.1 is a matrix of 3 by 4, with numbers from 0 to 15 that represent the wavelengths assigned to any specific link. The actual value needs to be interpreted as 4 binary number where the position of active digits (1’s) correspond to the number of wavelengths designated.

5.2.3.2 Reward

The reward is a scalar number obtained from the evaluation of the metrics from the data collector. The reward given for a specific set of metrics can be fine-tuned for learning purposes. Section 5.1.3 already talks about the purpose of the reward and its relevance for the algorithm.

5.2.4 Deep Learning Agent

Almost at the end process is where the agent is fed with the values for the current state and reward of the environment. Before going into details, it is important to
mention that a DDQN model is implemented for the solution of this thesis. Therefore, as explained in Section 2.2.3.1.1, there are two networks which together among other advantages, speeds the learning of the agent.

All starts with a random initialisation of the environment. Then, a state and reward are given and forwarded to the agent which stores to an experience replay list the values of the previous state, action taken, reward, next state and a flag that activates when a maximum reward is reached. This maximum reward does not represent the total sum of the maximum reward you get for all the metrics, but it is a high enough number that illustrates when the reward is a good approximation to the maximum. When the flag is activated the target network is aligned with the $q$-network, i.e., weights of the NN are updated at the target network.

5.2.4.1 Neural Network

The neural network of both networks is the same, they have one input layer, two hidden layers and one output layer. As an input it takes the state at time $t$ and returns an action according to the learning of the network.

5.2.4.2 Decision Making

The agent decides which action to perform according to a $\epsilon$-greedy policy (Section 2.2.2.2.1). Depending on the value of $\epsilon$ the agent will weigh more the values obtained from the neural network (experience) or it will choose random actions (exploration), this parameter can be fine-tuned and potentially has a big influence on the final results.

5.2.4.3 Action Implementation

After the action is chosen the agent updates the ONOS configuration that changes or not the topology of the network.

5.2.5 Iterative process

Lastly, the process explained above enters into an iterative process for $E$ epochs and $T$ timesteps. Each epoch has different values at the batch and during $T_t$ timesteps the batch is kept until it finishes all the timesteps or the maximum reward is reached.

5.3 Traffic and stress scripts

In order to analyse the results, traffic flow was needed to be generated. For this purpose, a script creating `iperf` connections between nodes through the two network interfaces activated was used. This script allowed to create more or less flow agents to increase/decrease the bandwidth used, every flow agent was basically generating client-server connections through the different ports available. From the node stress point of view, a software for Linux called `stress` was used. A script was created to randomly stress the CPUs and RAM from every node to expose the agent to the maximum possible states.
5.3. Traffic and stress scripts

(A) Network initialisation.

(B) Netdata collection of metrics at all nodes.

(C) Data collection from netdata by the main node.

(D) Data and topology analysis by the main node.

(E) Analysis of metrics and result of new state and reward from environment.

(F) Feeding DL agent with information of state and reward.

(G) Decision making by the agent.

(H) Update of the new topology according to the action.

Figure 5.2: Implemented solution with all the steps taken by the algorithm.
Chapter 6

Performance Evaluation

This section shows the results obtained from the tests conducted at the setup from the ECO group at the Eindhoven University of Technology.

Previous sections have explained some of the parameters that can be fine-tuned in order to obtain the optimal solution. As a reminder we are searching for the best topology at any time $t$ that maximizes the reward.

6.1 Fine-tuning parameters

Starting point to better comprehend the results obtained is to detail which parameters can be fine-tuned. Below there is a definition for each of them and how they might affect to the final solution:

- **learning rate**: determines the step size at each iteration while moving towards a minimum loss function. When setting the learning rate there is a trade-off between convergence or overshooting of local minimums.

- **batch size**: number of training examples used in one epoch. High number may slow down the decision making but a slow number may affect the learning of the agent.

- **number of episodes**: number of times we iterate over the environment. Necessarily, the number needs to be big enough to check the results obtained by the algorithm, small number may lead to no results.

- **timesteps per episode**: maximum number of repetitions of one epoch. Needs be a big enough number to let the algorithm reach the maximum reward.

- **flow agents**: number of agents that generate traffic within the network; more agent flows generate more traffic which leads to higher bandwidth, thus higher rewards.

- **stressed values**: there is a file that is in charge of stressing different nodes. It can be modified by just stressing specific nodes or specific memories or CPUs in the network.

- **optimizer**: in deep learning it shapes and molds the model into its most accurate possible form.

- **maximum reward**: is the reward that optimizes the solution. Not the sum of all possible rewards but a good approximation that lead the agent to a proper learning.
Chapter 6. Performance Evaluation

<table>
<thead>
<tr>
<th>Metric</th>
<th>Test1</th>
<th>Test2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum reward</td>
<td>1960</td>
<td>2150</td>
</tr>
<tr>
<td>Flow agents</td>
<td>35</td>
<td>40</td>
</tr>
<tr>
<td>Gamma</td>
<td>0.6</td>
<td>0.7</td>
</tr>
<tr>
<td>Epsilon</td>
<td>0.1</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Table 6.1: Parameters fine-tuned for the tests launched.

- **reward function**: as explained in previous sections, the reward for all the metrics can be fine-tuned to weight differently the parameters. According to the end goal one can decide what are the most relevant metrics and assign a higher weight to them.

- **state size**: defines how many different states our environment might experience; larger sizes might cause problems of convergence through time or computation problems. Also, small numbers of states might lead to a non-realistic problem-solution.

- **action size**: defines the number of actions that the algorithm can perform and has the same trade-off as the state size.

- **gamma**: is the discount rate of the reinforcement learning algorithm. It varies between 0 and 1; the higher the value the more weight you are giving to previous rewards for a specific action.

- **epsilon**: is a credit assignment variable; it is also compressed between 0 and 1; decides how much bootstrap on earlier learned value versus using the current roll-out.

- **model**: which is the connection on the deep learning model; number of convolutional neural networks, embedding layers, hidden layers, etc.

### 6.2 Results

This section shows the results obtained from two different test (6.1), the parameters fine-tuned for tests are the following: maximum reward, flow agents, gamma, and epsilon. The other parameters kept the same for both tests.

Training both algorithms took approximately the same amount of time, around 13 hours. For Test1, a total of 238 different actions, for 1078 states experienced by the agent, were taken within a total of 6749 decisions. On the other hand, Test2 took 255 different actions for 7601 decisions by looking at 979 unique states. The magnitude on the number of states experienced compared with the total amount of possibilities (number of states: 1.686.177) represents around a 0.1% of the total, i.e., from all the possible states due to different topologies and information about the stressed nodes, the agent just learned its solution from an infinitesimal states. Interesting results from first analysis of the solution for the tests show which are the most used topologies. For instance, at first test the topology most used was the one from Figure 6.1a with a total of 293 occurrences, while for the second one the topology most chosen had 147 (Fig. 6.1b) instances. Advancing on our analysis, one could evaluate if the most selected topologies by the agent are accordingly to our approach. From previous sections, on Table 4.1 there are just two wavelengths
that communicate properly between primary-secondary nodes. As seen in these examples, it reaffirms the thesis that for those topologies where the wavelengths used are the functioning ones have more occurrences. On the opposite hand, if we take a look at those topologies with the smallest average reward, they either use one of the available wavelengths (non-profiting all the resources available) or topologies where wavelength assignment do not allow the flow of packets.

![Figure 6.1: Topologies most used for the agent during training.](image)

Both scenarios, could represent situations where just two secondary RANs are demanding/providing to the primary node, thus the assignation of wavelengths to maximize the resources is done in a manner that the demanding nodes send/receive traffic from two different wavelengths. Probably during the learning, these nodes were the most actives which provided higher reward to the deep agent that led the selection of these topologies. How the agent acts relates on how the network is behaving, at any point of time, the agent might experience that some nodes are either demanding more resources, or a specific non-connected node is saturated, or connections with other wavelengths are malfunctioning, or any other situation that leads to a change on reward. This might be confusing, but our environment is dynamic and could experience sudden changes in short periods of time. Therefore, agent has to learn about these changes and has to perform according to past-learning and exploration of new results.

Although, these are interesting values to analyse they are not relevant to evaluate if the algorithm is working. This analysis requires a more in deep study. First of all, we have to check if the curve of the reward has an increasing tendency. As one can see in Figure 6.2, the trendline drawn for the first test has a flat shape while the other curve has clearly an increasing shape which means that the algorithm is learning, the reward increases accordingly to the training process.

Therefore, we have already seen that the first experiment failed in terms of learning. The fact, that reward does not increase in time means that generally either any of the metrics inside the reward function improved in value or that when improving on one side we were downgrading on another. As seen in the beginning of the section the tests had different values for the maximum reward, the number of agents, gamma and epsilon. Reasonably these changes affected the final results, and below there is a deep study on how these are connected.

There are basically three major changes from the four fine-tuned parameters. First one, related the increase on flow agents and the maximum reward. Related to the number of flow agents it is important to mention that in total a maximum of 35 and 40 were active for the first and second test, respectively. Every flow agent was able to generate a traffic flow of 800 Mbits/s using UDP protocol, which was the protocol used for traffic generation because it did not require any acknowledgement.
that the connection was on-line, thus generating the maximum traffic to simulate an scenario where every node/BS/RAN was asking for the maximum optical resources possible. Therefore, when maximum reward increases, we should expect a different behaviour from the agent because we are asking to find actions with better reward. Although, since we also changed the number of flow agents, to 35 instead of 40, we would also expect that the bandwidth used for the nodes would be higher. There is a trade-off between these metrics, but not changing the maximum reward and increasing the number of flow agents would probably lead to poor results. At this point, we cannot conclude that these changes helped the learning of the agent. Secondly, for a higher gamma we should expect that agent relies more on accumulated learned experience. Consequently, the $q$-value on past actions are weighted more for the second test, i.e., $q$-table should update basing on the accumulated experience and those actions with higher reward on the past should be highlighted by the $q$-learning algorithm. Lastly, doubling in factor the exploration value (epsilon) might represent the highest change between both tests. Dynamically our environment is really changing, thus more exploration on new wavelength assignments should lead to better performances by the agent.

After analysing, how the variation on the metrics between tests could have affected the result we have to analyse why reward has increased in time and which metrics from the reward function have helped the most.
We should analyse all the metrics that are included on the reward function, below we are going to analyse one by one to see how they have evolved in time.

The first, which is the one weighted the highest in the reward function is the bandwidth received by the network interfaces for all the nodes. Figure 6.3, shows the bandwidth received by all the nodes including a trendline for both network interfaces. Bandwidth received by the first node at the middle of the test suddenly jumps to 0 values. Taking a look at the results obtained we could address this issue to a problem of either the topologies chosen by the agent or a malfunctioning piece of hardware/software during test. Besides, second and third node have a similar behaviour, their trendline has a decreasing factor which means that the bandwidth received at the end is lower compared with the beginning of the simulation. Lastly, it is in the fourth node where we experience an increase of bandwidth received through time, meaning that the algorithm could be deciding actions according to the needs of this node which may lead to higher reward values.

Similarly, the amount of errors at network interface level are weighted on the reward function. Looking at the trendline (Fig. 6.4) one can see it is pretty flat which means that the number of errors has stayed constant during the test. Meaning the compensation of the metric on the reward function has not influenced on the final value.

Performance metrics, like the load factor obtained from Netdata or the CPU usage and RAM available were also included on the reward function and weighted on a lower level compared with the previous metrics mentioned. Results show that all the nodes were stressed on the same level, thus the algorithm could not take actions to perform task offloading, i.e., assign less bandwidth resources to those nodes with higher level of stress.
Lowering on the scale of importance in the reward function there is the temperature of the CPUs, this is important for energy efficiency measures. By rewarding/punishing this metric the agent could also decide which topology performed better for high/low average temperature of the CPUs. Even though, this is an important issue to save energy it was not exactly weighted as other parameters because this thesis did not aim to find the most energy efficient solution. Unluckily, during both test temperature of the CPUs stayed constant and thresholds by temperature were not adequate to actual values.

Lastly, the algorithm was also analysing the packet loss. Our approach consisted on including into the reward function information about the difference between the bandwidth sent and received by all the nodes through the different network interfaces. During both tests UDP protocol was used, which means that traffic was generated to send packets even if node was unreachable. Therefore, high amounts of traffic would be sent and compared with the bandwidth received by the nodes the difference would be too big, that is the reason why this parameter was weighted the lowest at the reward function. Same effect, by the information on just the bandwidth sent without comparison between the received\(^1\).

\(^1\)Note: results may not have been conclusive due to a problem with the setup that forbid the execution of more experiments.
Chapter 7

Conclusions and Future Work

The goal of this project was to model a ring metro access network as a virtual radio access network to implement a machine learning solution using deep reinforcement learning techniques to optimize and offload tasks within all the nodes of the network. Once, we had a clear understanding on how to model our environment, our system approach for the deep reinforcement learning algorithm was very relevant to get to the final solution. Assumptions needed to be made and simplification of the problem played a key role towards the definition of the state and action space. On the other hand, the definition of the reward function was straight forward; our learning algorithm was reinforced by reward-based solution. Its implementation included all those parameters that the algorithm should be able to optimize according to the values collected from the network and the nodes. Moreover, as explained on the thesis it is very adaptable and modifiable. Results show that almost all the actions were executed and without need to experience all possible states the agent was able to learn thanks to its internal neural network as it can be seen on the second test. Unluckily, we were just able to demonstrate that the bandwidth received at the primary node influenced the learning of the algorithm. Due to the lack of tests we were not able to prove that the agent was able to learn also from other metrics by fine-tuning other parameters in the algorithm. Probably in the future more tests can be done and lead to better results by modifying values, metrics, states, actions, rewards, etc. A part from this situation, insights on how to approach the problem and what are the most interesting parameters to study are presented which may lead to a wide variety of possibilities that could be implemented on the future. Moreover, the solution implemented is scalable, i.e., it could adapt to bigger scenarios but carefully looking on how the state and action space increase because of the limitation of computational resources we have available nowadays. Below there is some future work that could be done in order to improve results, escalate the solution, improve its energy efficiency, how to approximate to a real-world environment, etc.

First, to get to better results we should be able to run more tests and change values on different sections of the algorithm to analyse differences between implementations. As explained in Section 6.1, there are many parameters that could be fine-tuned in order to obtain different results, some of them have more direct influence on the final solution but being able to study how the agent adapts to those changes has a direct impact to fully understand their meaning which could lead to the optimal solution. Below there are two examples that could lead to better results:

- For our tests all the nodes were stressed on the same manner, by just stressing, e.g., half of the nodes. We could evaluate whether the agent provides resources to those stressed or offloads tasks to the other nodes of the network.
Chapter 7. Conclusions and Future Work

- Changing the number of flows generated by each node. Similarly, to the previous point, tests had the same amount of flows at all nodes, instead, by just generating flows at specific nodes we could have evaluate how the agent changes its behaviour and analyse which are the most defined network topologies.

Secondly, it is important to note that our environment is not constant, state of the environment (state and reward) at time $t$ is analysed and actions are performed at $t + T$. During this period the network may experience changes, for instance, new UEs could be connected to a RAN in the network or demanding of traffic by a certain node could decrease. Agent must be able to learn about this kind of behaviour but it may still present problems related to latency. Our solution is based on the collection of metrics from Netdata. Actually, Netdata offers a wide range of possibilities, hundreds of metrics from each node were provided on real-time. It is a software that has been key for the development of the solution although it has some restrictions. Metrics are provided with a granularity of one second which means that it is not possible to optimize latency values. Therefore, other solutions to collect metrics might be studied. In fact, it is possible to implement a manual solution with scripts that look for those metrics we are interested in. Then, collection procedure should be also changed but new issues might be addressed to the agent.

Thirdly, it would be very interesting to train the network for a specific scenario, whatever it is. Then, immediately modify the environment and check what are the metrics that could adapt better without further training. This could proportionate a better understanding on how the algorithm is learning and improve the solution for a better convergence.

We could also escalate our problem but on a different manner as previously explained. Due to the constraints from our setup we had to change our approach, we moved to a Virtual RAN scenario from a Metro Access Ring Network. Without limitations in terms of wavelength assignment between the links to all the different nodes, the approach of a Metro Access Ring Network would have been also valid. For this approach, some changes would need to be applied to the algorithm to adapt the new solution. In fact, learning would be different, thus state and action space together with the reward function should be modified but the core idea from the solution of the project could be helpful for the new implementation.

Moreover, interesting work could be to test the algorithm on a real scenario. The experiments were done on a laboratory setup, where random traffic was generated by us on a non-sophisticated manner that tried to simulate real traffic. On a real-world scenario other metrics could be included on the algorithm and more in deep solution could approximate better and even lead to better results.

Finally, it would also be interesting to try other algorithms of deep reinforcement learning for our scenario. Specially a Deep Deterministic Policy Gradient algorithm which is widely used in the literature and it might proportionate better results to our case scenario. Moreover, researches on deep reinforcement learning are moving towards to new algorithms with multi-agent approaches, or with reinforcement action or state-based algorithms. Reinforcement learning is a really hot-topic and academia is studying and understanding how to obtain more general algorithms, more future work is going to reveal better solutions not just for this problem but for more global issues on real world.
Bibliography


