CENTRO UNIVERSITÁRIO FEI

SILVIO ROMERO DE ARAÚJO JÚNIOR

A PROFILE-BASED ARCHITECTURE FOR TRAFFIC FORWARDING THROUGH SERVICE FUNCTION CHAINING USING DEEP REINFORCEMENT LEARNING TECHNIQUES

São Bernardo do Campo

2021

SILVIO ROMERO DE ARAÚJO JÚNIOR

A PROFILE-BASED ARCHITECTURE FOR TRAFFIC FORWARDING THROUGH SERVICE FUNCTION CHAINING USING DEEP REINFORCEMENT LEARNING TECHNIQUES

Master thesis presented to Centro Universitário FEI to obtain the degree of Master in Electrical Engineering. Advised by Prof. Dr. Reinaldo Augusto da Costa Bianchi.

São Bernardo do Campo

2021

de Araújo Junior, Silvio Romero.
A Profile-Based Architecture for Traffic Forwarding through Service Function Chaining using Deep Reinforcement Learning Techniques / Silvio Romero de Araújo Junior. São Bernardo do Campo, 2021.
82 f.

Dissertação - Centro Universitário FEI.
Orientador: Prof. Dr. Reinaldo Augusto da Costa Bianchi.

1. Deep Reinforcement Learning. 2. Service Function Chaining. 3. Fifth-generation mobile networks (5G). 4. Autonomous Network. I. da Costa Bianchi, Reinaldo Augusto, orient. II. Título.

Elaborada pelo sistema de geração automática de ficha catalográfica da FEI com os dados fornecidos pelo(a) autor(a).

centro universitário FEI

APRESENTAÇÃO DE DISSERTAÇÃO
ATA DA BANCA EXAMINADORA

Programa de Pós-Graduação Stricto Sensu em Engenharia Elétrica

Mestrado

PGE-10

Aluno: Silvio Romero de Araújo Júnior **Matrícula:** 118324-3

Título do Trabalho: A Profile-Based Architecture for Traffic Forwarding through Service Function Chaining using Deep Reinforcement Learning Techniques.

Área de Concentração: Inteligência Artificial Aplicada à Automação e Robótica

Orientador: Prof. Dr. Reinaldo Augusto da Costa Bianchi

Data da realização da defesa: 20/12/2021

ORIGINAL ASSINADA

Avaliação da Banca Examinadora:

A banca ocorreu no dia 20 de dezembro de 2021 de maneira on line as 9:00 horas

O aluno realizou a apresentação no tempo regular, e em sequência foi arguido pela banca.

O aluno respondeu às perguntas de maneira satisfatória.

A aprovação foi por unanimidade.

São Bernardo do Campo, / / .

MEMBROS DA BANCA EXAMINADORA	
Prof. Dr. Reinaldo Augusto da Costa Bianchi	Ass.: _____
Prof. Dr. Renato Camargo Giacomini	Ass.: _____
Prof. Dr. Carlos Alberto Kamienski	Ass.: _____

A Banca Julgadora acima-assinada atribuiu ao aluno o seguinte resultado:

APROVADO ☒ REPROVADO ☐

VERSÃO FINAL DA DISSERTAÇÃO

APROVO A VERSÃO FINAL DA DISSERTAÇÃO EM QUE FORAM INCLUÍDAS AS RECOMENDAÇÕES DA BANCA EXAMINADORA

_____

Aprovação do Coordenador do Programa de Pós-graduação

_____

Prof. Dr. Carlos Eduardo Thomaz

I dedicate this work to my wife for her unlimited support on this journey. And to my parents to help me build my character, the essential quality for the human being.

ACKNOWLEDGMENTS

I want to thank all the professors who have gone through my formal education because they are somehow responsible for all my acquired knowledge and inspiring ideas that made me move forward. However, a special thanks to my advisor Professor Dr. Reinaldo Augusto da Costa Bianchi, for his advice that allowed me to overcome the barriers encountered during the execution of this work. And I thank my wife Thaís because the journey becomes easier and more delightful with her.

"No man is an island entire of itself; every man is a piece of the continent, a part of the main; if a clod be washed away by the sea, Europe is the less, as well as if a promontory were, as well as any manner of thy friends or of thine own were; any man's death diminishes me, because I am involved in mankind. And therefore never send to know for whom the bell tolls; it tolls for thee."

Jonh Donne

RESUMO

Os sistemas de comunicação, como por exemplo as redes móveis de quinta geração (5G) têm se desenvolvido rapidamente e devido à esta evolução, o uso dos seus recursos tem se tornado mais complexo. Assim, são necessários novos métodos para a concepção das redes e uma das tecnologias para tal é o Encadeamento de Funções de Serviço que permite o tráfego através de Funções de Rede Virtualizadas flexibilizando o uso dos recursos. Entretanto, há alguns desafios para sua implementação, como por exemplo o volume de dados gerados pelas novas aplicações. O Aprendizado por Reforço Profundo tem sido usado para resolver diversos problemas computacionais, inclusive aqueles relacionados às redes de comunicação. Para atingir os objetivos de otimização de recursos são necessários: a identificação e tratamento do tráfego de rede e o seu correto roteamento pelos dispositivos. O objetivo deste trabalho é investigar como as técnicas de Aprendizado por Reforço Profundo combinadas com a arquitetura de Encadeamento de Funções de Serviços podem proporcionar um mecanismo eficiente de identificação e roteamento de tráfego baseado em perfis, auxiliando os dispositivos responsáveis pelo controle da rede a reconhecer comportamentos indesejáveis e tomar as ações necessárias. Para isto, será proposta uma implementação prática para demonstrar como estas técnicas podem ser aplicadas.

Palavras-chave: Encadeamento de Funções de Serviço. Aprendizado por Reforço Profundo. Redes Móveis de Quinta Geração (5G).

ABSTRACT

Communication systems, such as fifth-generation mobile networks (5G), have developed rapidly, and due to this evolution, the use of their resources has become more complex. Thus, new methods are required to design networks. One of these technologies is the Service Function Chaining (SFC) that allows traffic through Virtualized Network Functions (VNFs), making resources more flexible. However, its implementation has some challenges, such as the volume of data generated by the new applications. Deep Reinforcement Learning (DRL) has been used to solve several computational problems, including those related to communication networks. To achieve the objectives of resource optimization are required: the identification and treatment of network traffic and its correct routing through devices. This work investigates how Deep Reinforcement Learning techniques can be used with the Service Function Chaining architecture to identify and route based on profiles efficiently. That helps the elements responsible for network control recognize undesirable behaviors and take the necessary actions. A practical implementation is proposed to demonstrate how techniques can be applied.

Keywords: Service Function Chaining. Deep Reinforcement Learning. Fifth-generation mobile networks (5G).

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

5G	Fifth-generation mobile networks
A2C	Advantage Actor-Critic
ANN	Artificial Neural Networks
AP	Application Plane
API	Application Programming Interface
BSS	Business Support Systems
CLI	Command Line Interface
COTS	Commercial-Off-The-Shelf
CP	Control Plane
CSV	Comma-Separated Values
DL	Deep Learning
DNN	Deep Neural Networks
DP	Data Plane
DPI	Deep Packet Inspection
DQL	Deep Q-Learning
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
FCAPS	Fault Configuration Accounting Performance Security
GPU	Graphics Processing Unit
GUI	Graphical User Interface
IAAS	Infrastructure-as-a-Service
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
ITU	International Telecommunication Union
ITU-T	ITU Telecommunication Standardization Sector
MANO	Management and Orchestration
MDP	Markov Decision Process
MEC	Mobile Edge Computing

MP	Management Plane
NBI	Northbound Interface
NETCONF	Network Configuration Protocol
NF	Network Function
NFQ	Neural Fitted Q-learning
NFV	Network Functions Virtualization
NFVI	NFV Infrastructure
NFVO	NFV Orchestrator
OSI	Open Systems Interconnection
OSS	Operation Support Systems
PPO	Proximal Policy Optimization
QoS	Quality of Service
REST	Representational State Transfer
RFC	Request for Comments
RL	Reinforcement Learning
SBI	Southbound Interface
SDN	Software-Defined Networking
SF	Service Function
SFC	Service Function Chaining
SFF	Service Function Forwarder
SFP	Service Function Path
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VIM	Virtualized Infrastructure Manager
VNFM	VNF Manager
VNFs	Virtual Network Functions

CONTENTS

1 INTRODUCTION

Communication systems such as the Internet have been developing rapidly, and due to this evolution, the infrastructure, the devices, and resources of networked systems have become increasingly complex and heterogeneous. Therefore, significant changes to design the network are needed in the access layer and the network's core to meet the new demands for applications and services (LI, R. et al., 2018).

Fifth-generation mobile networks (5G) were designed to be the critical element that will enable new services and technologies, such as Big Data and the Internet of Things (IoT). According to Fu et al. (2019), some technologies have emerged as candidates to underpin 5G networks and help them to deal with unique traffic characteristics, such as the dynamic nature and high volume of data required by new applications: the Software-Defined Networking (SDN) (HALEPLIDIS et al., 2015; ITU-T, 2014), the Network Functions Virtualization (NFV) (ETSI, G., 2013) and the Service Function Chaining (SFC) (HALPERN; PIGNATARO, 2015).

The SFC architecture allows the flow of information to travel through Virtual Network Functions (VNFs), which provides better flexibility in using resources (LI, G. et al., 2020) as defined in (HALPERN; PIGNATARO, 2015) and (ITU-T, 2018). However, such technologies that will support next-generation networks bring some challenges for their implementations. For example, the efficient use of the radio spectrum, hardware computational resources that support the virtualized functions, the amount of data generated from new applications, and further information security threats arising from such applications. Therefore, it is crucial in this new scenario to direct efforts towards investigating using resources efficiently and intelligently (LI, R. et al., 2018).

1.1 MOTIVATION

According to Luong et al. (2019), Deep Reinforcement Learning (DRL) methods have solved the most various computational problems successfully. Therefore, it is natural to use them in environments related to communication networks (FU et al., 2019). Given the complexity of the networks and the volume size of traffic generated, defining the parameters used by Reinforcement Learning (RL) such as state spaces and actions can become cumbersome. The RL may not find the ideal policy reasonably, limiting its application in the dynamic environment of new communication networks. The combination of Deep Learning (DL) with RL helps to overcome such limitations.

Part of optimizing computing resources, providing better Quality of Service (QoS) to the customers, and reacting to security threats consists of identifying, handling network traffic, and routing it correctly through the devices. Thus, a study of applying Deep Reinforcement Learning methods in an environment with dynamic traffic, such as networks based on the concept of Service Function Chaining, is needed. It is possible to see the academic community's growing interest in the topic by analyzing the number of articles found in the related work research phase. For example, in 2018, the research found three, and in 2021, 112 so far, proving this research's relevance.

1.2 OBJECTIVES

This work investigates how DRL techniques can be used with the SFC architecture can provide a mechanism for identifying and routing traffic based on profiles by forwarding the packets to the correct path according to decisions taken by a learning agent. With the specific objective of helping devices responsible for network control to recognize data flows that are harming its performance, in terms of the use of computing resources and security threats, taking the necessary actions to mitigate such undesirable behaviors.

For this, the work deploys a profile-based traffic routing architecture, differentiating regular network traffic (applications known by the environment) and abnormal (unknown traffic). It uses Deep Reinforcement Learning techniques based on Deep Q-Network (DQN) and its variants (the Double DQN and the Dueling DQN) with the Service Functions Chaining mechanism.

The goal is to provide enough information to the network controller to take the appropriate actions, control the packet flow, and send the unknown traffic to a specific service chain. One of the possible actions in this service chain is implementing stricter security policies, such as bandwidth restriction or even packet blocking.

1.3 STRUCTURE OF THE WORK

The text is organized as follows: Chapter 2 presents the basic theory to understand the proposed model. Chapter 3 reviews the related works. Chapter 4 shows the methodology used by this research and explains the proposed model. Chapter 5 presents the experiments and discusses the results. Chapter 6 presents the conclusions, research contribution, and suggestions for future works.

2 THEORETICAL BACKGROUND

This chapter presents the theoretical background to understand the proposed architecture. It is divided into two parts: the first will address the concepts involving technologies considered the pillars of 5G systems, such as SDN, NFV, Mobile Edge Computing (MEC), and SFC. The second part describes the concepts related to Deep Reinforcement Learning.

2.1 NETWORK TECHNOLOGIES IN FIFTH-GENERATION MOBILE SYSTEMS

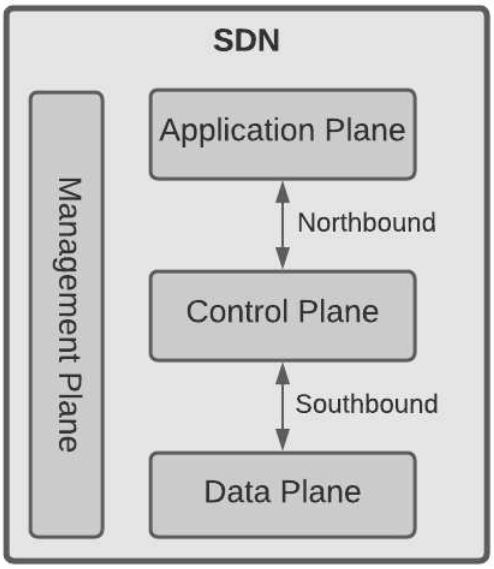
According to Blanco et al. (2017), the pillars of 5G networks are SDN, NFV, and MEC. The presentation of such concepts and terminologies is essential because the SFC technology, although not considered a pillar in the article mentioned, other authors place it as a facilitating component for new services in next-generation networks (BHAMARE et al., 2016; MEDHAT et al., 2016).

2.1.1 Software-Defined Networking (SDN)

The SDN architecture and its definitions, terminology, and objectives are found in the following technical standards: Request for Comments (RFC) 7426 (HALEPLIDIS et al., 2015), the International Telecommunication Union (ITU) recommendation Y.3300 (ITU-T, 2014), and the ITU document TR-521 (ONF, 2016). In addition to these documents prepared by organizations for standardization in networks and telecommunications, other works can be mentioned, such as publications by Kreutz et al. (2014) and Singh e Jha (2017). These papers present definitions, terminologies, examples of use, efforts of other researchers, and open opportunities for future research. This section summarizes the concepts found in these documents.

The main idea of the SDN architecture is the network programmability through the abstraction of layers, aiming at efficient resources usage and the acceleration in implementing new functionalities within the scope of next-generation networks. SDN achieves these objectives by separating the Control Plane (CP), the Data Plane (DP), and the Application Plane (AP). In addition to these three, ONF (2016) conceives a fourth plan: the Management Plane (MP). Figure 1 shows this functional structure. The following subsections describe the functions of the SDN planes.

Figure 1 – SDN Architecture



Source: adapted from Kreutz et al. (2014)

2.1.1.1 Application Plane

This plane is responsible for accommodating the applications that demand resources and services. Its requirements will define the behavior of the network elements, as examples of applications that reside in this layer: access control systems to network services, monitoring services, load balancers, and intrusion detection systems.

The Application Plane communicates with the Control Plane through standardized interfaces (crucial for SDN architecture). Although the SDN architecture documents do not define the protocols for communication between the interfaces, the interaction between the Application Plane and the Control Plane has been carried out through Application Programming Interface (API) based on Representational State Transfer (REST) (SINGH; JHA, 2017). In the context of ITU Telecommunication Standardization Sector (ITU-T), they are called Application-Control Interfaces, but the term Northbound Interface (NBI) is quite common in the technical literature as in Kreutz et al. (2014).

2.1.1.2 Control Plane

The Control Plane is the layer responsible for defining how to forward the packets by the network elements in the Data Plane. This function is performed by establishing the topology and controlling the routing/forwarding tables according to the network resources required by the services in the Application Plane. The great advantage arising from the separation of the Control Plane from other planes is the possibility of programming resources allocation dynamically.

The previous section explained the interaction with the Application Plane. Communication with the Data Plane takes place through what ITU-T defines as Resource-Control Interfaces, also known as Southbound Interface (SBI), according to Kreutz et al. (2014). Some examples of protocols that allow this communication is the Openflow, currently at version 1.5.1, specified in the ONF (2015), and the ForCES defined in RFC 5810 (HALPERN et al., 2010).

2.1.1.3 Data Plane

This layer is called Resource Layer by ITU-T, Forwarding Plane by Internet Engineering Task Force (IETF), and Data Plane by Kreutz et al. (2014). In this layer are the network elements responsible for receiving, processing, and sending (or discarding) the data packets according to the decisions taken by the Control Plane. In the present work, it will follow the nomenclature Data Plane.

It is noteworthy that such elements are deployed in hardware (physical) or software (virtualized). In the technical documentation, there are no restrictions regarding this aspect. The network infrastructure made up of links and devices such as routers and switches (physical or virtual) builds the functional structure of this layer. The Control Plane stores the routing/forwarding tables. The ONF has been providing invaluable efforts to support the integration between the planes.

2.1.1.4 Management Plane

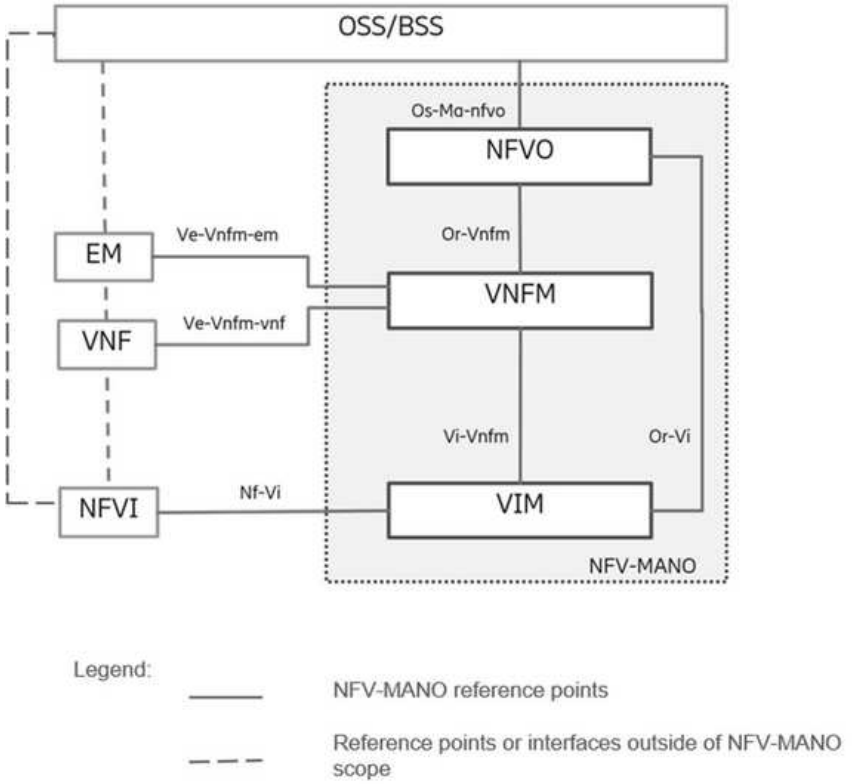
The Management Plane provides functions to ensure that the network is operating correctly, communicating with other planes or layers to support Fault Configuration Accounting Performance Security (FCAPS) as per recommendation M.3400 (ITU-T, 2000). These functions are essential for billing, customer service, statistics gathering, monitoring, and allowing dynamic provisioning of services.

How this plane communicates with others can occur through protocols, APIs, or internal communication between processes. Some examples are: the Simple Network Management Protocol (SNMP) described in RFC 3411 (HARRINGTON; WIJNEN; PRESUHN, 2002), the Network Configuration Protocol (NETCONF) proposed in RFC 6241 (ENNS et al., 2011), and the Syslog (RFC 5424) presented in Gerhards (2009).

2.1.2 Network Functions Virtualization (NFV)

Network Functions Virtualization decouples functional entities Network Function (NF) from defined hardware, making them elements based on software running on a server infrastructure with high computational capacity. These servers are called Commercial-Off-The-Shelf (COTS), which allows for more flexible, better performance, and fault-tolerant scalability of communication networks and provides a dynamic provisioning mechanism according to the demand of resources.

Figure 2 – NFV Architectural Framework



Source: GSNFV ETSI (2013)

The document: Network Functions Virtualisation Architectural Framework as described in GSNFV ETSI (2013) defines the NFV architecture. Figure 2 shows the topology for NFV and its blocks, and the following sections will explain their roles.

It is also important to emphasize that the communication between the different components and functions runs through standardized interfaces (reference points) described in the standardization document, which offers better flexibility to the operators that need to deploy the Telco cloud using a vendor-neutral architecture.

2.1.2.1 Management and Orchestration (MANO)

The block called Management and Orchestration (MANO) is responsible for managing and orchestrating VNFs. Such functions consist of instantiating (to start from a template) the virtual machines and the network services through the automation, provisioning, and coordination of workflows together with the blocks: Virtualized Infrastructure Manager (VIM) and the VNF Manager VNF Manager (VNFM). The MANO block is also responsible for integrating the NFV architecture with the Operation Support Systems (OSS) and Business Support Systems (BSS). MANO has three functional areas:

a) NFV Orchestrator (NFVO): responsible for the network resources and services orchestration. It interacts with the other NFV blocks to ensure that the resources and services requested by the applications are available and in an organized way to serve them;
b) VNFM: its function is to manage the so-called VNF life cycle (instantiate, monitoring, allocating/deallocating computational resources according to the demand, finalize and delete them), and coordinate changes to VNF images;
c) VIM: controls and manages the NFV Infrastructure (NFVI), where the COTS servers reside, allocating computing resources such as storage capacity, memory, virtual processors, and connectivity to VNFs.

2.1.2.2 NFV Infrastructure (NFVI)

The NFVI provides the virtualization layer and the physical computing resources (memory, processing, storage, and connectivity) using COTS servers (generic hardware). Hypervisors, defined as a specific operating system for managing and allocating hardware resources

to virtual machines, do the task of mapping the physical resources to the virtual ones. VIM manages NFVI.

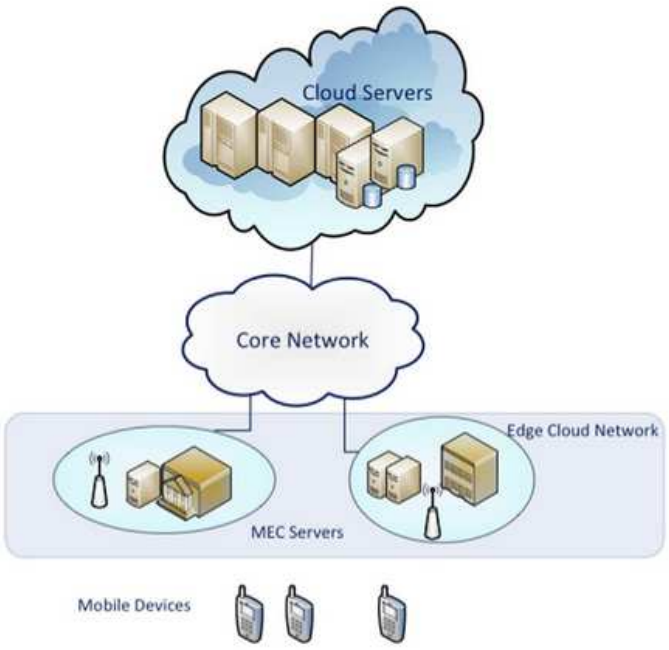
2.1.2.3 Virtualized Network Functions (VNFs)

A VNF is a software-based application that provides network services, such as routers, firewalls, intrusion detection systems, or load balancers. VNFs usually run in several virtual machines using the resources provided by NVFI to deliver the services and connect them to the network.

2.1.3 Mobile Edge Computing (MEC)

MEC defined in ETSI (2019) is a concept that allows cloud computing that processes information in centralized data centers can be done at the edge of networks, that is, close to the elements that require such processing.

Figure 3 – MEC Use Case



Source: Blanco et al. (2017)

In 5G networks, the goal is to integrate such technology in base stations (those that deal with radio frequency signals). The objective is to make delay-sensitive applications (such as

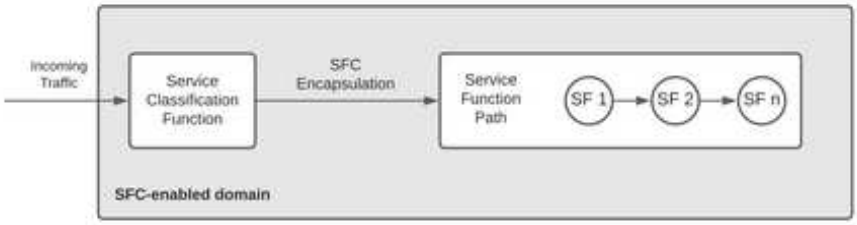
those based on Augmented Reality) not suffer from undesirable characteristics. A topology with a possible implementation of the MEC is shown in Figure 3.

2.1.4 Service Function Chaining (SFC)

Aiming to create a virtual path of network functions for packet forwarding, the SFC architecture establishes a chain of virtual service functions to route different types of network traffic according to particular characteristics through the VNFs. SDN and NFV technologies are the pillars that allow SFC deployment to offer a method of allocating network resources efficiently and flexibly.

DRL methods will enable it to do so dynamically without manual configuration and provide advantageous options for new networks, such as 5G systems and virtual network services in data centers for cloud implementations (ZHANG et al., 2018). Figure 4 illustrates a generic topology of the SFC architecture.

Figure 4 – SFC Architecture



Source: adapted from Halpern e Pignataro (2015)

Efforts by standardization bodies have been made to propose how to implement the SFC. For example, the IETF in the RFC 7665 (HALPERN; PIGNATARO, 2015), and in the ITU recommendation Y.2242 (ITU-T, 2018).

The SDN makes SFC possible by applying appropriate programming techniques to dynamically control the topology of virtual network functions and direct traffic through them. This mechanism provides advantageous options for new networks, such as 5G systems and virtual network services in data centers (ZHANG et al., 2018). On the other hand, NFV allows for efficient orchestration and implementation of the network functions.

(HALPERN; PIGNATARO, 2015) describes the components and functions, leaving it up to the implementations how they will be done, for example, in a single module or separately.

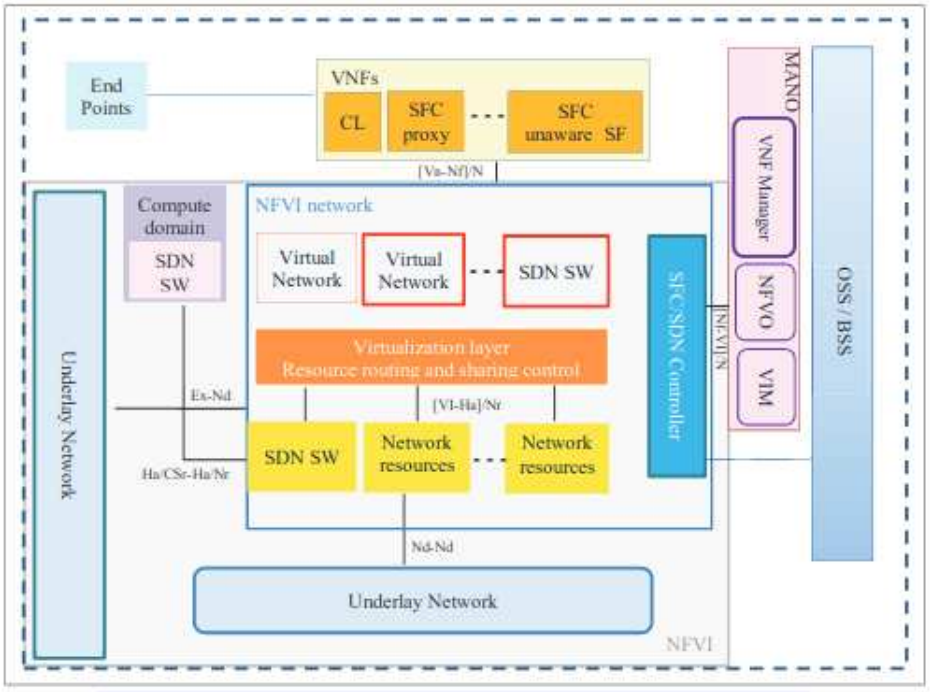
Such elements can be interconnected using the encapsulation (SFC encapsulation) defined by Quinn, Elzur, and Pignataro (2018) and form a path called Service Function Path (SFP) that establishes the route to be taken by the packets. According to Halpern e Pignataro (2015), four logical components form the SFC architecture:

a) Classifiers (Service Classification Function): entry (or exit) point into the SFC domain. Its function is to establish the criteria for classifying packets based on policies. The traffic that complies with such rules has its data encapsulated (VLAN tags or MPLS labels) and then forwarded to the Service Function Forwarder (SFF) function. Such classification can be done by criteria such as 5-tuple (source/destination IP addresses, source/destination ports, and transport protocol) or with Deep Packet Inspection (DPI);
b) Forwarders (SFFs): after defining the classification policy, the SFF directs traffic to the Service Function (SF) connected according to the assigned encapsulation, besides processing the traffic received from the SFs. It is also possible to change the information on packets so that they are reclassified or even close the SFP;
c) Service Functions (SFs): a component implemented in hardware or software (VNFs) whose objective is to handle the packets received by the forwarding function (SFF). Such operations are performed at any layer of the Internet Protocol (IP) stack;
d) SFC Proxies: responsible for communication between SFC domains and those not part of this architecture (legacy networks), allowing interoperability. The proxies remove or add header information according to the traffic direction (adds in the order from non-SFC to SFC and pull it out in the opposite one).

2.1.5 Considerations about the Section

SFC is essentially formed by applying SDN and NFV to customize traffic flows, aiming at differentiated treatment through routing policies and optimally allocating network resources (firewalls, load balancers, and routers). SDN supports network control, which means how to connect the network elements, and NFV provides the computing resources to them. Hantouti, Benamar, and Taleb (2020) present how SFC uses SDN and NFV to deploy an end-to-end architecture, as shown in Figure 5.

Figure 5 – Service Function Chaining in an SDN and NFV enabled network



Source: adapted from Hantouti, Benamar and Taleb (2020) .

Regarding the SFC placement, there is one example of putting it at the edge (using MEC architecture), for instance, in Meng Wang et al. (2019) or using the service chaining in the core as in Sun and Kim (2016) . This work uses SDN and NFV concepts to deploy an intelligent agent using DRL algorithms at the core network.

2.2 DEEP REINFORCEMENT LEARNING

This section summarizes the basic Deep Reinforcement Learning theory, including the concepts needed to understand it. That includes Reinforcement Learning basics, Markov Decision Processes, and Deep Learning. Afterward, the section discusses Deep Q-Networks (DQN) and two enhancements, Double DQN and Dueling DQN.

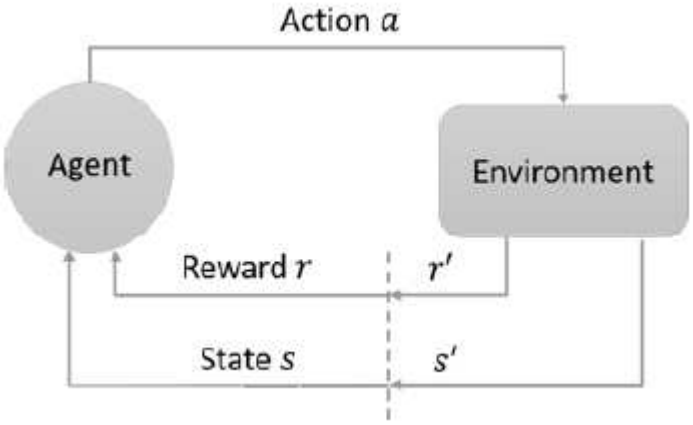
2.2.1 Reinforcement Learning Overview

In Reinforcement Learning (RL), an autonomous agent learns to make the best decisions based on the interaction with an unknown environment to maximize the final cumulative reward. The agent experiments randomly (trial and error) given a set of actions (***a***) at a time step (***t***);

such actions change the state (***s***) of the environment to a new state ($s_t + 1$), and based on their consequences, the agent receives a positive reward (**+*r***) for doing the right thing or a negative reward (punishment) (**-*r***) otherwise.

The agent does not have any information about the action to take. It learns how to act and optimize its behavior from the reward it got from its actions; in other words, it tries to determine the best actions from its own experience. Figure 6 illustrates the general architecture of RL (NGUYEN; REDDI, 2019).

Figure 6 – Reinforcement Learning



Source: NGUYEN; REDDI, 2019

Sutton and Barto (2018) describe the RL system as composed of four basic elements:

a) The **policy** (π): defines the agent's strategy to choose the following action based on the current state. It maps the states perceived from the environment to actions taken in those states. It plays a critical role for an RL agent since the policy selected by the agent dictates the learning behavior;

b) The **reward function**: is a numerical value that the agent receives as immediate feedback based on its action. It defines the goal in a reinforcement learning problem and should make a clear difference between good and wrong actions for the agent (SUTTON; BARTO, 2018);

c) The **value function**: represents the expected return value that an agent gets starting from a state following some policy. It determines the long-term payoff of a state;

d) The **model** of the environment: this is something that represents the behavior of the environment, that allows inferences to be made about how the environment

will behave (SUTTON; BARTO, 2018). When the agent tries to learn the optimal policy with the model dynamics, it is called **model-based** learning. When the agent tries to learn the optimal policy without the model dynamics, it is called **model-free** learning (RAVICHANDIRAN, 2018).

The objective of RL is to choose the optimal action that depends on the current state that will maximize the cumulative reward over the episodes. The cumulative reward R_t is given by the immediate reward r_t plus the expected future rewards discounted by γ at each time step (Equation 1). The factor γ prevents the return from reaching infinity by deciding how important future and immediate rewards are (RAVICHANDIRAN, 2018).

$$R_t = r_t + E[\gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots] = E_{a,s}[\sum_i \gamma^i r_{t+i}]. \quad (1)$$

The expected reward is called the ***value function*** $V(s)$. The value-action, or Q-value, function is the expected reward when the agent starts in state (***s***) and taking an action (***a***):

$$Q(s_t,a_t) = E[\sum_i \gamma^i r_{t+i} | s_t, a_t] \quad (2)$$

Both value and Q-value functions depend on the agent policy π. The policy is the description of how the agent chooses to act at a specific state (PIENROJ; SCHÖNBORN; BIRKE, 2019). According to Mitchell (1997), the agent's objective is to learn a policy π that maximizes rewards or minimizes accumulated penalties after choosing a sequence of actions. The policy π maps states to a distribution of probabilities about actions, π : S $\rightarrow$ p(A = a | S).

2.2.1.1 Markov Decision Processes (MDP)

According to Grasser and Keng (2019) , a Reinforcement Learning problem can be described as a Markov Decision Process (MDP). All MDPs satisfy the Markov property, which indicates that any transition to another state is only dependent on the current state, irrespective of the previous states. The following equation describes the Markov property:

$$Pr\left(s_{t+1} = s, r_{t+1} = r | s_t, a_t\right) \quad (3)$$

According to Wilson and Riccardi (2021) , this property is crucial because all future states and rewards under a control policy π can be known at any given state.

An MDP is defined as a tuple (***S, A, T, R***), where: (***S***) is a set of states, (***T***) is the transition probability function that maps the state-action pair at time (***t***) on the distribution of states at discret-time (***t + 1***) and (***R***) is the function that gives the cost (or reward) for making a decision $\boldsymbol{a} \in \boldsymbol{A}$, when the process is in a state $\boldsymbol{s} \in \boldsymbol{S}$ (PELLEGRINI; WAINER, 2007).

2.2.1.2 Q-learning

A commonly used algorithm in Reinforcement Learning to calculate the Q-value is the Q-learning (WATKINS and DAYAN, 1992) . Q-learning is a model-free algorithm. It estimates the Q-value iteratively: $Q(s,a)$ for each pair (s,a), its goal is to maximize the accumulated reward after the action sequence:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]. \tag{4}$$

Where:

a) $Q(s,a)$ represents the value obtained the last time action a was executed at state s;
b) s is the current state;
c) a is the last action taken by the agent;
d) α is the learning rate used to determine the impact of new information to the existing Q-value;
e) r is the reward obtained after performing action a in state s;
f) γ is the discount factor ($0 \leq \gamma < 1$) and represents the effect of valuing rewards received earlier higher than those received later;
g) $\max_{a'} Q(s',a')$ is the maximum Q-value that can be obtained from the state s, independently of the action chosen;
h) s' is the state reached after performing action a in state s;
i) a' represents a possible action from state s.

The Q-learning updates directly approximate the optimal action-value function through the ‘max’ operator. Once either all Q-values converge or a certain number of iterations is reached, the algorithm will terminate (NGUYEN; REDDI, 2019). The Q-Learning algorithm implementation is illustrated below:

Algorithm 1 – Q-learning

```
Algorithm parameters: step size α ∈ (0, 1], small ϵ > 0;
Initialize Q(s, a), for all s ∈ S+, a ∈ A(s), arbitrarily except that Q(, ·) = 0;
foreach episode do
    Initialize S;
    foreach step of episode do
        Choose a from s using policy derived from Q (e.g., ϵ-greedy);
        Take action a, observe r, s′;
        Q(s, a) ← Q(s, a) + α[r + γ max_a Q(s′, a) − Q(s, a)];
        s ← s′;
    end
end
```

Although Q-Learning has proven to be important in many applications of Reinforcement Learning, it needs to use a table called a Q-table to store the expected rewards (Q-values) of actions given a set of states (tabular method). As the action/state pair increases, the Q-table also increases, requiring more significant amounts of memory and processing, making applying Q-learning prohibitive in some real-world scenarios, including those related to next-generation communication systems.

This characteristic is known as ”state-space explosion”(RESTUCCIA; MELODIA, 2020) or the ”curse of dimensionality”(XIONG et al., 2019). To overcome this issue, Deep Reinforcement Learning combines the use of Deep Learning through Artificial Neural Networks (ANN) to approximate the values of Q-function, and Reinforcement Learning to analyze the rewards received through each action in different states (XIONG et al., 2019).

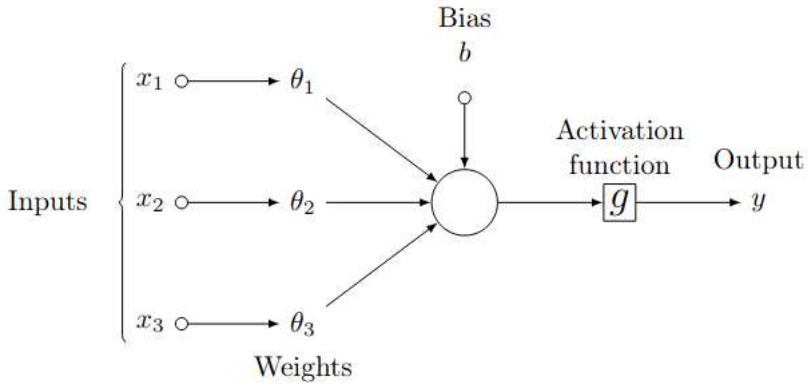
2.2.2 Deep Learning Overview

According to LeCun, Bengio e Hinton (2015), Deep Learning is a set of algorithms and techniques that allow computational models that use multiple processing layers to learn representations of data with various levels of abstraction. The objective is to avoid manual description of features in a data set by automatic learning from data (NGUYEN; REDDI, 2019). The most common method to deploy Deep Learning models is by using Artificial Neural Networks (ANNs).

These ANNs use artificial neurons to create a mathematical model to simulate human brain behavior. The goal of the ANNs is to approximate a function f that describes the input and output data relationship. Every neuron has adjustable weighted inputs (θ) (that simulate the synapses), an activation function (defines the output given an input), and one output (WONG;

LEUNG; FIELD, 2021). Figure 7 shows a simple model of an artificial neuron as proposed by McCulloch e Pitts (1943):

Figure 7 – Artificial Neuron



Source: Wong, Leung and Field .

The output of the artificial neuron represented in Figure 7 can be calculated by following equation:

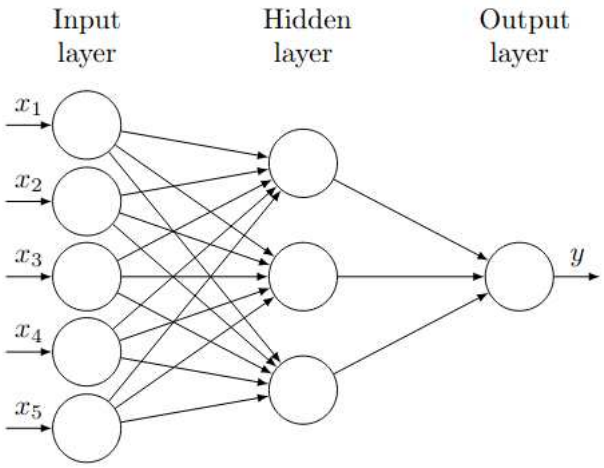
$$y = g(\theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + b) \tag{5}$$

Where:

a) y is the output signal;
b) g is the activation function;
c) θ_i is the weight that will multiply the respective x_i;
d) x_i is the input signal;
e) b is the bias.

The ANN architecture is organized into interconnected layers: input, hidden, and output. The input layer contains the neurons that send data to the hidden layer, and then this layer sends information to the output layer. The number of hidden layers is specific to each model depending on the problem complexity. Every neuron receiving multiple inputs takes a weighted sum of them, passes it through an activation function, and responds with an output. Deep Learning uses two or more hidden layers to train the model to learn internal data patterns using multiple processing (hidden) layers. The terminology Deep Neural Network (DNN) is used in this case. Figure 8 illustrates a simple ANN architecture:

Figure 8 – Artificial Neural Network (ANN)



Source: Wong, Leung and Field .

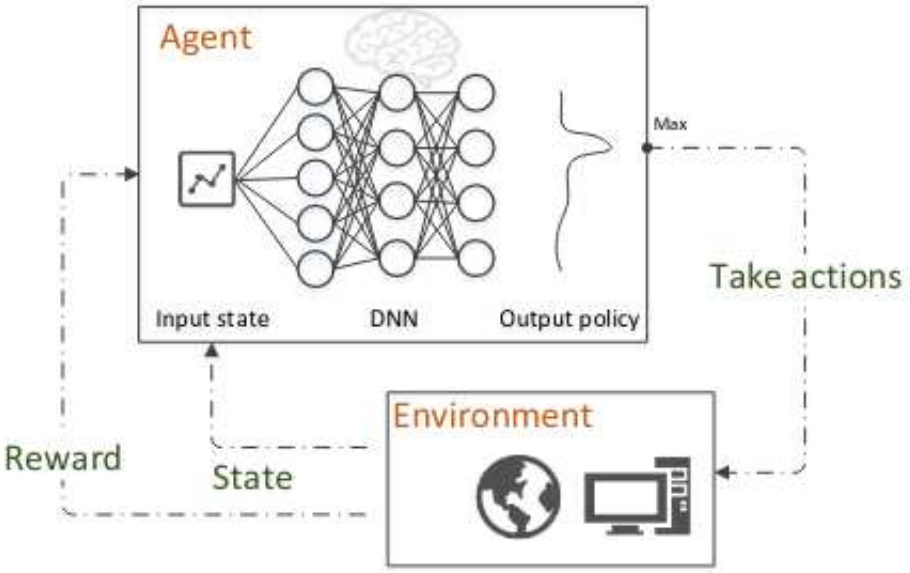
To learn how to adjust neurons' weights for a multilayer network, ANNs use a method called backpropagation. It employs a gradient descent optimization algorithm to try to minimize the squared error between the network output values and the target values for these outputs as described in Mitchell (1997).

2.2.3 Deep Q-Networks (DQN)

The Q-learning algorithm has been used as an efficient mechanism to reach the optimal policy when the state space and action space are small. However, in most scenarios in the communications systems, these spaces are usually large. Then, finding find the optimal policy may be computationally infeasible for the Q-learning algorithm. (NGUYEN; REDDI, 2019).

Mnih et al. (2015) presented a variation of Q-learning to deal with this shortcoming. The method implements a Deep Q-Networks (DQN) using Deep Neural Networks (DNN), instead of the Q-table to derive an approximate value of $Q^*(s,a)$ (NGUYEN; REDDI, 2019). Figure 9 shows the DQN. This combination of Deep Neural Networks (inherited from Deep Learning) and Reinforcement Learning is called Deep Reinforcement Learning (DRL).

Figure 9 – Deep Q-Networks



Source: (XIONG et al., 2019)

According to Guo et al. (2019), two characteristics make DQN different from the other algorithms:

a) Experience Replay: consists of adding a replay buffer, whose function is to save the agent's previous experiences. This memory contains a collection of experience tuples (S, A, R, S'). Such tuples are gradually added to memory in each interaction with the environment. The goal is to store past experiences and then use a random subset of those experiences to update the network, rather than just using the most recent experience;

b) Target Network: a separated neural network Q^* is used in an attempt to keep the algorithm more stable. A copy of the neural network is maintained in this separated network, and it is used for the value $Q(s',a')$. In this way, the Q-values predicted in this second network are used to backpropagate and train the main network. An essential part of this process is that the Target Network parameters are not updated but periodically synchronized with the network parameters of the main one. The goal is to use Target Network Q-values to train the main network and improve training stability.

Algorithm 2 presents the Deep Q-Learning (DQL) algorithm with experience replay and fixed target q-networks. The use of experience replay is one of the main concepts in Mnih et al.

(2015) work. This method stores the agent's experience (transitions between states in the past and the corresponding actions and rewards) at each recorded time step $e_t = (s_t, a_t, r_t, s_{t+1})$ in a replay memory to calculate the loss correctly in the future at any time step. Mini-batches of experiences are then sampled randomly and used to perform the weight updates for the DQN training.

Algorithm 2 – Deep Q-Learning with experience replay and target network (DQN)

Init: replay memory D with capacity M, Q_θ with random weights, $\theta^- = \theta$
for *a number of episodes* **do**
  initialize s_0
  while *episode not done* **do**
    take action $a_t \sim \pi(s_t)$, observe r_t and s_{t+1} `// π can be ϵ-greedy for instance`
    store transition (s_t, a_t, r_t, s_{t+1}) in D
    sample random mini-batch of transitions $((s_j, a_j, r_j, s_{j+1}))_{j=1,\dots,N}$ from D
    set $y_j = \begin{cases} r_j & \text{if } s_{j+1} \text{ is a terminal state} \\ r_j + \gamma \max_{a'} Q_{\theta^-}(s_{j+1},a') & \text{otherwise} \end{cases}$
    $\theta \leftarrow \theta - \alpha \nabla_\theta \frac{1}{N} \sum_{j=1}^{N} (y_j - Q_\theta(s_j,a_j))^2$
    every k steps, update $\theta^- = \theta$
  end
end

There are several advantages in using experience replay over traditional Q-Learning. The network can use every stored experience for other updates, making the learning process more efficient. This mechanism reduces the variance of the updates by eliminating the correlation between these samples. It has been shown that such a mechanism improves training stability. The following sections discuss two DQN enhancements, the Double DQN and the Dueling DQN

2.2.4 Double DQN

DQN uses the ***max*** operator to estimate the Q-value of the next state-action pair, and this information is based on the same values both to select and evaluate action. This over-estimation of Q-value can cause poor performance of Q-learning. To overcome this issue, Hasselt (2010) presented the Double Q-learning.

Double Q-learning implements two Q functions (Q^A and Q^B) in the target value computations, each of which is used to update the other for the next state. Algorithm 3 illustrates this process.

Algorithm 3 – Double Q-Learning

Init: Q^A, Q^B, s
repeat
  Choose a, based on $Q^A(s, .)$ and based on $Q^B(s, .)$, observe r, s'
  Choose (e. g. random) either Q^A or Q^B to update
  if *update* Q^A **then**
    Define a^* = argmax$_a$Q$^A(s',a)$
    $Q^A(s, a) \leftarrow Q^A(s, a) + \alpha(s, a)(r + \gamma Q^B(s', a^*) - Q^A(s,a))$ **else if** *update* Q^B **then**
      Define b^* = argmax$_a$Q$^B(s',a)$
      $Q^B(s, a) \leftarrow Q^B(s, a) + \alpha(s, a)(r + \gamma Q^A(s', a^*) - Q^B(s,a))$
    end
  end
  $s \leftarrow s'$
until *end*

Double Q-learning uses tabular methods, so it has the same drawbacks as methods as Q-learning (the state-space explosion). Then, Van Hasselt, Guez e Silver (2016) proposed the Double DQN that uses two different Deep Neural Networks: the Deep Q-Network (DQN) and the Target Network, to update the parameters. Double DQN can generalize the Double Q-learning algorithm to work with arbitrary function approximation, including Deep Neural Networks. The objective is to get the most benefit of Double Q-learning and keep the DQN with minimal changes.

2.2.5 Dueling Q-learning

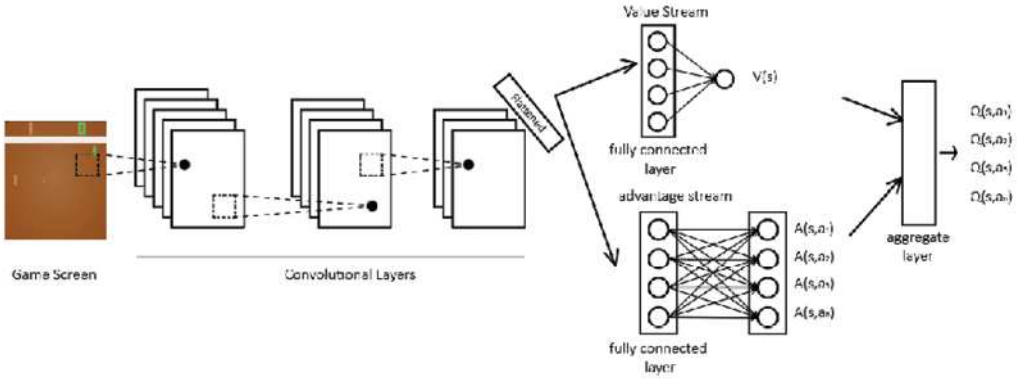
Using Enduro (the Atari game) as an example, Ziyu Wang et al. (2016) noted that for many MDPs, it is unnecessary to estimate each action taken at every time step. For many states, the choice of action has no impact on what happens. Inspired by this idea, they introduced the dueling network architecture that explicitly separates the representation of state values and (state-dependent) action advantages using two different streams to estimate the values for each action. The dueling architecture splits the Q-values in the value function $V(s)$ and the advantage function $A(s,a)$.Equation 6 represents how to compute the Q-values using this method:

$$Q(s, a) = V(s) + A(s, a) \tag{6}$$

What the Dueling DQN algorithm (WANG, Z. et al., 2016) proposes is that the same neural network splits its last layer into two parts, one of them to estimate the state value function for state s (V(s)) and the other one to evaluate the advantage function for each action $a(S(a,a))$.

In the end, it combines both parts into a single output that estimates the Q-values. Figure 10 presents the architecture of a Dueling DQN.

Figure 10 – Dueling Q-learning



Source: (RAVICHANDIRAN, 2018)

The two streams are combined via a particular aggregating layer to estimate the state-action value function Q, as shown on the right side of Figure 10. Both streams share a common convolutional feature learning module. The last module uses the following mapping:

$$Q\left(s,a,\theta,\alpha,\beta\right)=V\left(s,\theta,\beta\right)+\left(A\left(s,a,\theta,\alpha\right)-\frac{1}{|\mathcal{A}|}\sum_{a'}A\left(s,a';\theta,\alpha\right)\right) \tag{7}$$

Where:

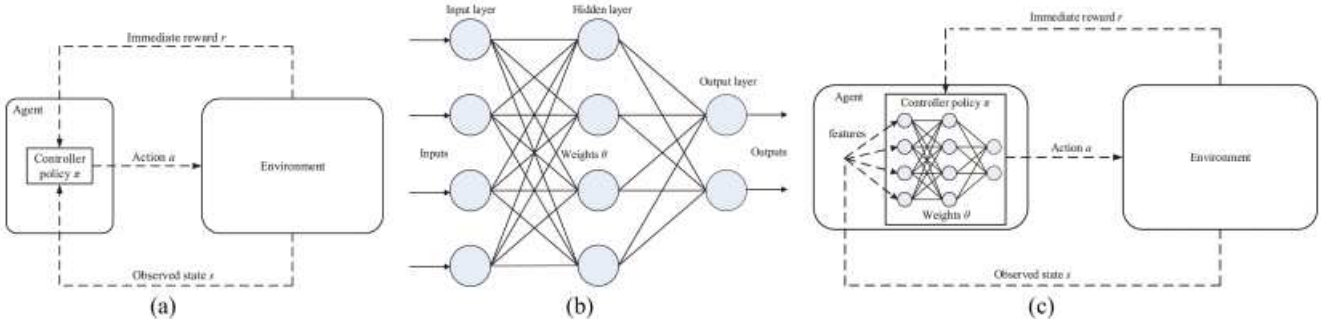
a) θ represents the parameter value of the convolutional network;
b) α is the parameter of the value stream;
c) β is the parameter of the advantage stream;
d) A represents the length of the action space.

According to Ziyu Wang et al. (2016), the advantage of the dueling architecture is its capacity to learn the state-value efficiently. In experiments, the authors demonstrated that the proposed method could more quickly identify the correct action during policy evaluation as redundant or similar actions are added to the learning problem.

2.2.6 Comments about the Section

This section presented the essential background to understanding Deep Reinforcement Learning, especially the DQN and its enhancements (Double DQN and Dueling DQN). The application of Reinforcement Learning using Deep Learning (through Artificial Neural Networks) to approximate the Q-value $Q^*(s,a)$, results in Deep Reinforcement Learning methods and their variations. The relationship between RL, DL, and DRL is illustrated in Figure 11.

Figure 11 – (a) Reinforcement Learning, (b) ANN, and (c) Deep Q-learning



Source: (NGUYEN; REDDI, 2019)

This work chose the DQN and its variants due to the research done for Chapter 3 (Related Work), where it could see that the DQN is the most algorithm used. Table 1 below the DRL algorithms used by the researchers in their works.

Table 1 – Algorithms Used by the Authors

Algorithm	Papers
Deep q-networks	104
Deep Deterministic Policy Gradient	43
A3C	31
Double Deep Q-Learning	19
A2C	10
REINFORCE	9
Dueling Deep Q-Learning	8

Source: the Author.

3 RELATED WORK

Applying Deep Reinforcement Learning techniques in emerging networks, such as 5G networks, has been the subject of several kinds of research by the academic community. This chapter will present some efforts of other authors that underpin this research. It divides into three sections: the first will present the works related to the application of DRL techniques to traffic classification, the second those about the application of DRL to the SFC, in the third, those associated with the theme of applying the SFC in traffic classification.

The material used in this chapter was obtained by searching in Google Scholar using the terms: Deep Reinforcement Learning and Service Function Chaining (or Chain), from 2018 to 2021. The search resulted in 215 articles after the following selection criteria: (i) articles not written in English or Portuguese were discarded, and (ii) the selected sources: ACM, IEEE, SBC, Elsevier, Springer, and Wiley. Table 2 shows the articles by the publication source:

Table 2 – Papers per Publisher

Source	**Papers**
ACM	4
Elsevier	28
IEEE	156
SBC	1
Springer	15
Wiley	11
Total	**215**

Source: the Author.

Table 3 below shows the published papers by year and demonstrates the academic community growing interested in the topic proving this work relevance:

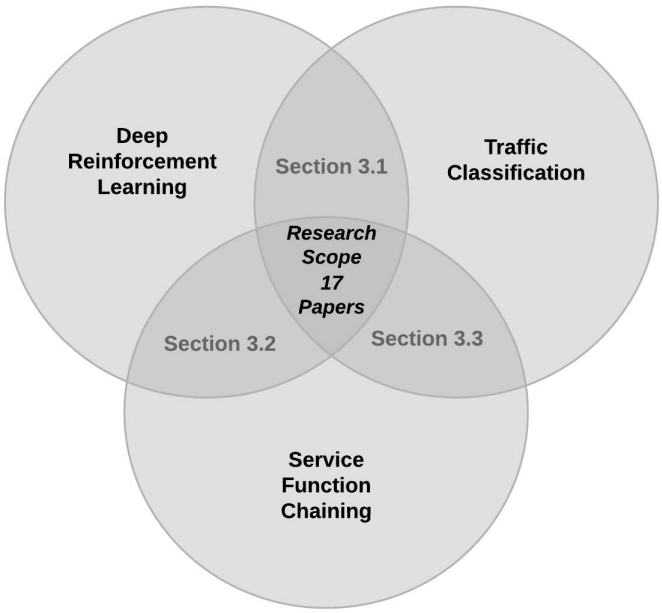
Table 3 – Papers per Publication Year

Year	**Papers**
2018	3
2019	26
2020	74
2021	112
Total	**215**

Source: the Author

After analyzing the keywords in the 215 papers, the following step consisted of finding articles that covered traffic classification and then exploring the references cited by the papers according to the number of citations mentioned in Google Scholar. The goal was to verify the relevance of the selected articles. This refinement resulted in 17 works. Figure 12 below illustrates the relationship between topics and sections in this chapter, where the intersection of the three previous themes establishes the scope of this research. The following sections discuss the relevant papers that supported this work.

Figure 12 – Research Scope



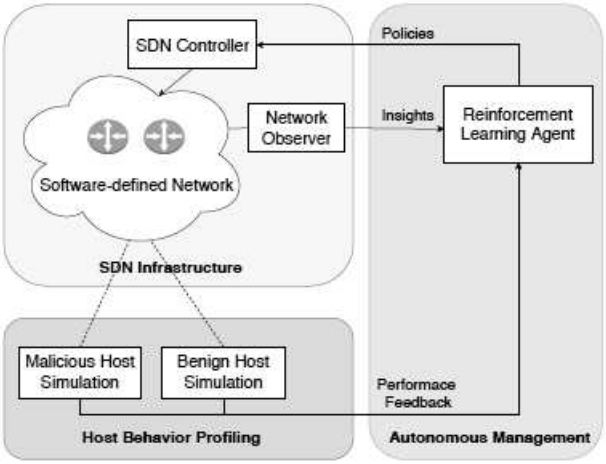
Source: the Author

3.1 DEEP REINFORCEMENT LEARNING APPLIED TO TRAFFIC CLASSIFICATION

Akbari et al. (2020) used the Neural Fitted Q-learning (NFQ) algorithm (RIEDMILLER, 2005) to implement an autonomous threat mitigation architecture in SDN networks (Autonomous Threat Mitigation in SDN using Reinforcement Learning – ATMoS). They provided definitions of how to formulate threat mitigation as a Reinforcement Learning problem, which they said is one of the main challenges in network security.

The architecture has three modules: the SDN infrastructure that fits the network observer, implemented with Snort (ROESCH, 1999), responsible for monitoring network traffic and providing information about the state of the network for the autonomous management module. This module is the solution intelligence, composed of the Reinforcement Learning agent that executes API commands to implement the policies in the network via the SDN controller. The framework created by the authors can be seen in Figure 13:

Figure 13 – ATMoS Architecture



Source: AKBARI et al., 2020

Such policies define the path that traffic from a given host must take, with the options (i) being less restricted and (ii) more restricted security. The third module, called host behavior profiling, accommodates two types of hosts: those that send benign traffic and those that send malicious traffic. The results presented the number of iterations for the NFQ algorithm convergence as a function of the number of hosts involved. They showed that the solution could be used for attack mitigation. The great advantage of the work was its conception based on open source tools and its availability on the Github site for the reproduction of the experiments. This work adapted the SDN paradigm to use the SFC, which the authors did not cover.

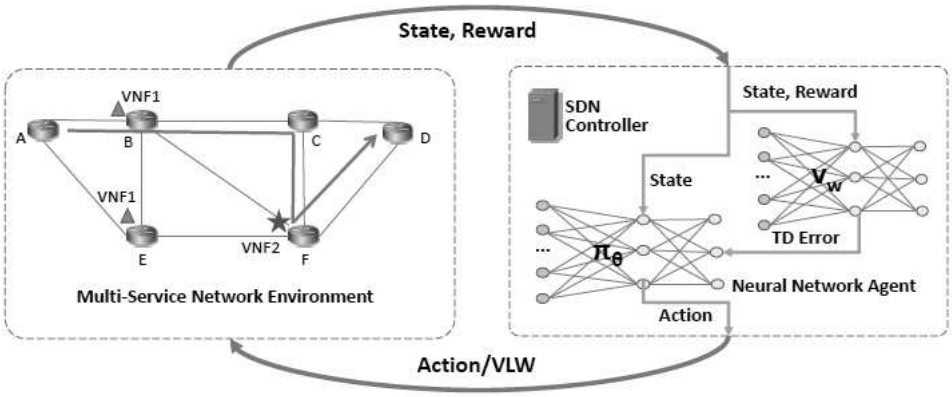
Liang et al. (2019) proposed a packet classification system called ”Neural Packet Classification”to deal with the limitations of methods based on heuristics. The first limitation mentioned was the ”manual”adjustment for the construction of decision trees. Decision trees provide good accuracy in classifying packets, but with the application of Deep Reinforcement Learning, they can be more efficient. The second limitation is related to the fact that heuristics do not explicitly optimize the objective function.

3.2 DEEP REINFORCEMENT LEARNING AND SERVICE FUNCTION CHAINING

Zhang et al. (2018) presented a framework for provisioning service chains (SFCs) that combine SDN and NFV to provide the most efficient use of network resources and is promising for emerging technologies like 5G systems and virtual network services in data centers. In addition to applying dynamic resource provisioning mechanisms, the survey also mentioned the open challenges and opportunities. One of them that motivated this work was the possibility of using DRL in allocating paths and forwarding traffic autonomously. Although they did not mention how to do it, the concepts presented helped adapt and apply the SFC in the test scenario of this research.

Ning, Wang, and Tafazolli (2020) proposed a network model to solve the SFC optimization problem in a multiservice environment. They introduced a method based on Deep Reinforcement Learning to obtain optimized operations by adapting the issue as a Markov decision process, as seen in Figure 14.

Figure 14 – Multiservice Environment



Source: NING et al., 2020

They conceived the network as a directed graph, where the VNFs are the vertices, and the links between them compose the edges. That is crucial to adapt the objective functions that base the optimization of SFC flows: Maximum Link Usage (MLU), Maximum Path Delay (MPD), and VNF Maximum Usage (VMU). They obtained the data to evaluate the model from a real network, the GÉANT project. The author compared the data with those resulting from the Mixed-Integer Linear Programming (MILP) method (SMITH; TASKIN, 2008). The DRL-

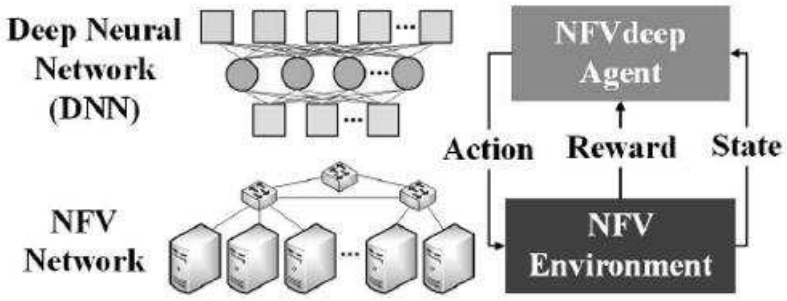
based solution achieved an almost ideal performance compared to MILP and proved better when compared to traditional solutions (without learning mechanisms).

He et al. (2020) proposed an intelligent provisioning framework for VNFs to optimize resource scheduling in a cloud environment called Cloud of the Things (CoT). A traffic identification mechanism based on Deep Learning was combined with a VNF path selection using Deep Reinforcement Learning to improve the Actor-Critic algorithm using Kronecker-Factored Trust Region (ACKTR) (WU et al., 2017). They performed the simulation in two phases: the first was to assess the accuracy of traffic identification, where they used a dataset with eight types of real traffic classes got in Sharafaldin, Lashkari e Ghorbani (2018). Using the mGBDT (Multi-Layered Gradient Boosting Decision Trees) algorithm (FENG; YU; ZHOU, 2018), they compared it with others (k-nearest neighbors, Decision Trees, and Random Forests).

They concluded that it had the best precision rate. The second part consisted of provisioning the VNFs using Pytorch to build the DRL algorithms test architecture. They demonstrated that the proposed solution had superior performance than other DRL algorithms, improving the users' Quality of Service (QoS).

A limitation in applying DRL in a dynamic SFC implementation, according to Xiao et al. (2019), is to assume that network resource requests are predetermined, regardless of real-time variations. The authors' proposal consists of NFVDeep, an adaptable, online, and Policy Gradient-based approach that treats dynamic network state transitions as a Markov decision process based on Peters and Bagnell (2010) . In this way, the SFC requests are automatically implemented according to their QoS characteristics. They created a testbed architecture where the environment from a DRL perspective was the NFV network topology, as illustrated below:

Figure 15 – NFVDeep



Source: XIAO et al., 2019

The simulations showed that the architecture significantly outperforms the solutions found in other research. The strength of the article was to demonstrate how a dynamic network environment, such as those found in NFV, can be mapped as a Markov decision process, in addition to using the SFC paradigm to solve the issue of dynamic resource allocation such as processing load and bandwidth.

3.3 SERVICE FUNCTION CHAINING APPLIED TO TRAFFIC CLASSIFICATION

Shin and Kwon (2017) proposed a technique to execute the SFC flexibly to manage traffic adaptively, with its real-time monitoring according to a method known as Recyclable Counter with Confinement (RCC) (NYANG; SHIN, 2016). The RCC consists of a counter that aims to aggregate the large-scale traffic flow in real-time, resulting in optimized use of device memory. The SFC allows allocating virtualized resources (network functions) on-demand.

They applied the RCC method into the SFC classifier module, and thus the architecture was divided into three modules: classification, RCC, and encapsulation. They tested three network functions to verify the functioning: Firewall (FW), Deep Packet Inspection (DPI), and Load Balancer (LB). Depending on the amount of traffic generated, the packets travel along a path, Figure 16(a). According to pre-defined thresholds that define an anomalous behavior, they go through the other route, Figure 16(b). In this way, the packet routing decision process can be done differently according to its characteristics.

Figure 16 – SFC Paths by SHIN and KWON, 2017

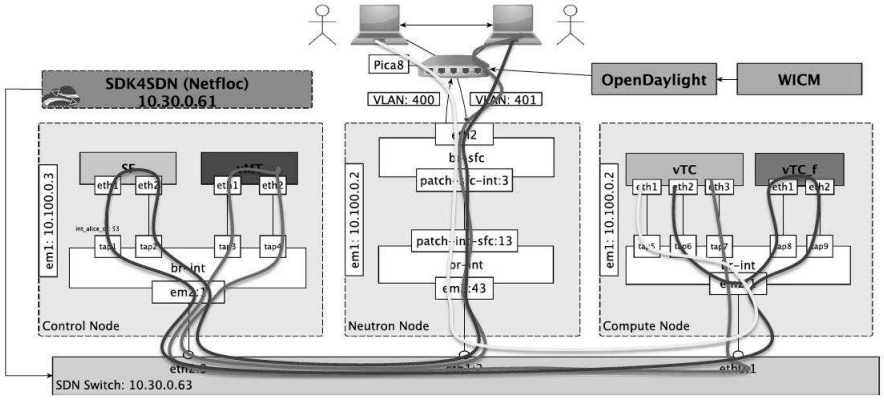
(a) SFC1 (b) SFC2

Source: SHIN and KWON, 2017

In Trajkovska et al. (2017), the authors designed an SFC-based traffic routing solution that used the concepts of SDN and NFV using open-source solutions to deploy the components, a valuable opportunity in terms of implementation cost and experiment reproducibility. They used OpenStack and OpenDayLight as solutions to build the architecture to improve packet

processing performance and optimize network resources. Figure 17 shows the testbed topology.

Figure 17 – Testbed Architecture by TRAJKOVSKA et al., 2017



Source: TRAJKOVSKA et al., 2017

They performed extensive traffic tests with the following protocols: Internet Control Message Protocol (ICMP), Transmission Control Protocol (TCP), User Datagram Protocol (UDP), and video traffic to assess: (i) the effectiveness and accuracy of the traffic routing in a working NFVI environment, (ii) the robustness of the Service Functions Chaining configuration, (iii) the efficiency of traffic redirection and packet handling and (iv) the scalability of the implementation.

Experimental evaluations have shown that the SFC, with its ability to initialize dynamically, can reconfigure and relocate VNFs without the need to install new hardware. Also, SFC enables the implementation of an efficient mechanism for delivering end-to-end traffic through VNFs in a cloud environment, differentiating traffic based on protocol type and profile according to its requirements for quality of service parameters (such as packet loss, delay, and jitter).

3.4 FINAL CONSIDERATIONS FOR THIS CHAPTER

This chapter presented a summary of the works related to this research that helped find open opportunities in decision-making to address the topic of the application of Deep Reinforcement Learning in the Service Function Chaining for packet routing.

For example, Akbari et al. (2020) used the concept of Software Defined Networks, which can be adapted in the SFC environment, which the authors did not address. Liang et al. (2019)

concluded that DRL could make the package classification task more efficient, but they did not perform tests in the SFC environment. Zhang et al. (2018) and He et al. (2020) helped elucidate how to apply the SFC to organize network functions, route packets through them, and adapt the topology of the present work.

The issue of modeling an SFC-based network as a DRL problem was the subject of Ning, Wang, and Tafazolli (2020). However, traffic routing was not the subject of the research. In Xiao et al. (2019), the QoS parameters were the authors' focus, and this work has adapted aiming at configurations of routing policies based on the application type. Other articles have proposed efficient methods of applying SFC for traffic classification and routing, as in Shin and Kwon (2017) and Trajkovska et al. (2017). However, they did not use machine learning mechanisms, an opportunity for improvement that the present work used to make networks more autonomous.

A decisive topic in the choice of the works presented in this chapter was open-source software to elaborate their test platforms. As can be seen in Akbari et al. (2020) and Trajkovska et al. (2017) this is very important to enable the practical implementation of the research and a possible lab demonstration.

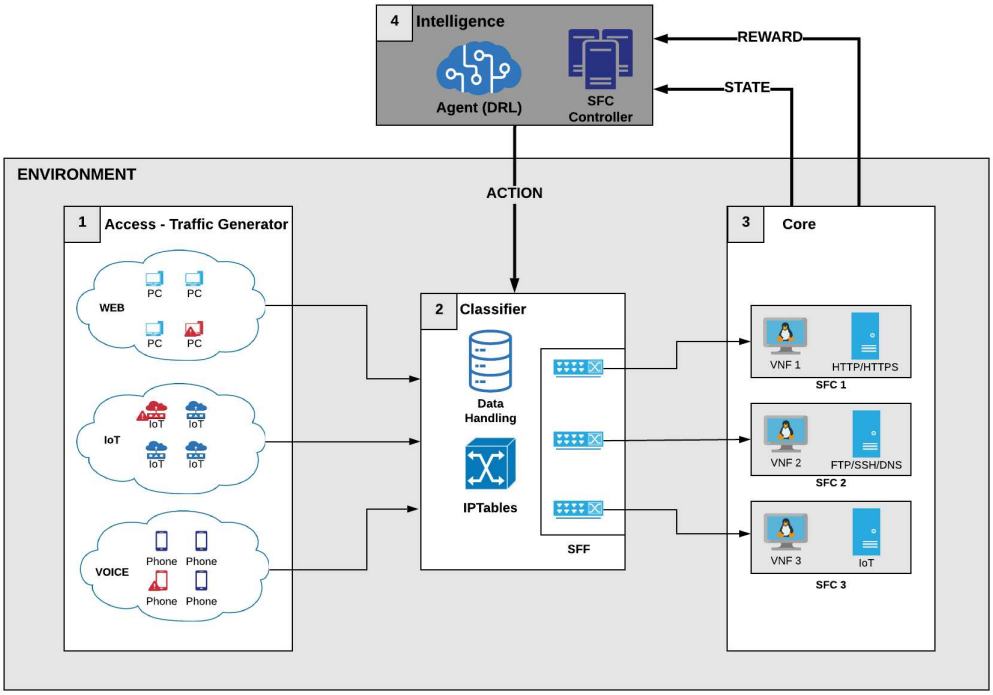
4 PROPOSED ARCHITECTURE

This work investigates combining Deep Reinforcement Learning techniques with the Service Function Chaining architecture, using the references discussed in Chapter 3. It aims at providing a mechanism for identifying and routing traffic based on profiles by forwarding the packets to the correct path according to decisions made by a learning agent. It performs through a practical implementation on a test platform designed with open-source tools. Subsequent sections describe the proposed model and tools used.

4.1 PROPOSED TESTBED

Figure 18 shows the proposed model. It consists of four modules described in the following sections.

Figure 18 – Proposed Model



Source: the Author

4.1.1 Module 1: Access - Traffic Generator

This module is responsible for simulating the access layer of a telecommunications network, such as those used in mobile networks of fourth/fifth generations (4G and 5G) and broadband networks. It generates network user IP (Internet Protocol) traffic for multiple applications to create and send the necessary information (dataset) to the ***classifier*** module.

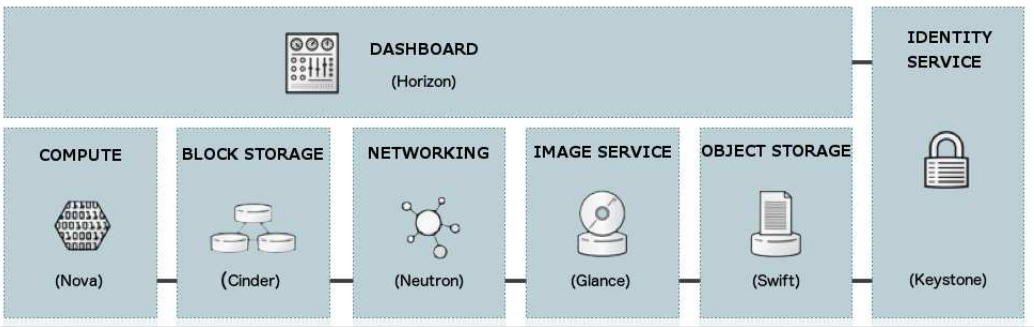
Here, Virtual Machines (VMs) based on Linux have been deployed on the OpenStack platform to execute the software nping to generate IP packets. The objective is to simulate a cloud environment and to save resources. The following subsections give an overview of the tools used in this module.

4.1.1.1 Openstack Platform

OpenStack (OPENSTACK, 2021) is a free and open-source software platform for cloud computing, mainly deployed as an Infrastructure-as-a-Service (IAAS). It controls large pools of computing, storage, and networking resources throughout a data center, all managed and provisioned through APIs with authentication methods. A Graphical User Interface (GUI) dashboard is available, giving administrators control while providing tools to provide resources through a web interface. It also has a Command Line Interface (CLI).

It is the tool used to deploy virtual machines (containers are also possible) and the networking needed to connect them. Beyond standard IAAS functionality, additional components provide orchestration, fault management, and service management amongst other services to ensure the high availability of user applications. Figure 19 illustrates the OpenStack architecture.

Figure 19 – OpenStack Architecture



Source: https://docs.openstack.org/security-guide/introduction/introduction-to-openstack.html

Table 4 describes the main components:

Table 4 – OpenStack Main Components

Component	Function
Nova	In charge of VM creation and computing resources allocation
Glance	Image management
Neutron	Network management
Keystone	Authentication management
Horizon	The dashboard for OpenStack management
Cinder	Provides persistent block storage for VMs
Swift	Provides support for storing and retrieving arbitrary data in the cloud

Source: the Author, adapted from www.openstack.org

The complete OpenStack installation is sometimes cumbersome, and it is beyond the scope of this work. However, it is possible to check an all-in-one installation option (used in this work): https://www.rdoproject.org/install/packstack/.

4.1.1.2 nping

Nping (NPING, 2021) is a command-line oriented packet assembler and analyzer. The ping command inspires the software, but ping can only send ICMP echo requests. On the other hand, nping supports TCP, UDP, and ICMP. The Linux command below shows how to install the nping on an Ubuntu machine:

$**sudo apt-get install nping**

Some examples of using nping to generate packets is given below:

$**nping -c [number of packets] –[transport protocol] -p [destination port] [target host]**
$**nping -c 10 –tcp -p 80 www.fei.edu.br** (to simulate HTTP packets)
$**nping -c 10 –tcp -p 443 www.fei.edu.br** (to simulate HTTPS packets)
$**nping -c 10 –tcp -p 5060 10.10.10.5** (to simulate SIP packets)

4.1.2 Module 2: Classifier

This module performs the packet classifier function in the context of the SFC and selects the appropriate forwarder (SFF), and tags the packet according to proper rules. The ***classifier*** module receives the traffic generated by the ***access module***. It assigns an SFC, creating a tuple (destination port, SFC) where the SFC must be set according to Table 5. This tuple is sent to the next module, the ***core***.

Table 5 – Traffic Type per SFC

Traffic Type	Protocol	SFC Traffic Profile
WEB	HTTP/HTTPS	1
Management	FTP/SSH/DNS/NTP/SNMP	2
Other	Other	3

Source: the Author

Within the ***classifier***, there are three functions:

a) Classifier: the tool used was the Linux iptables. The advantage of this approach lies in its ability to analyze the network, and transport layers of the Open Systems Interconnection (OSI) model (ZIMMERMANN, 1980);
b) Handling: This process captures the packets with the TCPDUMP tool, generating a file in *pcap* format. A pre-processing is performed to prepare the file and send it in a format understandable by the ***intelligence*** module (agent). It converts the pcap format into a Comma-Separated Values (CSV) file;
c) Forwarder (SFF): As seen in the theory chapter, the forwarder module is responsible for forwarding packets to the appropriate network functions (SFs). In this case, a Layer 2 Switch meets the need. The Linux operating system already has this feature internally.

The following subsections provide a short description of the tools used in this module.

4.1.2.1 IPtables

IPtables (IPTABLES, 2021) is a tool to set up, maintain, and inspect the IP packets filter rules in the Linux kernel. Several tables with different policies may be defined to send or drop the IP packets to the external network. It is also possible to change some internal information

inside the IP packet. It is software that comes with standard Linux distributions without installing. Some examples of rules are:

To accept HTTP packets:

$**iptables -A INPUT -i eth0 -p tcp –dport 80 -m state –state NEW,ESTABLISHED -j ACCEPT**

$**iptables -A OUTPUT -o eth0 -p tcp –sport 80 -m state –state ESTABLISHED -j ACCEPT**

To accept HTTPS packets:

$**iptables -A INPUT -i eth0 -p tcp –dport 443 -m state –state NEW,ESTABLISHED -j ACCEPT**

$**iptables -A OUTPUT -o eth0 -p tcp –sport 443 -m state –state ESTABLISHED -j ACCEPT**

To reject SIP packets:

$**iptables -I INPUT -p udp -m udp –dport 5060 -j DROP**

4.1.2.2 TCPDUMP

Tcpdump (TCPDUMP, 2021) is a network capture tool for protocol analysis, also called a network packet sniffer. It helps to monitor the traffic in IP networks by intercepting the data packets as they go through the network interface card. One of its advantages is passive monitoring, which does not change the packet information.

It is available under most Linux/Unix-based operating systems through the command-line interface. Tcpdump also gives an option to save captured packets in a file for future analysis. It keeps the file in a pcap format, and it is possible to convert it into a CSV file. To install tcpdump:

$**sudo apt-get install tcpdump**

Some example of commands to capture packets:

To capture 50000 packets and save in a file (master.pcap):

$**tcpdump -w master.pcap -i eth0 -c 50000**

To capture http packets:

$**tcpdump -i eth0 port 80**

To capture packets from source IP address 192.168.0.2:

$**tcpdump -i eth0 src 192.168.0.2**

4.1.3 Module 3: Core

This module has two sub-modules: Packet Processing and Get Reward. The first one calculates the percentage of packets processed successfully based on the SFC assigned by module 2. The latter calculates the rewards, whose mechanism is explained in the next section. Module 3 is also responsible for simulating the VNFs that receive and processes the packet sent by the ***classifier***. It also provides and sends the information (states and rewards) to the ***intelligence*** Module.

At the beginning of the experiments, this work sought to create a mechanism that would allow the interaction between the VMs that terminate the traffic (Service Functions) and the agent (intelligence module). However, it was unsuccessful in providing information to the agent in real-time. Then, it was adapted as Python (VAN ROSSUM; DRAKE, 2009) code to allow the experiments to keep running.

4.1.4 Module 4: Intelligence

This module is the agent from the Deep Reinforcement Learning perspective, whose function is to run the Deep Q-Networks (DQN) algorithms. It calculates and sends the action to the ***classifier*** module to move the packet to a new SFC. The proposed model is described as a DRL problem in the next section. The tools used here were: Python, OpenAI Gym, PyTorch, Matplotlib, and Pandas.

4.1.4.1 OpenAI Gym

According to Brockman et al. (2016), the OpenAI Gym is an open-source Python library for developing and comparing reinforcement learning algorithms by providing to the developers

a standard API to communicate between learning algorithms and environments (standard or customized). This work chose this tool because it is easy to deploy a customized environment used for the experiments.

4.1.4.2 PyTorch

It is open-source and provides tools to implement Deep Learning algorithms. Created by Paszke et al. (2019), PyTorch is a computational tensor library that takes advantage of using Graphics Processing Unit (GPU). It is used for broad applications, such as Natural Language Processing and Computational Vision.

4.1.4.3 Matplotlib

Matplotlib (HUNTER, 2007) is a library that allows the creation of interactive visualization and plotting graphs in Python. And like the other tools used in this work, it is open source.

4.1.4.4 Pandas

Pandas is a Python library for data analysis and manipulation, and it helps Python to deal with more complex data structures, such as CSV files and Excel sheets (TEAM, 2020; MCKINNEY, 2010). It is a tool used to handle a high volume of data. This work used it to help with dataset cleaning, feature selection, and processing.

4.2 DATASET OVERVIEW

The dataset for this research was created by combining the tools described in subsections 4.1.1 and 4.1.2. Nping generated the packets. This application runs on top of virtual machines. To capture the packets, tcpdump performed the function. Afterward, the pcap file was converted to CSV. The dataset consists of 50000 IP packets (raw data) with several types of traffic. The destination port (dst.port) was used to define the application type in IP networks. Table 6 shows the distribution of packets per application type. Table 7 shows the distribution per profile as defined in the classifier (WEB traffic, management traffic, and other traffic).

Table 6 – Packets per Application

Application Type (PORT)	**Packets**
HTTP (80)	5319
HTTPS (443)	5174
FTP (21)	40
SSH (22)	39
DNS (53)	10765
NTP (123)	26
SNMP (161)	40
Other	28597
Total	**50000**

Source: the Author.

Table 7 – Packets per SFC Profile

Application Type (PORT)	**Packets**
WEB	10493
Management	10910
Other	28597

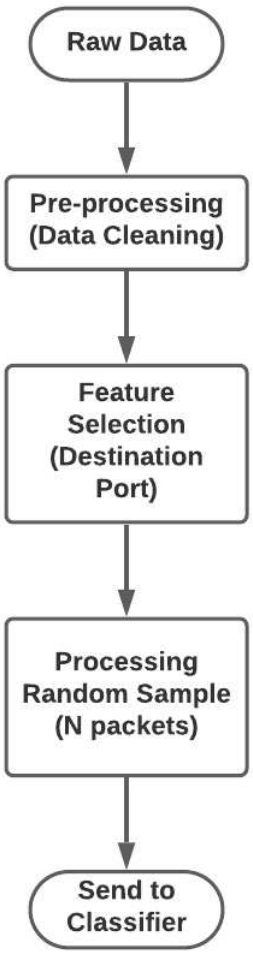
Source: the Author.

The application traffic should be assigned an SFC path according to the policies received as actions from the ***agent*** by the ***classifier*** module. Packets with dst.port = 0 or dst.port = 1 were deleted (data cleaning), since they do not represent a practical application. This work followed the process shown in Figure 20 to handle the dataset.

4.3 PROPOSED MODEL AS DEEP REINFORCEMENT LEARNING PROBLEM

In the proposed architecture, the ***environment*** from a Deep Reinforcement Learning perspective is composed of network elements organized according to the Service Function Chaining (SFC) architecture, including the processing nodes, communication links, and Virtual Network Functions (VNFs).

Figure 20 – Dataset Handling



Source: the Author

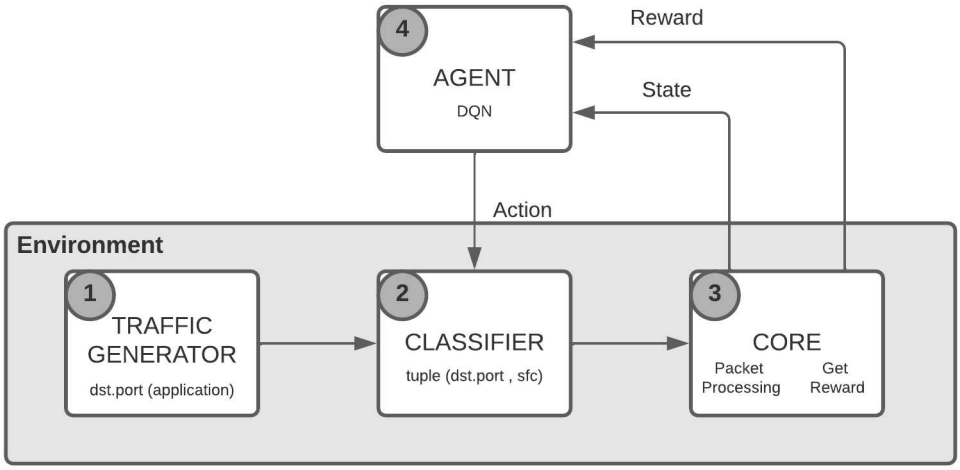
4.3.1 State Space

When a new episode starts, the ***classifier*** sends the packets based on tuple (dst.port, SFC) to the ***core*** module, and then this module calculates the percentage as follow:

$$SFC\% = (packet\_OK/(packet\_OK + packet\_NOK)) * 100 \quad (8)$$

Where ***packet_OK*** is the correct packet in the correct SFC, and ***packet_NOK*** is the packet in the wrong SFC. This percentage is sent as an array [SFC1%, SFC2%] to the ***intelligence*** module. The ***core*** module sends to the agent rewards evaluated for each step.

Figure 21 – Proposed Model as DRL Problem



Source: the Author

The percentage calculated by the ***Core*** module is the state space (observations) in terms of the Deep Reinforcement Learning environment. It consists of an array with two elements, percentage of packet processing successfully in SFC1 and SFC2 [SFC1%, SFC2%]. The information for SFC3 was discarded just for simplicity since most of the application types are relevant to SFC1 and SFC2.

4.3.2 Action Space

The actions consist of changing the traffic between the SFCs. The objective is that after the training period, the intelligent agent forwards each packet to its proper destination application. In other words, it goes through the correct SFC. For example, packets destined for the HTTP service must go through SFC 1 (WEB SFC). Table 8 shows the actions space.

Table 8 – Actions Table

Action Value	Action Meaning
0	Move packet to SFC1
1	Move packet to SFC2
2	Move packet to SFC3

Source: the Author

4.3.3 Reward System

The reward system is based on packet processing performance in SFC1 (WEB traffic) and SFC2 (network management traffic). Table 9 shows the values.

Table 9 – Reward System

Parameter	Reward Value
If (SFC1% or SFC2%) = 100%	+2
If next state value after action is higher than actual state	+1
If next state value after action is lower than actual state	-1
If packets that supposed to be sent to SFC1 or SFC2 were sent to SFC3	-5

Source: the Author

4.4 FINAL CONSIDERATIONS ABOUT THE CHAPTER

This chapter presented and described the proposed model for an architecture based on SFC according to RFC 7665 (HALPERN; PIGNATARO, 2015) and Y.2242 recommendation (ITU-T, 2018), with the goal of the application of DRL algorithms to establish a mechanism of traffic routing that try to provide a better use of network resources. The application of DRL in the SFC environment is essential given the volume of data generated by new applications and the complexity of new network architectures. It is expected that the current limitations will be overcome, providing more intelligent and secure networks.

Several researchers have developed and evaluated test platforms based on the SFC architecture due to their importance in the context of new communication networks and cloud computing. Such efforts are in the papers of Zhang et al. (2018), Castanho et al. (2018), and Peuster et al. (2019).

The list of software and tools used for this research, their versions, functions, and the test platform module to which they are applied are presented in Table 10 below. It is noteworthy that all of them are or have their free versions.

Table 10 – Software and Tools

Software	**Version**	**Function**	**Module**
OpenStack	Queens	VMs Resource Managemnt	All
NPING	0.7.80	Traffic Generator	1-Access-Traffic Generator
IPTABLES	v1.8.4	Packet Flow Control	2-Classifier
TCPDUMP	4.9.3	Packet Capture	2-Classifier
Python	3.8.5	Programming Language	3-Core and 4-Intelligence
OpenAI Gym	0.17.3	DRL Agent	4-Intelligence
PyTorch	1.9	Machine Learning Library	4-Intelligence
Matplotlib	3.4.2	Data Visualization Library	4-Intelligence
Pandas	1.3.0	Data Analysis Library	4-Intelligence

Source: the Author.

5 EXPERIMENTS AND RESULTS

This chapter describes the experiments performed. It is divided into two main sections, the 5.1 that shows the results for the algorithms and the 5.2 that discusses them. Specific details of each algorithm used are explained in their respective subsection.

To evaluate the proposed model, this work deploys different agents using the Q-learning (WATKINS and DAYAN, 1992) , the DQN (MNIH et al., 2015), the Double DQN (VAN HASSELT; GUEZ; SILVER, 2016) and the Dueling DQN (WANG, Z. et al., 2016). The goal is to compare their performance related to routing traffic through the SFCs. Although Q-learning is not considered a DRL algorithm, this research finds it essential to compare some well-known DRL methods with another that does not apply DNNs to show the benefits of using them.

As explained in Chapter 4, the state space is composed of the packet processed in each SFC (Web and Management). And rewards are based on the next state, and it will depend on the next percentage value for the packet processing in each SFC. The rewards were normalized because there was a high variance depending on the number of packets processed. The following formula was used:

$$rewards = rewards\_received\_per\_episode / packets\_processed\_per\_episode \quad (9)$$

The hardware set up used was a server with two Intel X5650 processors (six cores each, 2.66GHz), 32 GB RAM, and an NVIDIA card GeForce GTX 1050Ti (Graphical Processor Unit – GPU), running Linux (Ubuntu 20.04 LTS).

Each algorithm executed 1000 episodes with three packet sampling (250, 500, and 1000 packets). It was repeated the experiments five times. Due to the fact they showed similar results, only one graph for each algorithm using the three different packet samples is presented. Table 12 shows the hyperparameters for the DQN and its variants (Double and Dueling).

Table 11 – Hyperparameters Values

Parameter	Value
Hidden Layers	1
Neurons (Hidden Layer)	50
Discount Factor (γ)	0.8
Learning Rate (LR)	0.001
Exploration Rate (ϵ)	1
Epsilon Decay	0.99
Minimum Epsilon	0.1
Batch or Replay Size	20

Source: the Author

5.1 RESULTS

This section presents the results, and the subsections are divided according to the algorithm used. The subsequent sessions discuss them.

5.1.1 Q-learning

This experiment used Q-learning to evaluate the proposed model. The goal is to investigate if the algorithm can improve the packet processing in the SFCs by learning over episodes. Three experiments were performed using different values for γ (Discount Factor = 1.0, 0.5, and 0), Jay et al. (2019) used the same approach to compare their results. Besides the different γ values, a different number of packets for the training was used. The objective is to understand if the sampling changes the agent behavior. The tables below present the results in terms of statistical data, where SD means Standard Deviation.

Table 12 – Q-learning for 250 Packets

Experiment	Mean	SD	Mean - Last 100 Episodes	SD - Last 100 Episodes
$\gamma = 1.0$	0.090	0.241	0.208	0.001
$\gamma = 0.5$	0.093	0.233	0.214	0.008
$\gamma = 0.0$	0.091	0.237	0.214	0.014

Source: the Author

Table 13 – Q-learning for 500 Packets

Experiment	Mean	SD	Mean - Last 100 Episodes	SD - Last 100 Episodes
$\gamma = 1.0$	0.141	0.259	0.272	0.001
$\gamma = 0.5$	0.155	0.237	0.276	0.006
$\gamma = 0.0$	0.153	0.240	0.275	0.062

Source: the Author

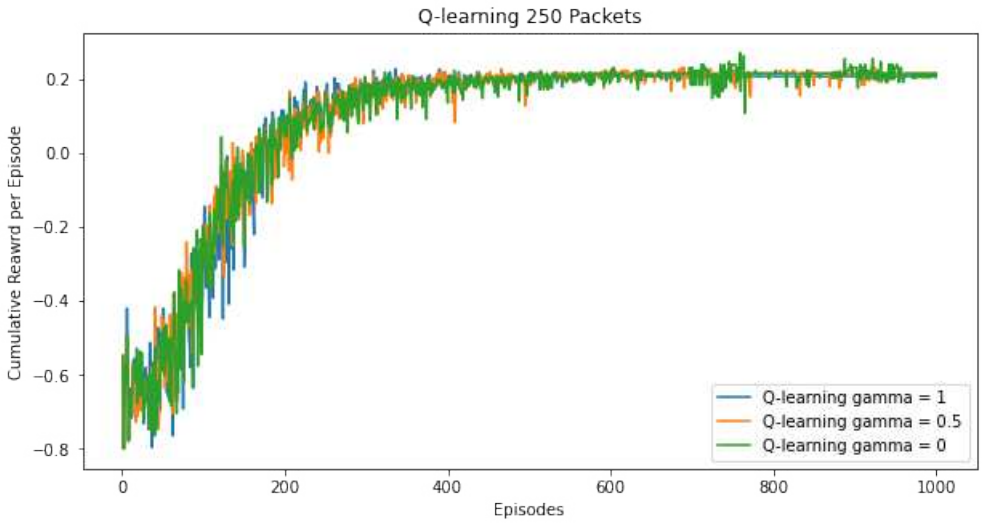
Table 14 – Q-learning for 1000 Packets

Experiment	Mean	SD	Mean - Last 100 Episodes	SD - Last 100 Episodes
$\gamma = 1.0$	0.075	0.253	0.232	0.001
$\gamma = 0.5$	0.115	0.223	0.232	0.006
$\gamma = 0.0$	0.109	0.237	0.213	0.197

Source: the Author

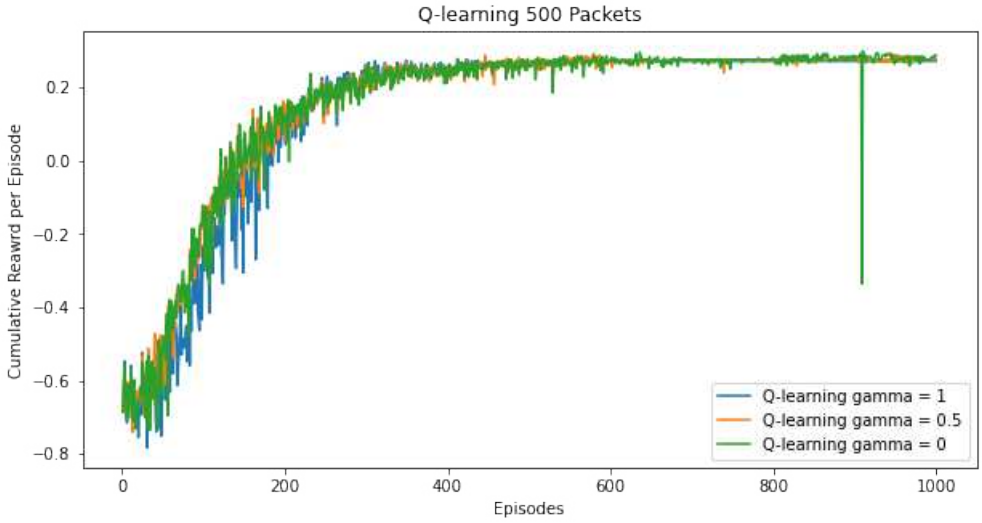
The values on the tables show that the $\gamma = 0.5$ resulted in the best performance in terms of mean rewards for the last 100 episodes, some after the agent converged and started to show stability. The following figures show the results. The points in the graphic indicate the sum of normalized rewards received for each episode.

Figure 22 – Q-learning for 250 Packets



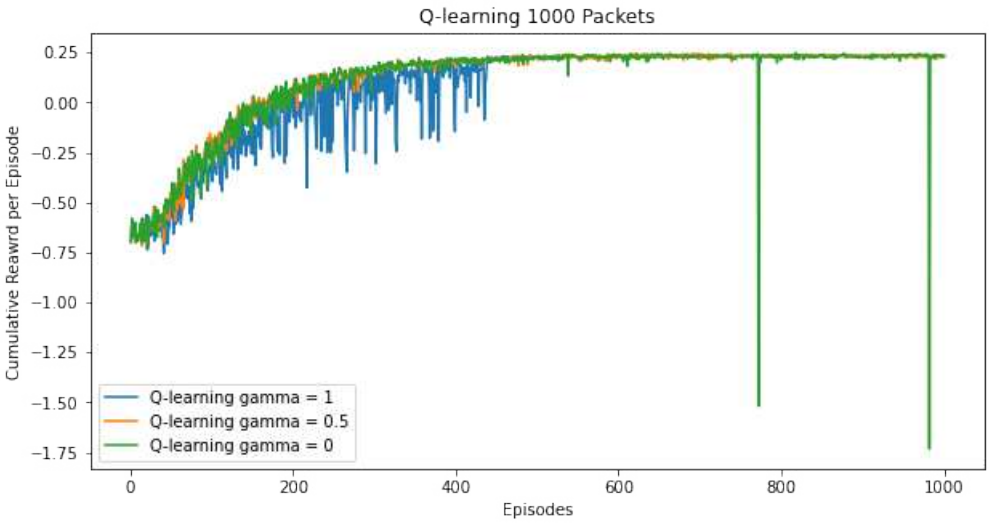
Source: the Author

Figure 23 – Q-learning for 500 Packets



Source: the Author

Figure 24 – Q-learning for 1000 Packets



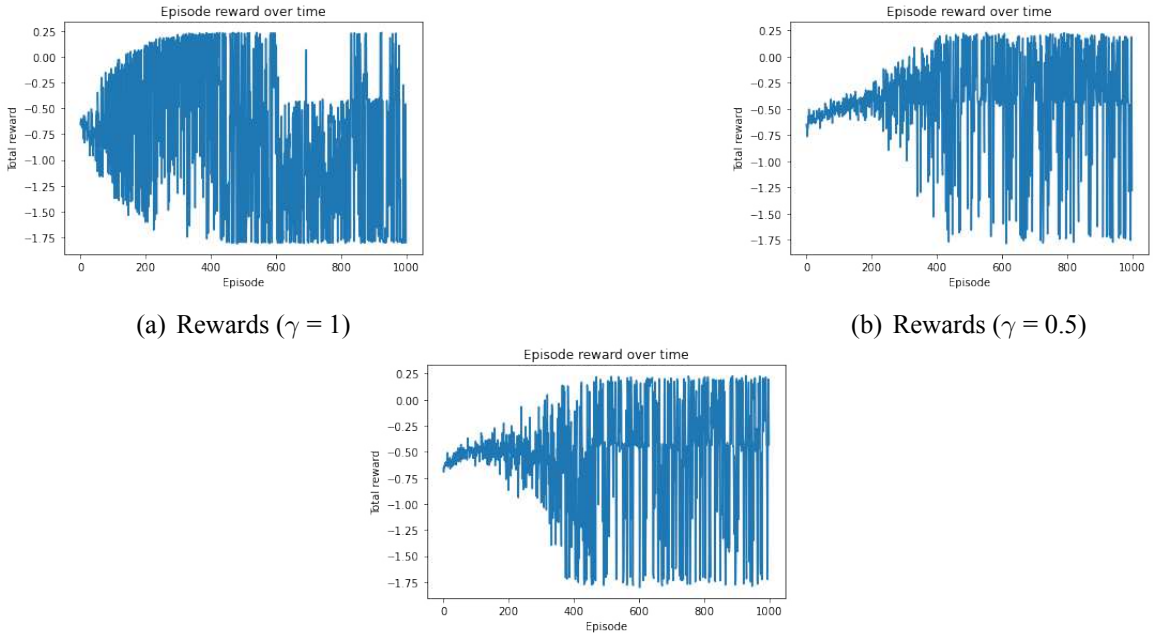
Source: the Author

The results show that the agent can learn over the episodes, selecting the correct SFC path according to the traffic type, even though the $\gamma = 0$ had instability for 500 and 1000 packets at the end. The shape of the figures is due to the randomness of the traffic in the training environment. This fact is valid for all the other algorithms.

5.1.2 DQN

Here the results for the experiments using DQN are presented. They used the same procedure as for Q-learning (same values for packet sampling and γ). However, the agent resulted unstable, as Figure 25 shows.

Figure 25 – Reward over Episodes for 1000 Packets



(a) Rewards ($\gamma = 1$)

(b) Rewards ($\gamma = 0.5$)

(c) Rewards ($\gamma = 0$)

Source: the Author

Due to this issue, other experiments were performed using $\gamma = 0.8$. The results improved, as it is possible to see in the graphics below. The evaluation also used two different learning rates (LR = 0.01 and LR = 0.001) to check the better performance and stability. The tables below show the statistical results.

Table 15 – DQN for 250 Packets ($\gamma = 0.8$)

Experiment	Mean	SD	Mean-Last 100 Episodes	SD-Last 100 Episodes
LR = 0.01	-0.185	0381	-0.054	0.248
LR = 0.001	-0.138	0.562	0.175	0.027

Source: the Author

Table 16 – DQN for 500 Packets ($\gamma = 0.8$)

Experiment	Mean	SD	Mean-Last 100 Episodes	SD-Last 100 Episodes
LR = 0.01	-0.272	0.393	-0.044	0.271
LR = 0.001	-0.009	0.510	0.207	0.123

Source: the Author

Table 17 – DQN for 1000 Packets (γ = 0.8)

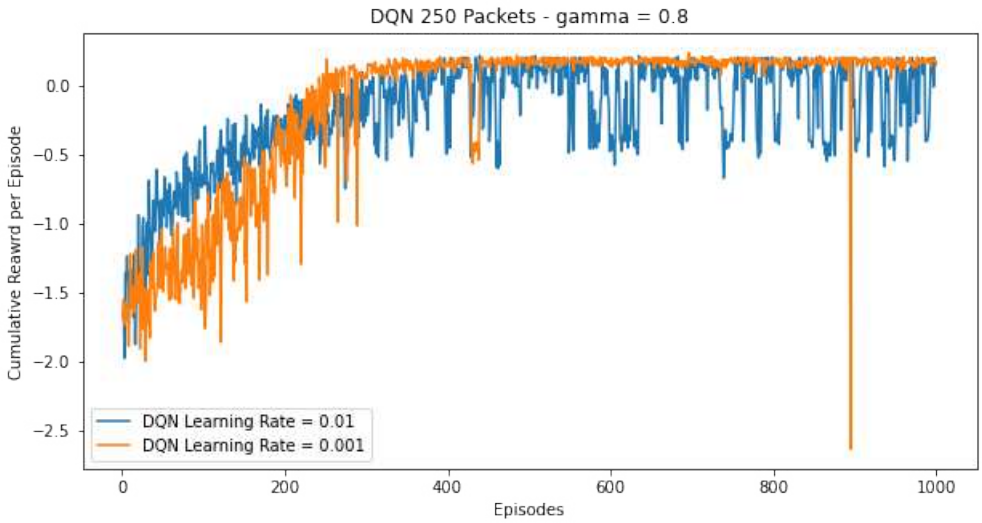
Experiment	Mean	SD	Mean-Last 100 Episodes	SD-Last 100 Episodes
LR = 0.01	-0.376	0385	0.173	0.275
LR = 0.001	-0.077	0.432	0.109	0.154

Source: the Author

As the tables above show, the experiments that used LR = 0.001 presented a better performance for the rewards average, except for the 1000-packet sampling. However, the sample mentioned provided better stability with the same LR as the standard deviation value shows. And sometimes, network environments must have a more stable system than those with a better average metric.

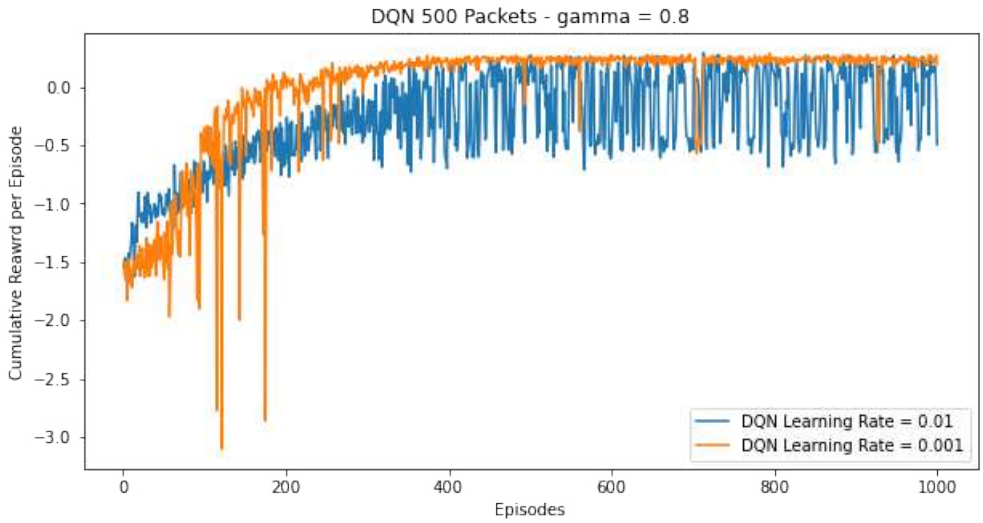
The following graphics show the rewards received from the environment for each episode during the training.

Figure 26 – DQN for 250 Packets (γ = 0.8)



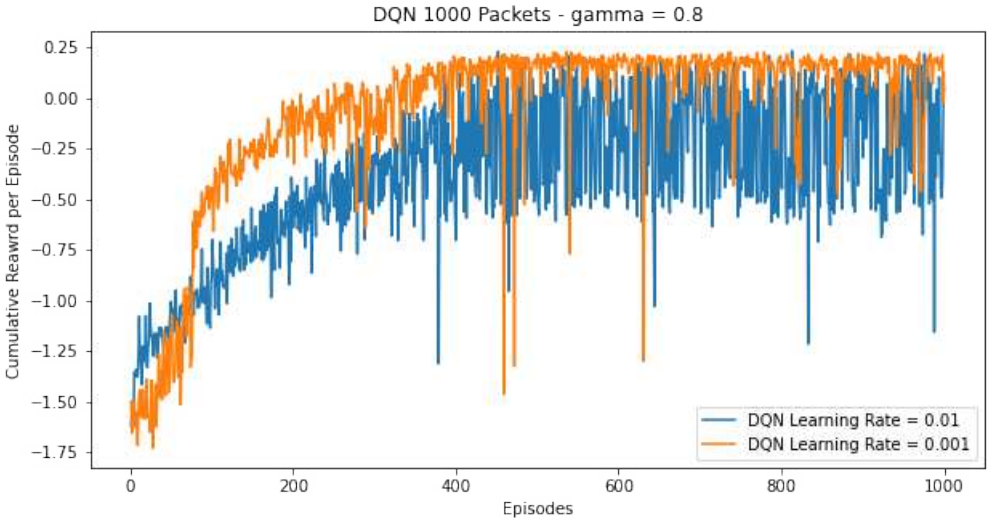
Source: the Author

Figure 27 – DQN for 500 Packets ($\gamma = 0.8$)



Source: the Author

Figure 28 – DQN for 1000 Packets ($\gamma = 0.8$)



Source: the Author

It is possible to conclude that for the environment used in this work, the best value for gamma using DQN is 0.8, and the learning rate of 0.001 provided more stability at the end episodes. Then, the following experiments (Double and Dueling) will use the same parameters.

5.1.3 Double DQN

In this experiment, the agent used the Double DQN algorithm. But, now, the tests only used the Learning Rate = 0.001 since this was the best value for DQN. The tables below show the statistical results for each packet sampling. The other hyperparameters values are the same as illustrated in Table 18.

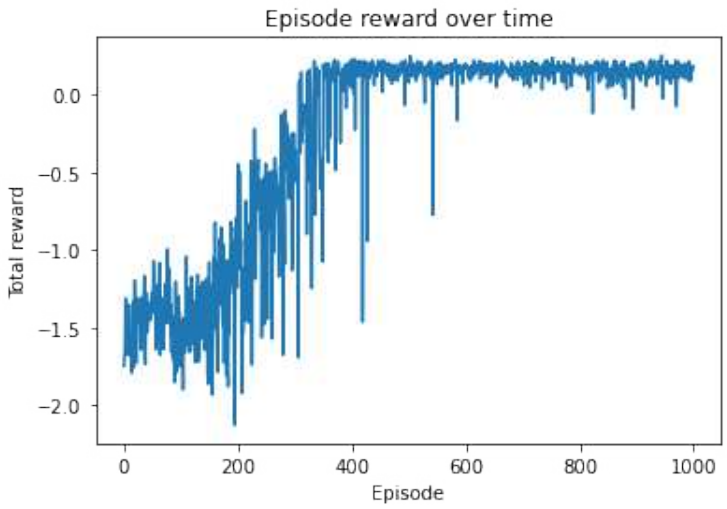
Table 18 – Dueling DQN Experiments

Experiment	Mean	SD	Mean - Last 100 Episodes	SD - Last 100 Episodes
250 Packets	-0.282	0.665	0.151	0.051
500 Packets	-0.123	0.592	0.191	0.055
1000 Packets	-0.026	0.627	0.092	0.085

Source: the Author

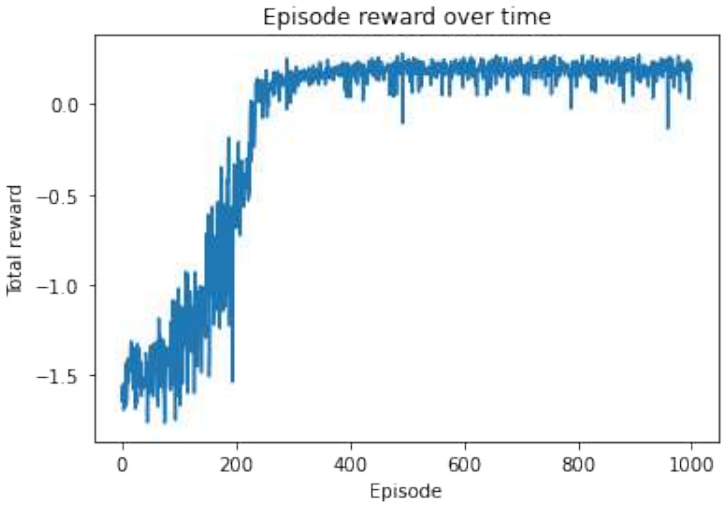
Verifying the table is possible to check that the 500-packet sample had the better performance in terms of mean rewards over the episodes. The figures below illustrate it.

Figure 29 – Double DQN for 250 Packets (Learning Rate = 0.001)



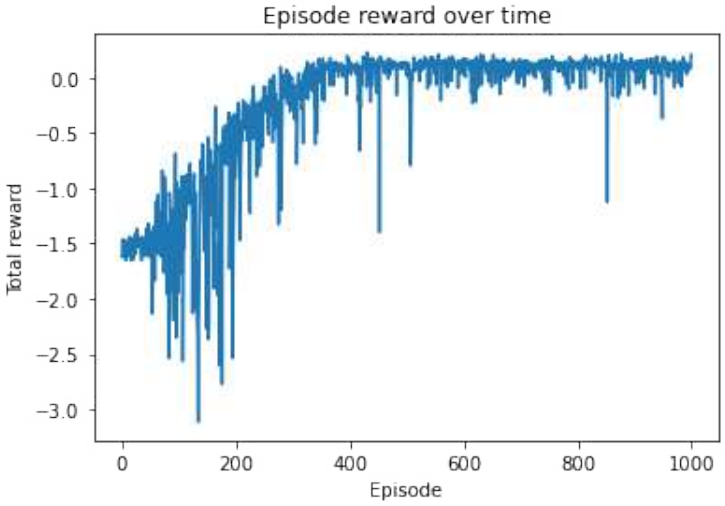
Source: the Author

Figure 30 – Double DQN for 500 Packets (Learning Rate = 0.001)



Source: the Author

Figure 31 – Double DQN for 1000 Packets (Learning Rate = 0.001)



Source: the Author

The results show that the agent can learn over the episodes, selecting the correct SFC path according to the traffic type and reaching stability.

5.1.4 Dueling DQN

For Dueling DQN was followed the same procedure as in Double DQN experiments. The tables below show the statistical data.

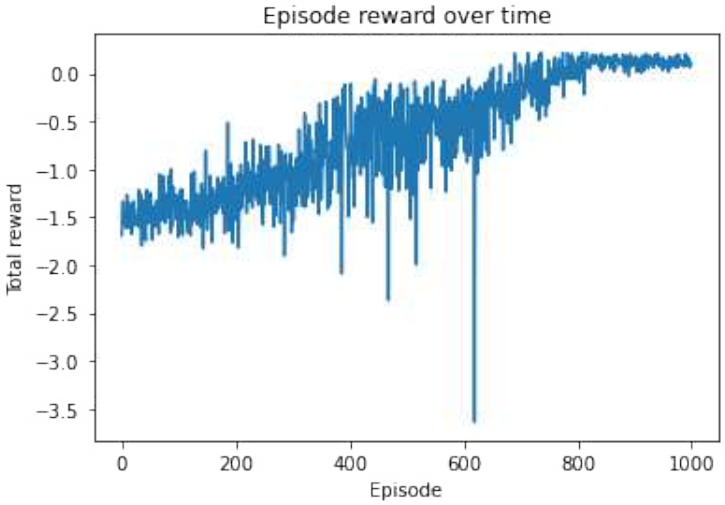
Table 19 – Dueling DQN Experiments - Learning Rate = 0.001

Experiment	Mean	SD	Mean - Last 100 Episodes	SD - Last 100 Episodes
250 Packets	-0.620	0.612	0.118	0.047
500 Packets	-0.380	0.671	0.136	0.033
1000 Packets	-0.222	0.542	0.070	0.032

Source: the Author

And the graphics below show the results for the rewards over episodes.

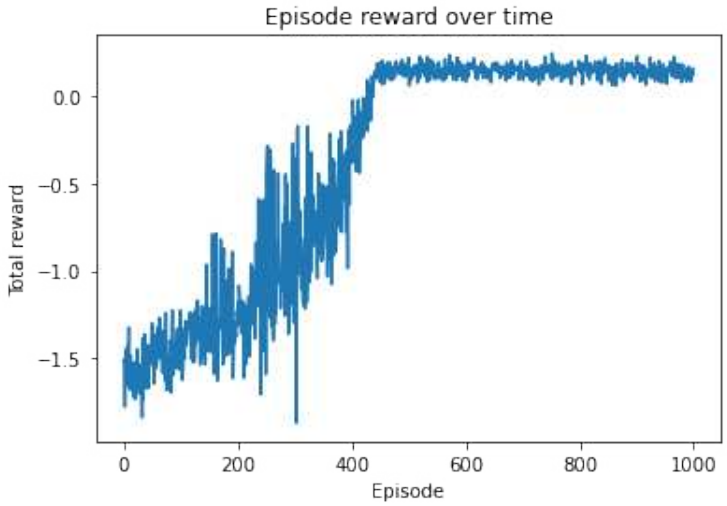
Figure 32 – Dueling DQN for 250 Packets (Learning Rate = 0.001)



Source: the Author

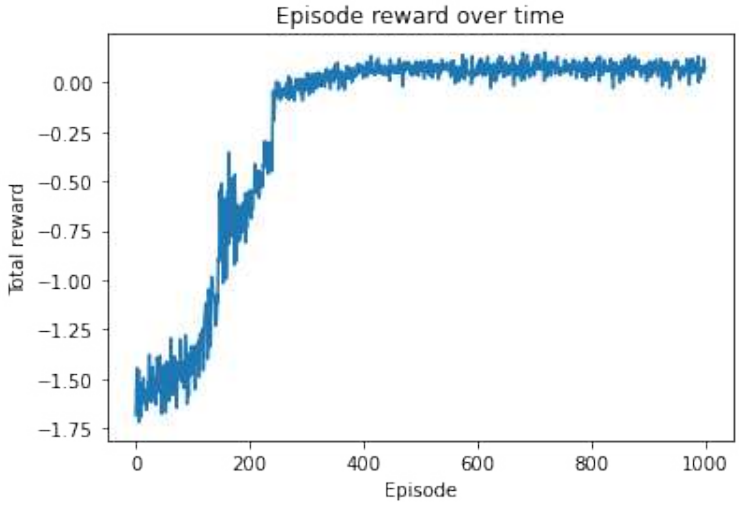
The results show that the agent can learn over the episodes, selecting the correct SFC path according to the traffic type, and it reaches stability. A comparison between the performance of the different algorithms will be discussed in the next section.

Figure 33 – Dueling DQN for 500 Packets (Learning Rate = 0.001)



Source: the Author

Figure 34 – Dueling for 1000 Packets - Learning Rate = 0.001



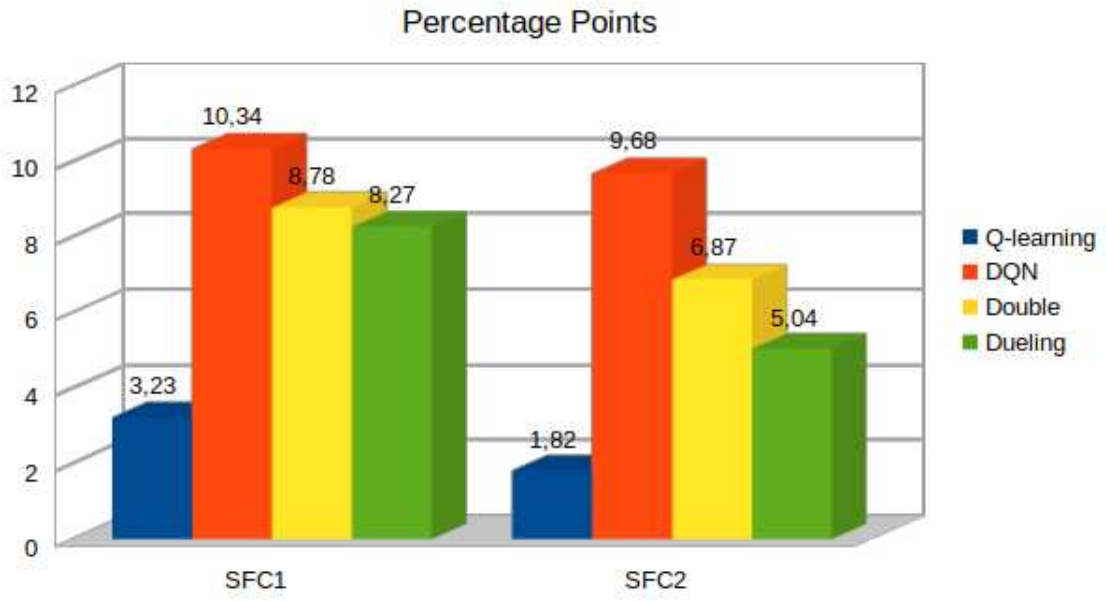
Source: the Author

5.2 RESULTS DISCUSSION

In this section, the results are discussed. The experiments tried to evaluate the applicability of DRL algorithms to forward the packets through the correct SFC. Then what should be analyzed: is it possible to use those algorithms to improve the dynamic SFC selection related to the network's initial state?

A fact that should be evaluated is the importance of using DNNs. The advantage is shown in Figure 35, where algorithms using DNNs overcome the results of Q-learning (without DNNs). The graphic below presents a summary of the algorithms used in this work. It shows the difference in percentage points (p.p.) compared with the initial state (packet processing values) after the training period.

Figure 35 – Performance (Comparing with Initial State)



Source: the Author

Packet processing was used as the metric for comparison because, in terms of network performance, it is considered one of the most important, along with delay, jitter, and throughput. The following table illustrates the packet processing per SFC after the training period.

Table 20 – Dueling DQN Experiments - Learning Rate = 0.001

Algorithm	**SFC1** %	**SFC1** %
Q-learning	61.94	70.00
DQN	63.15	34.39
Double DQN	61.59	38.46
Dueling DQN	63.79	34.38

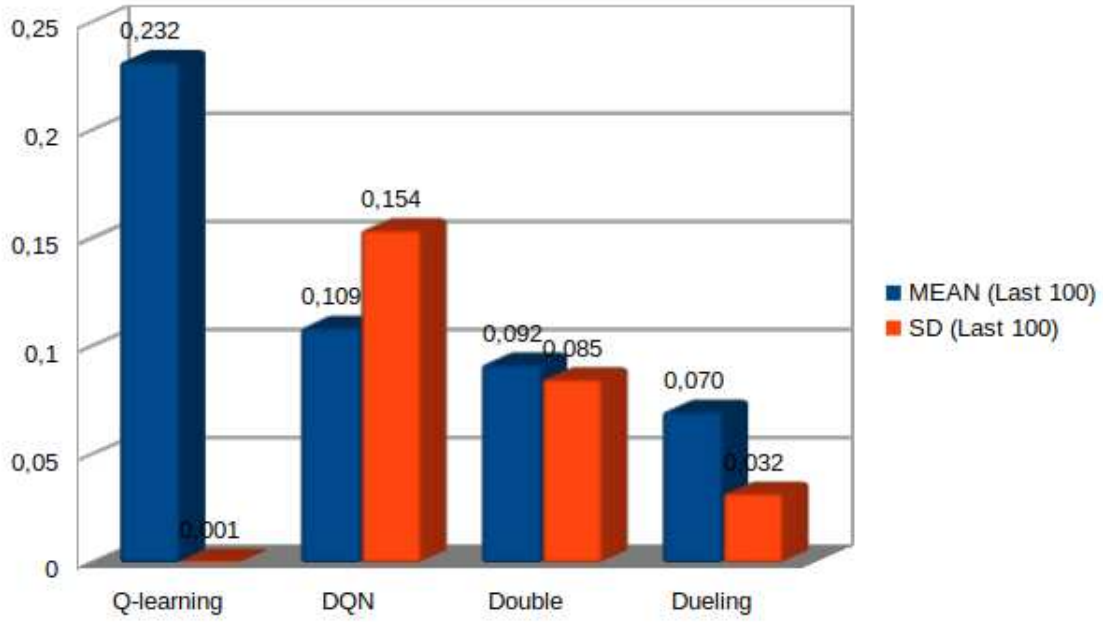
Source: the Author

Although the metrics in the experiments are far from ideal for a production environment, it is possible to improve the algorithms to get better performance. Moreover, it is possible to

test other algorithms such as Advantage Actor-Critic (A2C) and Proximal Policy Optimization (PPO) using the proposed architecture. Another alternative for improvements is to try different hyperparameters for the DNNs.

Another performance measurement is the rewards mean and standard deviation. That is important not only for understanding the stability but also the advantage in using DRL methods and comparing their performance. The following graphic shows this data.

Figure 36 – Performance (Mean and Standard Deviation)

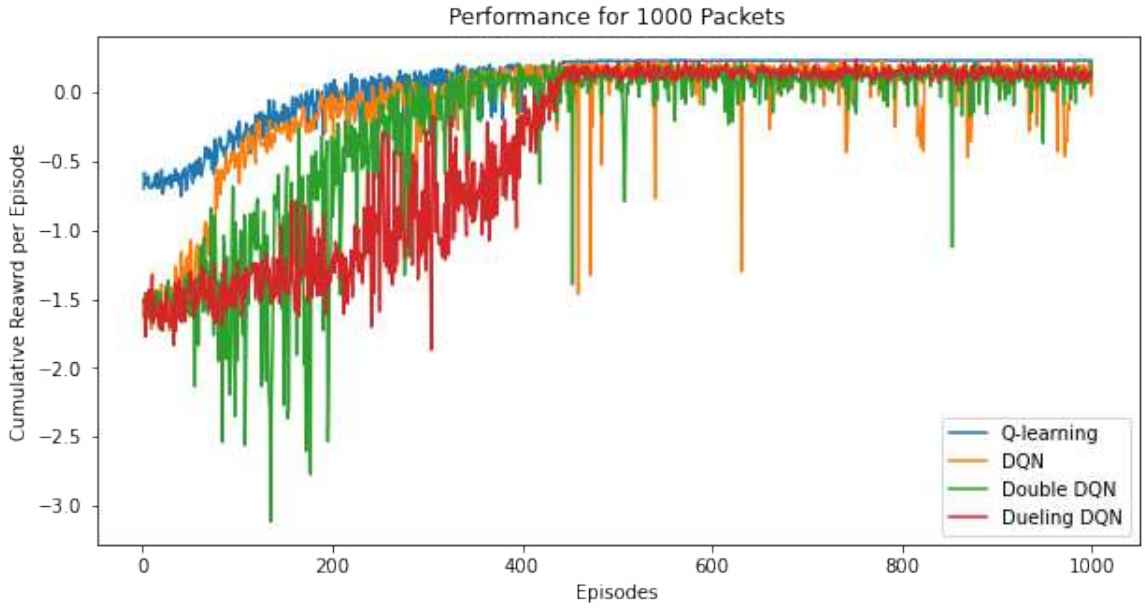


Source: the Author

It is possible to conclude that Q-learning offers the best performance in terms of mean and stability (lower standard deviation). However, in terms of packet processing, DQN had the best performance for both SFCs. It could improve in 10.34 p.p. for SFC1 (Web) and 9.68 p.p. for SFC2 (Management).

Finally, in Figure 37 where the four algorithms used in this work are compared. The shape of the figure is due to the randomness of the traffic in the training environment. It shows the rewards over the episodes. We can conclude that the agent learns over the episodes, selecting the correct SFC path according to the traffic type.

Figure 37 – Q-learning x DQN x Double x Dueling



Source: the Author

The proposed architecture tried to use Double and Dueling DQN variants. However, these algorithms did not have better performance than using DQN. According to Luong et al. (2019) a possible reason is using them only benefits for MDPs with large action spaces. For small state spaces, as in the case of the environment used in this work, the performance of DQN is even better than Double and Dueling. Moreover, during the experiments, the Dueling takes more time to finish the training period; thus, more computing resources are needed to apply such a method.

6 CONCLUSION

This work proposed architecture to investigate how Deep Reinforcement Learning algorithms can be used in the context of the Service Function Chaining to provide a mechanism for identifying and forward network traffic based on profiles. The proposed architecture divided the traffic into three profiles (WEB, Management, and Unknown). To evaluate the model, it deployed an agent using Python, and it got the environment's information and took action (selects an SFC). It receives a reward according to the result of its actions. The experiments used Q-learning, Deep q-networks, and two variants, the Double DQN and Dueling DQN. A packet sampling with a different number of packets and IP applications was used as the dataset to evaluate the model.

The experimental results elucidated that the agent in charge of making the routing decisions can learn how to do that over time (episodes), applying the policies to get optimized rewards. Such mechanisms help the networks be more resilient, intelligent, and secure. Q-learning provided a better result in learning stability and rewards average. The DQN resulted in better performance related to packet processing, the metric that matters in production networks. The other algorithms tested also presented positive results in the experiments improving the percentage of packets processed compared with the initial state and optimizing the rewards over the episodes.

This work faced some challenges, for example, modeling the proposed architecture as a Deep Reinforcement learning environment. According to Graesser e Keng (2019), it is a common issue that the DRL researchers must solve. Defining states and rewards is challenging if the environment does not provide enough information to the agent. In the case of this research, the reward mechanism was tested empirically, and it took much time to find the best values for rewarding.

Another challenge and point for improvement are getting better performance in packet processing. The results varied from 60% to 65%, which is unacceptable for production networks. However, it is possible to go further and find out what the reason is. Perhaps, the first thing to evaluate is the hyperparameters for each algorithm since they are susceptible depending on those parameters. Another thing to test would be a different reward mechanism.

The main contribution of this work is the implementation of a model to evaluate the use of machine learning techniques and technology trends in networking. Specifically, the Service Function Chaining has several applications in 5G systems, data centers, and other networks.

Also, another contribution was a paper with the initial results published in the ENIAC (Encontro Nacional de Inteligência Artificial e Computacional) (JÚNIOR; BIANCHI, 2021). That helped to demonstrate the relevance of this research.

From the issues raised from this work, there are some suggestions for future works to address them. For example, the experiments focused on Deep Q-Learning and its variants known as Value-based methods. Then a possible improvement is to test other algorithms based on Policy Gradient, such as REINFORCE or methods that combine value and policy methods. For instance, Advantage Actor-Critic (A2C), Proximal Policy Optimization (PPO), and Soft Actor-Critic (SAC). Other possibilities are using different network metrics as state variables, such as memory occupation, CPU load, and delay. That would help deploy a complete agent to improve packet loss performance.

From the network perspective, a possible change is using a dedicated SDN Controller, the OpenDayLight (OPENDAYLIGHT, 2021). It uses specific APIs to interact with the Python libraries, making the interaction between the agent and the environment more accessible. Besides, applying Docker (DOCKER, 2021) containers and Kubernetes (KUBERNETES, 2021) as cloud orchestrator would align with the cloud's newest technologies, even by 5G networks.

REFERENCES

AKBARI, Iman et al. ATMoS: Autonomous Threat Mitigation in SDN using Reinforcement Learning. In: IEEE. NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium. [S.l.: s.n.], 2020. P. 1–9.

BHAMARE, Deval et al. A survey on service function chaining. **Journal of Network and Computer Applications**, Elsevier, v. 75, p. 138–155, 2016.

BLANCO, Bego et al. Technology pillars in the architecture of future 5G mobile networks: NFV, MEC and SDN. **Computer Standards Interfaces**, v. 54, jan. 2017. DOI: 10.1016/j.csi.2016.12.007.

BROCKMAN, Greg et al. **OpenAI Gym**. [S.l.: s.n.], 2016. eprint: arXiv:1606.01540.

CASTANHO, M. S. et al. PhantomSFC: A Fully Virtualized and Agnostic Service Function Chaining Architecture. In: 2018 IEEE Symposium on Computers and Communications (ISCC). [S.l.: s.n.], 2018. P. 354–359. DOI: 10.1109/ISCC.2018.8538593.

DOCKER. **Docker**. [S.l.: s.n.]. Retrieved from: https://www.docker.com/. Accessed on: 10 out. 2021.

DONNE, J.; LUSH, H. **No Man Is an Island**. [S.l.]: Souvenir Press Limited, 1988. (Inspirational S). ISBN 9780285628748. Retrieved from: https://books.google.com.br/books?id=RFuhOgAACAAJ.

ENNS, Rob et al. **Network Configuration Protocol (NETCONF)**. [S.l.]: RFC Editor, jun. 2011. 113 p. RFC 6241. (Request for Comments, 6241). DOI: 10.17487/RFC6241. Retrieved from: https://rfc-editor.org/rfc/rfc6241.txt. Accessed on: 24 jul. 2021.

ETSI. **Multi-access Edge Computing (MEC); Framework and Reference Architecture**. [S.l.]: ETSI GS, 2019.

ETSI, GSNFV. Network functions virtualisation (nfv): Architectural framework. **ETsI Gs NFV**, v. 2, n. 2, p. v1, 2013.

FENG, Ji; YU, Yang; ZHOU, Zhi-Hua. Multi-layered gradient boosting decision trees. In: ADVANCES in neural information processing systems. [S.l.: s.n.], 2018. P. 3551–3561.

FU, Xiaoyuan et al. Dynamic service function chain embedding for NFV-enabled IoT: A deep reinforcement learning approach. **IEEE Transactions on Wireless Communications**, IEEE, v. 19, n. 1, p. 507–519, 2019.

GERHARDS, R. **The Syslog Protocol**. [S.l.]: IETF, mar. 2009. RFC 5424 (Proposed Standard). (Request for Comments, 5424). Retrieved from: http://www.ietf.org/rfc/rfc5424.txt. Accessed on: 24 jul. 2021.

GRAESSER, Laura; KENG, Wah Loon. **Foundations of Deep Reinforcement Learning: Theory and Practice in Python**. [S.l.]: Addison-Wesley Professional, 2019.

GUO, Boren et al. Deep-Q-network-based multimedia multi-service QoS optimization for mobile edge computing systems. **IEEE Access**, IEEE, v. 7, p. 160961–160972, 2019.

HALEPLIDIS, Evangelos et al. **Software-Defined Networking (SDN): Layers and Architecture Terminology**. [S.l.]: RFC Editor, jan. 2015. 35 p. RFC 7426. (Request for Comments, 7426). DOI: 10.17487/RFC7426. Retrieved from: https://rfc-editor.org/rfc/rfc7426.txt. Accessed on: 24 jul. 2021.

HALPERN, Joel M.; PIGNATARO, Carlos. **Service Function Chaining (SFC) Architecture**. [S.l.]: RFC Editor, out. 2015. 32 p. RFC 7665. (Request for Comments, 7665). DOI: 10.17487/RFC7665. Retrieved from: https://rfc-editor.org/rfc/rfc7665.txt. Accessed on: 24 jul. 2021.

HALPERN, Joel M. et al. **Forwarding and Control Element Separation (ForCES) Protocol Specification**. [S.l.]: RFC Editor, mar. 2010. 124 p. RFC 5810. (Request for Comments, 5810). DOI: 10.17487/RFC5810. Retrieved from: https://rfc-editor.org/rfc/rfc5810.txt. Accessed on: 24 jul. 2021.

HANTOUTI, Hajar; BENAMAR, Nabil; TALEB, Tarik. Service Function Chaining in 5G and Beyond Networks: Challenges and Open Research Issues. **IEEE Network**, v. 34, n. 4, p. 320–327, 2020. DOI: 10.1109/MNET.001.1900554.

HARRINGTON, David; WIJNEN, Bert; PRESUHN, Randy. **An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks**. [S.l.]: RFC Editor, dez. 2002. 64 p. RFC 3411. (Request for Comments, 3411). DOI: 10.17487/RFC3411. Retrieved from: https://rfc-editor.org/rfc/rfc3411.txt. Accessed on: 24 jul. 2021.

HASSELT, Hado. Double Q-learning. **Advances in neural information processing systems**, v. 23, p. 2613–2621, 2010.

HE, Bo et al. Towards Intelligent Provisioning of Virtualized Network Functions in Cloud of Things: A Deep Reinforcement Learning based Approach. **IEEE Transactions on Cloud Computing**, IEEE, 2020.

HUNTER, J. D. Matplotlib: A 2D graphics environment. **Computing in Science & Engineering**, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

IPTABLES. **IPtables**. [S.l.: s.n.]. Retrieved from: https://man7.org/linux/man-pages/man8/iptables.8.html. Accessed on: 17 set. 2021.

ITU-T. **Framework of Software-Defined Networking**. [S.l.], jun. 2014. Recommendation ITU-T Y.3300. Retrieved from: http://www.itu.int/rec/T-REC-Y.3300-201406-I/en. Accessed on: 30 ago. 2021.

ITU-T. **Service Function Chaining in Mobile Networks**. [S.l.], jun. 2018. Recommendation ITU-T Y.2248. Retrieved from: http://www.itu.int/rec/T-REC-Y.3300-201406-I/en. Accessed on: 12 nov. 2021.

ITU-T. **TMN Management Functions**. [S.l.], fev. 2000. Recommendation ITU-T M.3400. Retrieved from: https://www.itu.int/rec/T-REC-M.3400-200002-I/. Accessed on: 30 ago. 2021.

JAY, Nathan et al. A deep reinforcement learning perspective on internet congestion control. In: PMLR. INTERNATIONAL Conference on Machine Learning. [S.l.: s.n.], 2019. P. 3050–3059.

JÚNIOR, Silvio Araújo; BIANCHI, Reinaldo. A Model for Traffic Forwarding through Service Function Chaining using Deep Reinforcement Learning Techniques. In: ANAIS do XVIII Encontro Nacional de Inteligência Artificial e Computacional. Evento Online: SBC, 2021. P. 619–630. Retrieved from: https://sol.sbc.org.br/index.php/eniac/article/view/18289. Accessed on: 15 dez. 2021.

KREUTZ, Diego et al. Software-defined networking: A comprehensive survey. **Proceedings of the IEEE**, Ieee, v. 103, n. 1, p. 14–76, 2014.

KUBERNETES. **Kubernetes**. [S.l.: s.n.]. Retrieved from: https://kubernetes.io. Accessed on: 10 out. 2021.

LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. Deep learning. **Nature**, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015.

LI, Guanglei et al. Adaptive service function chaining mappings in 5G using deep Q-learning. **Computer Communications**, Elsevier, v. 152, p. 305–315, 2020.

LI, Rongpeng et al. Deep reinforcement learning for resource management in network slicing. **IEEE Access**, IEEE, v. 6, p. 74429–74441, 2018.

LIANG, Eric et al. Neural packet classification. In: PROCEEDINGS of the ACM Special Interest Group on Data Communication. [S.l.: s.n.], 2019. P. 256–269.

LUONG, Nguyen Cong et al. Applications of deep reinforcement learning in communications and networking: A survey. **IEEE Communications Surveys & Tutorials**, IEEE, v. 21, n. 4, p. 3133–3174, 2019.

MCCULLOCH, Warren S; PITTS, Walter. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, Springer, v. 5, n. 4, p. 115–133, 1943.

MCKINNEY, Wes. Data Structures for Statistical Computing in Python. In: WALT, Stéfan van der; MILLMAN, Jarrod (Ed.). **Proceedings of the 9th Python in Science Conference**. [S.l.: s.n.], 2010. P. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

MEDHAT, Ahmed M et al. Service function chaining in next generation networks: State of the art and research challenges. **IEEE Communications Magazine**, IEEE, v. 55, n. 2, p. 216–223, 2016.

MITCHELL, Tom. **Machine learning**. [S.l.]: McGraw hill Burr Ridge, 1997.

MNIH, Volodymyr et al. Human-level control through deep reinforcement learning. **Nature**, Nature Publishing Group, v. 518, n. 7540, p. 529–533, 2015.

NGUYEN, Thanh Thi; REDDI, Vijay Janapa. Deep reinforcement learning for cyber security. **arXiv preprint arXiv:1906.05799**, 2019.

NING, Zili; WANG, Ning; TAFAZOLLI, Rahim. Deep Reinforcement Learning for NFV-based Service Function Chaining in Multi-Service Networks. In: IEEE. 2020 IEEE 21st International Conference on High Performance Switching and Routing (HPSR). [S.l.: s.n.], 2020. P. 1–6.

NPING. **Nping: Measuring the Network**. [S.l.: s.n.]. Retrieved from: https://man7.org/linux/man-pages/man8/iptables.8.html. Accessed on: 15 set. 2021.

NYANG, DaeHun; SHIN, DongOh. Recyclable counter with confinement for real-time per-flow measurement. **IEEE/ACM Transactions on Networking**, IEEE, v. 24, n. 5, p. 3191–3203, 2016.

ONF. **OpenFlow Switch Specification: Version 1.5.1 (Protocol version 0x06)**. [S.l.], 2015.

ONF. **SDN Architecture**. [S.l.], out. 2016. P. 1–59.

OPENDAYLIGHT. **OpenDaylight Platform**. [S.l.: s.n.]. Retrieved from: https://docs.opendaylight.org/projects/sfc/en/latest/user-guide.html. Accessed on: 10 out. 2021.

OPENSTACK. **OpenStack**. [S.l.: s.n.]. Retrieved from: https://www.openstack.org/. Accessed on: 10 out. 2021.

PASZKE, Adam et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: WALLACH, H. et al. (Ed.). **Advances in Neural Information Processing Systems 32**. [S.l.]: Curran Associates, Inc., 2019. P. 8024–8035. Retrieved from: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf. Accessed on: 15 nov. 2021.

PELLEGRINI, Jerônimo; WAINER, Jacques. Processos de Decisão de Markov: um tutorial. **RITA**, v. 14, n. 2, p. 133–179, 2007. DOI: 10.22456/2175-2745.5694. Retrieved from: https://doi.org/10.22456/2175-2745.5694. Accessed on: 12 jun. 2021.

PETERS, Jan; BAGNELL, J Andrew. Policy Gradient Methods. **Scholarpedia**, Springer, v. 5, n. 11, p. 3698, 2010.

PEUSTER, Manuel et al. Automated testing of NFV orchestrators against carrier-grade multi-PoP scenarios using emulation-based smoke testing. **EURASIP Journal on Wireless Communications and Networking**, Springer, v. 2019, n. 1, p. 172, 2019.

PIENROJ, Panin; SCHÖNBORN, Sandro; BIRKE, Robert. Exploring Deep Reinforcement Learning for Autonomous Powerline Tracking. In: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). [S.l.: s.n.], 2019. P. 496–501. DOI: 10.1109/INFCOMW.2019.8845212.

QUINN, Paul; ELZUR, Uri; PIGNATARO, Carlos. **Network Service Header (NSH)**. [S.l.]: RFC Editor, jan. 2018. 40 p. RFC 8300. (Request for Comments, 8300). DOI: 10.17487/RFC8300. Retrieved from: https://rfc-editor.org/rfc/rfc8300.txt. Accessed on: 24 jul. 2021.

RAVICHANDIRAN, Sudharsan. **Hands-on reinforcement learning with Python: master reinforcement and deep reinforcement learning using OpenAI gym and tensorFlow**. [S.l.]: Packt Publishing Ltd, 2018.

RESTUCCIA, Francesco; MELODIA, Tommaso. DeepWiERL: Bringing Deep Reinforcement Learning to the Internet of Self-Adaptive Things. In: PROC. of IEEE Conference on Computer Communications (INFOCOM). [S.l.: s.n.], 2020.

RIEDMILLER, Martin. Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method. In: SPRINGER. EUROPEAN Conference on Machine Learning. [S.l.: s.n.], 2005. P. 317–328.

ROESCH, M. Snort: Lightweight Intrusion Detection for Networks. In: LISA. [S.l.: s.n.], 1999.

SHARAFALDIN, Iman; LASHKARI, Arash Habibi; GHORBANI, Ali A. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: ICISSP. [S.l.: s.n.], 2018. P. 108–116.

SHIN, Sang-Min; KWON, Gu-In. Real-time Monitoring Technique Using SFC Classifier in NFV Environment. **International Journal of Applied Engineering Research**, v. 12, n. 19, p. 8676–8680, 2017.

SINGH, Sanjeev; JHA, Rakesh Kumar. A survey on software defined networking: Architecture for next generation network. **Journal of Network and Systems Management**, Springer, v. 25, n. 2, p. 321–374, 2017.

SMITH, J Cole; TASKIN, Z Caner. A tutorial guide to mixed-integer programming models and solution techniques. **Optimization in Medicine and Biology**, Taylor e Francis Auerbach Publications, p. 521–548, 2008.

SUN, Kyoungjae; KIM, Young. Service Function Chaining Architecture for Distributed 5G Mobile Core Networks. **The Journal of Korean Institute of Communications and Information Sciences**, v. 41, p. 1914–1924, dez. 2016. DOI: 10.7840/kics.2016.41.12.1914.

SUTTON, Richard S; BARTO, Andrew G. **Reinforcement learning: An introduction**. [S.l.]: MIT press, 2018.

TCPDUMP. **Tcpdump Packet Analizer**. [S.l.: s.n.]. Retrieved from: https://www.tcpdump.org/. Accessed on: 17 set. 2021.

TEAM, The pandas development. **pandas-dev/pandas: Pandas**. [S.l.]: Zenodo, fev. 2020. DOI: 10.5281/zenodo.3509134. Retrieved from: https://doi.org/10.5281/zenodo.3509134. Accessed on: 28 nov. 2021.

TRAJKOVSKA, Irena et al. SDN-based service function chaining mechanism and service prototype implementation in NFV scenario. **Computer Standards & Interfaces**, Elsevier, v. 54, p. 247–265, 2017.

VAN HASSELT, Hado; GUEZ, Arthur; SILVER, David. Deep reinforcement learning with double q-learning. In: 1. PROCEEDINGS of the AAAI conference on artificial intelligence. [S.l.: s.n.], 2016. v. 30.

VAN ROSSUM, Guido; DRAKE, Fred L. **Python 3 Reference Manual**. Scotts Valley, CA: CreateSpace, 2009. ISBN 1441412697.

WANG, Meng et al. An Efficient Service Function Chain Placement Algorithm in a MEC-NFV Environment. In: p. 1–6. DOI: 10.1109/GLOBECOM38437.2019.9013235.

WANG, Ziyu et al. Dueling network architectures for deep reinforcement learning. In: PMLR. INTERNATIONAL conference on machine learning. [S.l.: s.n.], 2016. P. 1995–2003.

WATKINS, Christopher JCH; DAYAN, Peter. Q-learning. **Machine learning**, Springer, v. 8, n. 3-4, p. 279–292, 1992.

WILSON, Callum; RICCARDI, Annalisa. Improving the efficiency of reinforcement learning for a spacecraft powered descent with Q-learning. English. **Optimization and Engineering**, out. 2021. ISSN 1389-4420. DOI: 10.1007/s11081-021-09687-z.

WONG, Esther; LEUNG, Kin; FIELD, Tony. State-space decomposition for Reinforcement Learning, 2021.

WU, Yuhuai et al. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In: ADVANCES in neural information processing systems. [S.l.: s.n.], 2017. P. 5279–5288.

XIAO, Yikai et al. NFVdeep: Adaptive Online Service Function Chain Deployment with Deep Reinforcement Learning. In: PROCEEDINGS of the International Symposium on Quality of Service. Phoenix, Arizona: Association for Computing Machinery, 2019. (IWQoS '19). ISBN 9781450367783. DOI: 10.1145/3326285.3329056. Retrieved from: https://doi.org/10.1145/3326285.3329056. Accessed on: 15 jun. 2021.

XIONG, Zehui et al. Deep reinforcement learning for mobile 5G and beyond: Fundamentals, applications, and challenges. **IEEE Vehicular Technology Magazine**, IEEE, v. 14, n. 2, p. 44–52, 2019.

ZHANG, Jiao et al. Enabling efficient service function chaining by integrating NFV and SDN: architecture, challenges and opportunities. **IEEE Network**, IEEE, v. 32, n. 6, p. 152–159, 2018.

ZIMMERMANN, Hubert. OSI reference model-the ISO model of architecture for open systems interconnection. **IEEE Transactions on communications**, IEEE, v. 28, n. 4, p. 425–432, 1980.