Cognitive routing of flows in software-defined networks from the control plane

Undergraduate Work

Sofía Rubin Castillo
Brayam David Otero Pomeo

Advisor: PhD. Oscar Mauricio Caicedo Rendón
Co-advisor: PhD. Cristhian Nicolás Figueroa Martínez

Telematics Department
Faculty of Electronic and Telecommunications Engineering
Universidad del Cauca
Popayán, Cauca, 2023

Cognitive routing of flows in software-defined networks from the control plane

Sofía Rubin Castillo
Brayam David Otero Pomeo

Undergraduate work presented to the Faculty of Electronic and Telecommunications Engineering of the University of Cauca to obtain the title of:
Electronic and Telecommunications engineer

Advisor: PhD. Oscar Mauricio Caicedo Rendón
Co-advisor: PhD. Cristhian Nicolás Figueroa Martinez

Telematics Department
Faculty of Electronic and Telecommunications Engineering
Universidad del Cauca
Popayán, Cauca, 2023

Acknowledgment

For us is important to start sharing how grateful we are for our friendship; without it, we couldn't have found the light in the thought days as students, for the moments which will last forever in our memories, for the afternoons of laughter, for the many the hugs of support when everything seemed gray. In that sense, we also want to thank our closest friends for the companionship we had in an essential time like this, when human beings forge their worth for the rest of their lives. We want to thank our families for how fortunate we are for their support and understanding in the process. We thank the whole University del Cauca's Electronic Engineering and Telecommunication faculty, as it is a place where our voices are heard, and since the beginning, we have felt comfortable and encouraged. We especially thank our advisors, Oscar M. Caicedo and Cristhian N. Figueroa, for directing our vision and sharing essential lessons that will stick with us through our academic path. Also, we want to thank Daniela M. Casas-Velasco for her disposition to talk about her work and for inspiring our own.

And, finally remind the importance of mental health. A fundamental part of life, since an eased soul can create the best of works, always heading toward real aware advancement.

Agradecimientos

Para nosotros es importante manifestar lo agradecidos que estamos por nuestra amistad; sin ella, no hubiésemos podido encontrar la luz en los días difíciles como estudiantes, por los momentos que perdurarán para siempre en nuestra memoria, por las tardes de risas, por los tantos abrazos de apoyo cuando todo parecía gris. En ese sentido, también queremos agradecer por nuestros amigos cercanos a quienes tuvimos la fortunar de conocer un momento esencial de nuestras vidas, en el que el ser humano forja sus valores para el resto de su vida. Queremos agradecer a nuestras familias, somos afortunados por su apoyo y comprensión en el proceso. Agradecemos a la Facultad de Ingeniería Electrónica y Telecomunicaciones de la Universidad del Cauca, al ser un lugar donde nuestras voces son escuchadas y desde el inicio nos hizo sentir cómodos y alentados. Agradecemos especialmente a nuestros directores, Oscar M. Caicedo y Cristhian N. Figueroa, por dirigir nuestra visión y compartir lecciones esenciales que nos acompañarán a lo largo de nuestro camino académico. Asimismo, queremos agradecer a Daniela M. Casas-Velasco por su disposición para hablar de su trabajo y por inspirar el nuestro.

Y, finalmente recordar la importancia de la salud mental. Parte fundamental de la vida, ya que un alma aliviada puede crear la mejor de las obras, encaminándose siempre hacia un avance real consciente.

Summary

Flow routing algorithms are a fundamental part of telecommunication networks since they allow all nodes to communicate with each other. Routing algorithms must be efficient to avoid network's degradation. Traditional routing protocols usually use fixed link weight assignment and shortest path routing which that cause link overuse resulting in the service degradation. The appearance of Software Defined Networking (SDN) offered multiple advantages over traditional networks, such as a global view of the network and a programmable control and data planes entitling the introduction of new technologies.

In a first attempt, traditional routing protocols were implemented on the top of SDN, improving routing convergence. Still, they inherited limitations such as link overuse and did not consider the network's historical information to nourish decision-making. SDN's programmability made it possible to integrate Machine Learning (ML) techniques to take advantage of the network's historical information opening the way for novel routing strategies. However, ML-based solutions are still dependent on conventional routing protocols, therefore, do not fully exploit the network status. We propose a routing algorithm whose cognitive learning improves network performance and takes advantage of the broad potential of SDN.

Resumen

Los algoritmos de enrutamiento de flujos son una parte fundamental de las redes de telecomunicaciones ya que permiten la comunicación entre los nodos. Los algoritmos de enrutamiento deben ser eficientes para evitar la degradación de los servicios de red. Los protocolos de enrutamiento tradicionales generalmente usan asignación de pesos fijos en los enlaces y estrategias de enrutamiento como la priorizar la ruta más corta, esto, causa un uso excesivo de los enlace resultando en la degradación del servicio. La aparición de SDN ofreció múltiples ventajas sobre las redes tradicionales, como la visión global de la red y planos de control y datos programables que dieron paso a la introducción de nuevas tecnologías.

Inicialmente, se implementaron protocolos de enrutamiento tradicionales sobre SDN, mejorando la convergencia de enrutamiento. Aún así, estos heredaron limitaciones como el uso excesivo de enlaces y no consideraron la información histórica de la red para nutrir la toma de decisiones. La capacidad de programación de SDN hizo posible integrar técnicas de ML para aprovechar la información histórica de la red y abrió el camino a nuevas estrategias de enrutamiento. Sin embargo, las soluciones basadas en ML todavía dependen de los protocolos de enrutamiento convencionales y, cuando no utilizan los algoritmos de enrutamiento convencionales, no aprovechan completamente el estado de la red. Por lo tanto, proponemos un algoritmo de enrutamiento cuyo aprendizaje cognitivo mejora el rendimiento de la red y aprovecha el amplio potencial de SDN.

Content

List of figures

List of tables

List of acronyms

AI	Artificial Intelligence
API	Application Programming Interface
ANN	Artificial Neural Networks
BGP	Border Gateway Protocol
CoRA	Cognitive Routing Algorithm
DL	Deep Learning
DRL	Deep Reinforcement Learning
DPG	Deterministic Policy Gradient
DQN	Deep Q-Networks
DNN	Deep Neural Network
DDPG	Deep Deterministic Policy Gradient
DRSIR	Deep Reinforcement Learning and Software-defined networking Intelligent Routing
DDQN	Double Deep Q Network
DQL	Deep Q-Learning
EWBI	East/Westbound Interfaces
ECMP	Equal-Cost MultiPath Routing

IoV	internet of Vehicles
IRTF	Internet Research Task Force
ITU	International Telecommunication Union
I2A	Imagination-Augmented Agents
IP	Internet Protocol
IoV	The Internet of Vehicles
IRBRL	Intelligent Routing and Bandwidth Allocation System with Reinforcement Learning
JSON	JavaScript Object Notation
KDN	Knowledge-Defined Networking
LLDP	Link Layer Discovery Protocol
LSA	Link State Advertisement
ML	Machine Learning
MI	Management Interface
MBMF	Model-Based Model-Free
MBVE	Model-Based Value Estimation
NSAF	Network Situation-Aware Framework
NAT	Network Address Translation
NOS	Network Operating System
NBI	NorthBound Interfaces
ONF	Open Networking Foundation
OAM	Operations, Administration, and Management

OSPF	Open Shortest Path First
OFBGP	Open Flow Border Gateway Protocol
DROM	DDPG Routing Optimization Mechanism
SA3CR	SDN Asynchronous Advantage Actor-critic Routing
QoS	Quality of Service
REST	REpresentational State Transfer
RL	Reinforcement Learning
RIP	Routing Information Protocol
RSIR	Reinforcement Learning and Software-defined networking Intelligent Routing
SBI	SouthBound Interfaces
SDN	Software Defined Networking
SLA	Service Level Agreements
SPF	Shortest Path First
TIDE	Time-relevant Deep reinforcement learning
TC	Traffic Control
UDP	User Datagram Protocol

Chapter 1

Introduction

SDN manages networks effectively, reduces the operation costs, and promotes the development of networks through programmability [6, 7]. Therefore, SDN has been the focus of growing attention due to its efficiency and scalability to handle high traffic volumes in highly complex networks [8]. The potential of SDN compared to its conventional counterparts highlights the importance of reassessing how networks are implemented and the algorithms used in them for their operation since the search for effective routing plays an essential role in increasing SDN performance.

The use of Shortest Path First (SPF) [9] or link state routing algorithms within conventional routing protocols in SDN favors situations of congestion and degradation of the network environment by using a rigid allocation of weights in the calculation of routes [10]. Some solutions implement conventional routing protocols such as Open Shortest Path First (OSPF) [11–15], Routing Information Protocol (RIP) [16] or Border Gateway Protocol (BGP) [17–20]. Although these solutions achieve improvements regarding delay and packet loss compared to implementations in traditional network architectures, these solutions incur limitations when making routing decisions in the network. Fixed metrics in rigid weight assignments for route calculation are a drawback when facing intrinsic network variability in current environments. Therefore, the service ends up degrading due to link congestion. Furthermore, these protocols do not use prior experience in making routing decisions.

With time and the contants search for new ways to exploit SDN in particular, the programmability of SDN was seen as a routing solution toghether with ML capabilities in data processing, and decision-making [21]. Some studies [22–28] propose routing solutions based on Reinforcement Learning (RL) within SDN to find an optimal routing policy in terms of delay, packet transmission rate, packet loss rate, among others. Despite seeing an optimal policy in an RL algorithm requires many iterations, which can hamper proper routing algorithm performance in continuously time-varying environments with large node and link number [29].

Works like [30–42] are based on Deep Reinforcement Learning (DRL). These works create routing algorithms that adapt themselves to the state of the network and improve the distribution of resources, optimizing network performance in terms of delay and packet transmission rate.

The contributions above show different approaches related to routing, and thus, we can spot a couple of common factors to address. For example, we have seen a need for real topology usage and appropriate traffic matrices to nourish the implementation and evaluation of the routing algorithms. On the other hand, the routing decisions are prone to dismiss device state metrics. Hence, the network loses its global view by neglecting important device status metrics, using only link metrics.

This thesis aims to route packet flows according to the network's resources. Our purpose is to consider all sides of the network's status, meaning link and device metrics. Considering the limitations, this undergraduate work will guide its development starting from the following research question:

How to obtain a cognitive and efficient routing of packets in SDN considering link and device metrics from Control Plane?

During the development of the degree project, the research question was addressed considering the following hypothesis: **Cognitive routing in SDN can be achieved by analyzing link and device state metrics using DRL techniques.**

1.1 Objetives

1.1.1 General Objetive

- Propose a mechanism for cognitive routing in SDN based on DRL and link, and device state metrics.

1.1.2 Specific Objectives

- Design a DRL-based mechanism for cognitive routing in SDN considering link-state and device-state metrics.
- Implement a prototype of the proposed mechanism.
- Evaluate the mechanism through a prototype in an emulated SDN scenario regarding packets lost, delay, throughput and device status [1].

1.2 Contributions and Scientific Production

This degree work achieved the following contributions:

- A DRL-based mechanism for cognitive routing in SDN considering link-state and device-state metrics.
- An implementation and evaluation of the proposed mechanism over an emulated SDN scenario considering packets lost, delay, throughput and device status.
- The execution of the proposed algorithm over an existing topology and feed the knowledge with real traffic matrices.

[1]The queue occupation at the input of the switch was used to characterize the device status

1.3 Document Organization

This undergraduate work has been divided into the chapters described below.

- Chapter 1 presents the **Introduction** which includes the statement of the problem, Objectives, Contributions and Scientific Production, and the Organization of this document.
- Chapter 2 presents the **Background** organized by the **Theoretical Framework** on the relevant topics related to the research carried out including Routing, SDN, ML, and KDN.
- Chapter 3 introduces the **Related Work** divided in Classical Routing, RL-bases Routing, DRL-based Routing, and finally the research gaps.
- Chapter 4 presents the **Cognitive Flow Routing Algorithm for Programmable Control Plane**, and Overview, Architecture, the DRL-agent, and its algorithm.
- Chapter 5 addresses the **Results and Analysis**
- Chapter 6 presents the **Conclusions** obtained, the **Future work** and **Final remarks**.

Chapter 2

Background

In this chapter, we provide a description of fundamental concepts. We categorized routing dividing it in classical and modern routing concerning the nature of the algorithms use for the routing task, besides, SDN architecture, ML categories, and the Knowledge-Defined Networking (KDN) paradigm.

2.1 Routing

Routing is the process responsible for finding the route that information packets must follow from a source node to a destination one [43]. Routing is developed by assigning routes statically or dynamically. The route is statically configured manually by the network administrator usually in small networks [44]. Instead, dynamic routing uses heuristic algorithms to find the route either as the shortest path or by considering the link-state in large scale networks [45].

Due to SDN and its programmability, ML has endured an essential role within the network environment. As a consequence, we classify the routing algorithms into two categories; classical routing and modern routing.

- **Classical Routing.** Use algorithms such as distance vector, path vector, or link-state (e.g., Dijkstra) to find optimal path [46]. With the implementation of the distance vector algorithm, the routing takes into account the shortest path from the origin to the destination. Path vector algorithms are similar to distance vector algorithms in that they consider the number of hops between the source and destination; however, how the path is shared changes. On the other hand, link-state algorithms consider the weight of the link to choose the optimal route [47]. Traditional routing protocols as OSPF, RIP and BGP implement these algorithms [48].

RIP uses the distance-vector routing algorithm. Accordingly, distance is measured in hops from one device to another. So, to define the optimal route, the hop number to the destination node is considered [49, 50]. RIP evicts loops paths limiting the number of hops to 15, so if a route has 16 jumps is considered a non-optimal route. The devices configured with RIP send broadcast packets every 30 seconds; the devices then respond with the complete routing tables. The routing tables have the destination Internet Protocol (IP) addresses and the hop number of arriving at that destination. This process iterates till every route table is actualized.

OSPF uses link-state algorithm to find the optimal path [51]. The devices configured with OSPF send Link State Advertisement (LSA) to neighbor devices, and a database stores the information containing the link states from all the topology. Considering the link states database collected, each link has a weight allowing Dijkstra to calculate the optimal route. BGP uses path vector algorithm ,where the configured devices with BGP exchange available routes with their neighbor devices. Furthermore, the routes share the cost in relation to the available bandwidth, latency, and hop number from origin to destination node [52].

Classical routing is used in traditional networks due to its low computational cost and speed when calculating the optimal route [53]. The simplicity of the criteria of these algorithms leads to congestion scenarios and a waste of network bandwidth [10]. In addition, routing protocols based on link state and distance vector algorithms do not learn from previous experience, negatively affecting network causing degradation scenarios [54].

- **Modern Routing.** How routing is perceived has changed in recent years with software tools heading in many fields and their introduction to telecommunication networks. Thus, modern routing emerges fundamentally in SDN, allowing a glimpse of solutions not considered in traditional networks. The modern routing approach is linked to Artificial Intelligence (AI) applications and ML techniques. With SDN flexibility playing the main role in integrating ML strategies, making routing adaptable, scalable, and providing a wide variety of possible solutions as suggested by Amin, R. et.al [55].

The routing process achieved with ML in the communication networks, outperforms classical routing a like demostrates Valadarsky, A. et.al [56], although with a high computational cost. Therefore, modern routing implementations must have high throughput devices [3].

2.2 Software-Defined Networking

SDN is a network architecture that physically decouples the Control Plane from the Data Plane [6]. Decoupling the network infrastructure allows centralized control of network functions, simplifying management and reducing the operating cost of the network [57]. The usage of SDN allows a flexible and scalable network due to its programmability [58]. The latter draws the attention of academia and industry by facilitating diverse research proposals that introduce all kinds of computer science techniques such as ML or AI within this kind of networks [7].

Organizations such as Open Networking Foundation (ONF) [59], Internet Research Task Force (IRTF) [60], and International Telecommunication Union (ITU) [61] are responsible for standardizing the SDN architecture. Estrada-Solano, F. et.al [1] propose a high-level architecture, as shown in Figure 2.1.

The SDN architecture involves three horizontal planes [62]: Data Plane, Control Plane, and Application Plane. Also, it comprises a vertical plane: Management Plane [1].

- **Data Plane** located at the bottom of the architecture, where its primary function is packet forwarding. Is the layer of SDN that contains the net-

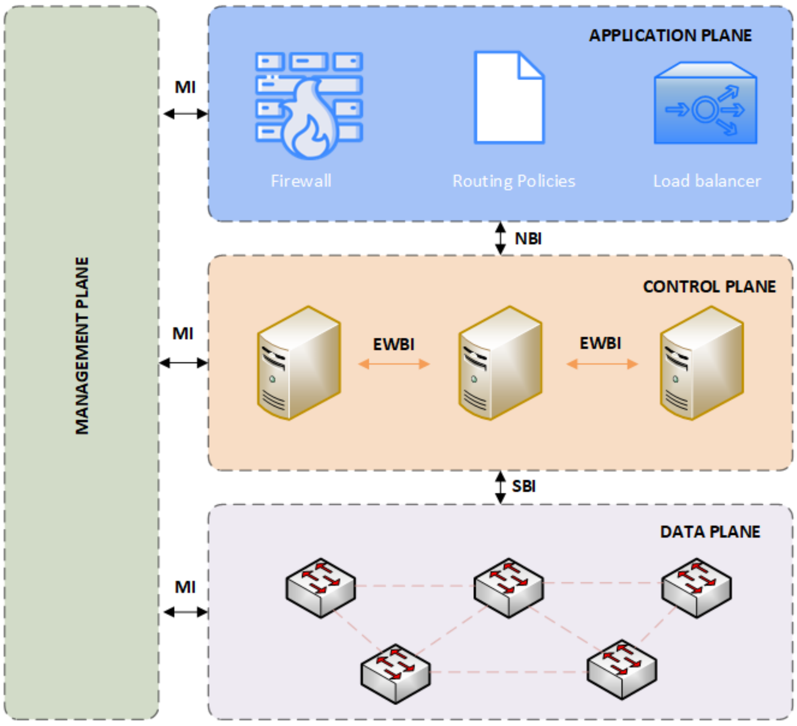


Figure 2.1: High-level SDN architecture [1]

work devices that physically handle network traffic. Novel approaches suggest adding programmable capabilities to the Data Plane. However, the essential way to work is to take the Control Plane instructions and pass them down to the Data Plane.

Some switches exclusively carry forwarding functions like the OpenFlow-Only [63] switches, which use their flow tables to follow the routing rules and carry out the forwarding process. While the control plane executes the flow table update installing the tables on the switches. Custom switches are available on programmable platforms (e.g., Open-Wrt and NetFPGA) allowing more advanced network functions than packet forwarding, such as Network Address Translation (NAT) and even implementing firewall [64].

- The **Control Plane** is the middle layer between the Data Plane and the Application Plane. This plane is composed of at least one instance of the Network Operating System (NOS) or controller. The controller is connected to all the data layer devices, representing the logical part of the plane in a centralized way. Therefore, a single entity has a global network view and can reconfigure the flow rules, a set of rules that enables the algorithm to prioritize specific routes from the routing tables.

The controller provides generic functions such as device discovery and network configuration in a distributed way, and it can also obtain network topology and network state information for decision-making. Furthermore, the controller allows the Application Plane to implement specialized services. The controller receives from the Application Plane the service policies through a high-level language or an Application Programming Interface (API). Then the controller translates this policy into rules that finally install or update in the Data Plane.

The SDN controller installs well-defined instructions for handling incoming packets on Data Plane devices, making it possible for a simple device to become a router, switch, firewall, or load balancer.

Multiple controllers can be implemented on the Control Plane of a SDN. One is NOX, a component-based controller which works with Python and C++ [65]. Besides, there's Beacon [66], Maestro [67], OpenDaylight [68], and Floodlight [69] that use Java to configure and create new controller characteristics. Finally, a relevant a well-reputed is the Ryu controller, a component-based controller that is highly scalable and used in wide-size networks. Ryu is flexible due to its easy use and configuration, furthermore has an active community that keeps supporting its use and documentation, helping its development [70].

- The **Application Plane** located at the top of the architecture. Possesses the network applications or network services implemented in the underlying planes. Such network applications are, for instance, firewall, access control, quality of service, routing, proxy service, and monitoring balancer. Due to the centralized logic of SDN, this plane can abstract the global state of the network to generate coherent service policies (e.g., optimal routing, Distributed Denial-of-service attack prevention), which is translated and applied to the Data Plane devices by the controller. Applications or network services on the Application plane can be developed within the NOS instance or outside of it, so a standardized form of communication is necessary.

- The **Management Plane** is transversal to the other planes of the architecture, allowing the implementation of Operations, Administration, and Management (OAM) functions in all planes. By adding this plane, it is possible to configure a particular Data Plane device to connect to a specific NOS, manage the different instances of NOS in the Control Plane, or modify Service Level Agreements (SLA) in the application plane.

These planes communicate through the following interfaces [71, 72]:

- The **Management Interface (MI)** connects the three planes, sending the devices' configurations to the data and Control Planes. In addition, it modifies or adds the SLAs of the network services in the Application Plane.
- The **East/Westbound Interfaces (EWBI)** connects the different existing controllers in the Control Plane to obtain a centralized logic since each controller can manage a part of the Data Plane devices or a particular network domain. Splitting the network achieves greater network scalability but creates synchronization challenges between controllers.
- The **NorthBound Interfaces (NBI)** communicates the global state of the network delivered by the Control Plane to the Application Plane for further processing and decision-making. In turn, this interface transfers the policies generated by the Application Plane to the Control Plane. The communication is through a high-level language or a REpresentational State Transfer (REST) API.
- The **SouthBound Interfaces (SBI)** disseminates new flow rules or updates existing Control Plane rules to install them on Control Plane devices. Also, it allows the Control Plane to collect the state of the different network resources of the adjacent devices. OpenFlow is the most widely used protocol in industry and academia [73].

2.3 Machine Learning

ML techniques are categorized according to how learning is acquired; supervised, unsupervised, and reinforcement [74]. In supervised learning, the algorithm is provided with example input-output pairs, so the algorithm has to discover a function that calculates the input to the labeled outputs. At the end of the supervised learning process, the algorithm will infer a function that adjusts to the desired outcome and can map new examples. In contrast, unsupervised learning captures patterns from a density of untagged data; in other words, the machine is forced to build its representation from the environment and generate resourceful content adjusting its biases and weight on the learning. Finally, the algorithm responds to a reward system in RL until it obtains the optimal action policy to achieve the desired efficiency in a environment [75]. The characteristics of RL are ideal for solving problems in complex environments such as networks since its cognition is an advantage in terms of making routing decisions [76].

Reinforcement Learning. It is a highly cognitive technique since the agent becomes intelligent through the algorithm iteration. This agent acquires high-quality management policies with little or no prior knowledge of the environment. This means that the foundation of this technique is experience and not the construction of a mathematical model [31], being ideal for environments that cannot be modeled due to the influence of numerous components [77]. Through a state-action pair, the RL agent acquires a characterization of its environment and, given the experience, decides to move to the next state. Finally, it obtains a reward that, depending on the success of the decision, feeds back to the RL agent's policy [78].

The objective of a RL agent is to maximize the cumulative reward function, defined as the possible reward values for all the scenarios of the state-action pair. The cumulative reward function is the possible reward values for all the scenarios of the state-action pair. And the objective of an RL agent is to maximize it through exploration and feedback. The RL agent learns the best policy whose use would allow the maximization of the cumulative reward function [79]. Fadlullah, Z. M. et .al [80] highlights RL as the most suitable ML technique for decision-making. In combination with Artificial Neural Networks (ANN) techniques, RL improves the performance related to its function of value action, acquiring more clever solutions.

Reinforcement Learning algorithms: The branching of RL has two broad categories; model-based and model-free. These categories tell whether the algorithm would perform regarding a constructed model or act freely on the immediate output.

- **Model-based RL algorithm.** This algorithm access experience to set up a model that will govern the learning and subsequent actions. Thus, the prediction of the environmental response modifies the agent's behavior. These algorithms develop the ability to plan ahead the action between various choices generated thanks to the modeled policy. Thus, allowing the algorithm to converge rapidly under the constructed model [81]. Some examples are World Models, Imagination-Augmented Agents (I2A), Model-Based Model-Free (MBMF), Model-Based Value Estimation (MBVE), Alpha Zero, Alpha Go.
- **Model-free RL algorithm.** Use experience to build and adjust a policy without considering environmental information beforehand. This means the algorithm cannot be biased, and the outcome is constantly adjusted. These algorithms have proven exceptional performance in complex environments with numerous tasks, and wide state space [81]. Simultaneously Model-free algorithms can be divided into two popular groups, Policy optimization and Q-Learning. Currently, each group has numerous approaches and developing proposes as Model-free algorithms have gained relevance over time.

Artificial Neural Networks. ANNs are an ML technique of great relevance due to their virtues in processing massive datasets and extracting their features with high processing speed, convergence, and precision [82]. This technique simulates the neural networks of human beings, where its basic unit is called the neuron. Neurons are connected and located in layers of various depths depending on the precision required [83]. Connections between layers have weights, and learning occurs by modifying those weights to generate a specific output [84]. The word "deep" in the field of ML is related to the number of layers needed alluding to the deepness of layers hidden. Deep Learning (DL) belongs to the ML methods based on the ANN with various architectures such as Deep Neural Network (DNN), deep belief networks, DRL, or recurrent neural networks where in many fields that have been applied have surpassed human expertise. Implementing DNNs in RL helps process a large amount of data. In ML approaches, the DNNs successfully supply well-characterized information to the agent, favoring strategies through speed and effectiveness in the convergence of these algorithms [85].

Deep Reinforcement Learning. It is an approach that integrates DL and RL, resulting in better agent performance in environments with a large volume of states and actions [86]. The enhancement in performance is due to the faculty of RL for decision-making without human surveillance, counted on the competence of ANN when it comes to analyzing and processing large volumes of information optimally. As shown in Figure 2.2, the ANN is inside the DRL agent, where it receives the state of the environment and the result of the reward function to obtain the next action to perform.

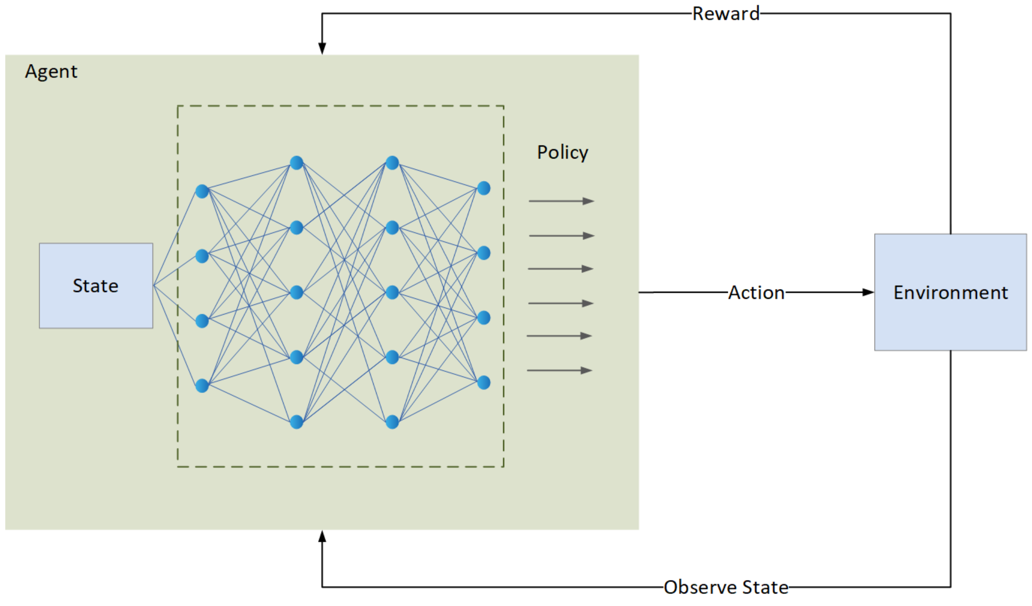


Figure 2.2: DRL operation. [2]

Deep Reinforcement Learning techniques: The classification of the different DRL techniques organizes according to the way they select a set of actions that maximize the reward, either based on value or policy [87]. In value-based techniques, state-action pairs are assigned a value according to their past states and rewards, thus indicating how advantageous it is to apply an action in a given state. With the integration of DL in RL, the DNN approximates the assignment of the state-action value.

- **Deep Q-Networks (DQN).** It has a ANN that receives as input the agent's current state and returns the Q-values of all the actions that the agent can take in that state [88].
- **Double DQN.** It proposes two ANNs with the same architecture but different weights; one ANN calculates the best action between the action space, and the other neural network evaluates the Q-value of the action, achieving thus stability in the training stage [89].
- **Dueling DQN.** Like double DQN presents two ANNs, the first one approximates the impact of the current agent's state, and the second deals with the utility of the action on the action space [90].
- **Deep Deterministic Policy Gradient (DDPG).** is a DRL algorithm close to Q-learning, given that it learns the Q-function in addition to an optimal action policy.DDPGis adapted for environments with continuous action spaces, but it is not a shortcoming for a discrete number of action spaces because the actions can be computed by getting the Q-value and comparing them. DDPG, similarly to the actor-critic method, uses an Actor ANN that proposes an action for a state and a Critic ANN that predicts the quality of the action for a state, in addition to the benefit of stability from two Target ANN. The Target ANNs grant the training with stability by holding a target and being updated slowly, hence keeping the estimating targets for a time while training develops being supported on those targets. AnotherDDPGtechnique is Experience Replay, which allows past experiences to disseminate feedback to the agent by learning from samples of historical information.

DRL is a powerful tool to face various challenges in communication networks, such as packet routing, resource allocation, and security, among others [29]. However, its application in traditional networks is very complex since each network node has a partial view and control. Therefore, centralized control and more significant cognitive ability are necessary to make agent learning a more effective process.

2.4 Knowledge-Defined Networking

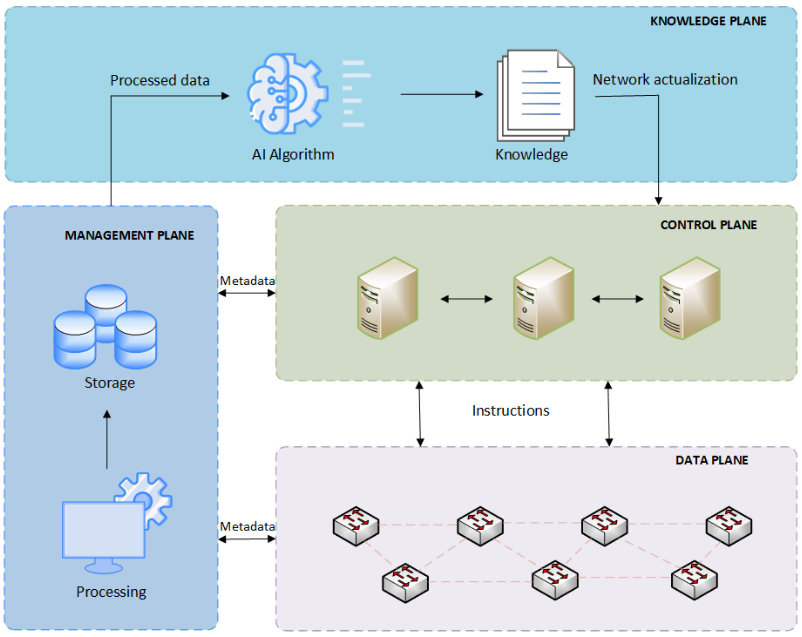


Figure 2.3: KDN architecture based in [3] proposal

As mentioned, SDN is an architecture that opens many possibilities for new paradigms. A promising one is KDN. That relies upon the ML cognitive techniques over SDN, arguing that this cognitive approach is best suited for resolving the complexities of the network environment. KDN was first proposed in [3] introducing a Knowledge Plane within SDN [91]. KDN is inherently distributed and based on tools provided by ML. It has become a powerful tool because of the growing developments and advantages in operation and network control.

[3] merge ML and cognition in the network through a Knowledge Plane to achieve an autonomous network control. In the real world, tasks are not divided but unified as knowledge to carry them over. In that sense, knowledge maintains a global point of

view through the Knowledge Plane, extending its point of view to the entire network and providing direction and awareness about the network environment [91]. The Knowledge Plane would have representation, learning, and reasoning capabilities that enable the knowledge to be aware of the network's states and actions in a cognitive manner.

Due to the global view and programmability of SDN, together with the telemetry and data analysis techniques that KDN incorporates into the network, it is possible to increase the processing of a large amount of information in real-time and to have better dimensioning of the network.

The KDN architecture distinguish into four planes (Figure 2.3). In the lower part is the Data Plane contains the network infrastructure devices for routing, processing data packets, and executing routing tables. In the center is the Control Plane, maps the topology and install the routing tables on the Data Plane devices. The Management Plane is transversal to the previous planes; it saves, processes, and normalizes the network's metadata that will be further added to the knowledge and Control Plane. The Control and Management Planes are transversal to the network layers, unluckily, when scaling network data they encounter difficulties despite its transversal view.

The inputs of the Knowledge Plane are the status information gathered by the Management Plane. The Knowledge Plane breaks boundaries to provide a conscious, unified view, rather than the partitioned management, control, and Data Plane [3]. The Knowledge Plane exploits Control and Management Planes through ML-based approaches capable of comprehending the network behavior accurately [38]. In other words, KDN is worthy for the network's intelligence since it contains the ML algorithm responsible for routing cognition and decision-making. In works like [92], research widely KDN as an indispensable evolutionary step toward autonomous self-driving networks in fields like new-generation wireless networks. As shown in [38], a DRL-based agent with convolutional neural networks in the context of KDN can improve the execution in QoS-aware routing. Both [5] and [40] show KDN as an essential part along the coherent implementation of ML in SDN, [5] is an RL algorithm that outperforms conventional routing algorithms, and its following proposal [40] a

DRL agent over a Knowledge Plane continue to surpass [5] performance among other routing approaches confirming that KDN is relevant and suitable for any ML technique.

Chapter 3

Related Work

This chapter describes the most representative works regarding routing based on SDN, ML, and device metrics. Among these are mechanisms addressed through different models and strategies that allow confronting the diverse solutions and approaches we have developed for this undergraduate work.

3.1 Classical Routing

[11] describes the advantage of SDN routing in large-scale networks over legacy routing mechanisms regarding convergence time and packet forwarding delay. In the same way, [13] compares legacy OSPF-based networks with SDN, SDN turning out to be a more competent approach even when large-scale networks encounter higher link delay. Similarly, [12] study how a dynamic routing protocol as OSPF performs inside SDN, analyzing the stability of the network in parameters like Quality of Service (QoS) on video streaming, showing favorable results regarding SDN holding conventional routing protocols improving convergence time versus the usual network architecture.

As fast network convergence became a vital feature on networks, [14] evaluates the impact on a OSPF based distributed network and a centralized SDN having as a result that each outperforms the other under different conditions depending on network size and link delay. [15] use OSPF in SDN to improve the QoS of large-scale networks in terms of delay, packet transmission rate, and packet loss. Research [17] aims to continue the use of classical routing protocols constructing a BGP implementation on SDN to transition the existing networks to SDN being a feasible approach compared with its conventional network implementation. Particularly, [18] uses SDN's programmability to adapt BGP so that the distance route travel is less, resulting in reduced latency and stretch of the paths. [20] proposes a BGP architecture called Open Flow Border Gateway Protocol (OFBGP), implemented as an SDN application, improving scalability and availability regarding to the conventional BGP routing algorithm. [19] approaches BGP limitations like fully distributed network, rigid scalability, and complexity by proposing an incrementally deployable internet routing paradigm in which the Control Plane is logically centralized following SDN architecture using its global view to benefit convergence times and churn rates on this BGP proposal. Nevertheless, SPF-based protocols have difficulty facing network variability, congestion, and growing sizes, because of their rigid, complex system, falling into limitation when challenged to adapt to the fast-growing current networks.

The Dijkstra link-state routing algorithm traditionally considers the shortest path or performance metrics (e.g., delay rate, packet transmission rate, and packet loss) to assign costs to network links. Dijkstra is used in solutions like [93], which presents a SDN control framework for QoS provisioning in prioritized flow using Dijkstra to calculate the route. Likewise, [94] Proposes Network Situation-Aware Framework (NSAF) to handle routing in changing network status considering the QoS metrics but instead of using the SPF protocols to calculate the routes, uses Dijkstra setting weights according to the QoS costs. And [95] uses an extended Dijkstra within SDN to compare its performance with the original Dijkstra algorithm regarding end-to-end latency, showing that Dijkstra together with SDN outperforms the original implementation.

These proposals create strategies that catch SDN flexibility advantages, such as dynamic weight assignment to link or network nodes, benefiting performance by reducing network latency and congestion. Nevertheless, even when speaking about dynamic monitoring in routing algorithms, the weight assigned to the links or nodes does not update in real time, creating a flaw in representing the network's current state. The routing decisions of the works mentioned in this section are calculated through conventional routing algorithms, which take into account only the network's current state, missing the opportunity to exploit the previous state and subsequently missing knowledge about previous decisions to generate more intelligent choices in the future. These shortcomings impact when aiming to reach prediction capabilities or a more competent global view.

3.2 Reinforcement Learning-based Routing

The RL trend is prevalent in SDN routing because RL does not need prior information on the network state. Proposals like [22] approach the hybrid surfacing techniques between ML and ad-hoc networks and improve them by merging existing techniques for wireless networks with multi-agent RL proactively updating routes and outperforming the conventional ad-hoc routing algorithms under QoS constraints. Besides, [23] propose a QoS-aware adaptative routing algorithm on SDN, proves that using RL achieves time-efficient, and QoS-provisioning in terms of packet delay, loss, and throughput. The proposal [24] is an SDN implementation in the The Internet of Vehicles (IoV) framework, where the use of a large amount of sensor data is required to be handled in real-time. The results show that this proposal use of RL to overcome the limitations efficiently coming through several other IoV techniques. In the Q-learning side, [25] a Q-learning algorithm for unicast routing in SDN designed to minimize the delay that unicast traffic suffers while on the network circulation. Results compare Dijkstra's performance in the same condition, the proposal being more effective than Dijkstra. SDN has an important role when solving problems caused by central management. Still, over SDN, the algorithms that are mainly used are Dijkstra-like algorithms generating network congestion since bandwidth overhead is not considered when a lot of traffic is circulating on the network; that's why [26] argues and demonstrates how a Q-learning routing algorithm can improve congestion and even prevent it. Also, [27] proposes an intelligent routing and allocation scheme named Intelligent Routing and Bandwidth Allocation System with Reinforcement

Learning (IRBRL), the same as the mentioned algorithms, uses SDN architecture and RL to create routing policies, dynamic routing, and link bandwidth allocation awareness, in other words, the globally visible network architecture makes the best routing and bandwidth allocation policies with reinforcement learning, as conclusion the RL routing algorithm actively modified to adapt itself to a changing requirement. Similarly, [28] is a RL on SDN proposal that aims to prevent network congestion and link-overutilization, focusing on the perspective of routing management evaluating from different scenarios like single controller and multi-controller SDN. [5] creates an algorithm called Reinforcement Learning and Software-defined networking Intelligent Routing (RSIR) which uses the intelligence of RL and the global view of the network of SDN to compute and install optimal routes in the Data Plane, the results show the improvement of RSIR regarding stretch, link throughput, packet loss, and delay compared with the Dijkstra algorithm.

The above proposals surpass conventional algorithms in terms of delay, congestion, adaptability, and intelligent management of networks with more excellent QoS provisioning, managing to obtain routes on demand even in large-scale networks. However, the high number of iterations necessary to find the optimal routing strategy and the extensive tables in Q-learning based solutions result in long convergence times and make essential efficiency improvements, increased processing speed, and storage optimization. [25] suggests that in the future, implementing other intelligence techniques (e.g., DRL, ANN) will be essential to overcome the limitations mentioned above. In [26], the need for adequate flexibility and scalability becomes evident when evaluated exclusively in an environment with a fixed traffic rate and bandwidth.

3.3 Deep Reinforcement Learning-based Routing

[40] proposes a novel approach called Deep Reinforcement Learning and Software-defined networking Intelligent Routing (DRSIR) that uses a DQN agent together with the KDN paradigm and SDN to adapt dynamically to traffic changes and overcome the limitations showed in RL approaches, the results show that DRSIR

outperform RL solutions and SPF-like routing algorithms regarding stretch, packet loss, and delay. [30] also uses DRL with prediction techniques to dynamically collect optimal path information and predict traffic demand on a SDN. The proposal also minimizes latency and packet loss, achieving its goal well above the performance among other ML algorithms used for this purpose (e.g., ARMA - *Auto- Regressive Integrated Moving Average*). In [31], the authors manage a variant of DQN called Double Deep Q Network (DDQN), where the agent's reward is associated with the delay and the packet transmission rate, surpassing the performance of OSPF implementation on those metrics. the [32] algorithm uses two ANNs to treat mouse and elephant flows. Then, they set a reward for each flow according to the specific metrics required. Mouse flows take into account packet loss and average delay. Besides, elephant flows bear the loss rate and the transmission of packets as metrics. [32] proposes a Deep Q-Learning (DQL) routing strategy for data center networks on the SDN architecture. The deep Q network is trained to adapt to the different metric demands of mice and elephant flows. The results show that the proposed routing algorithm outperforms algorithms such as Equal-Cost MultiPath Routing (ECMP). The routing agent in [34] is designed to optimize routing by adapting automatically to the traffic conditions in terms of delay. The results show operational advantages and low delay compared to a routing benchmark. [42] displays a DQN multicast routing approach where the state space consists of channel matrices and explores various methods to construct a multicast tree and take into account the computational time and compares it with original RL.

Furthermore, [38] is a proposal in the context of KDN that aims to enhance routing, the result display that the use of DRL and convolutional ANN can improve network performance even in complex network environments regarding packet loss and latency. In [39], the maximization of the reward is given by the packet transmission rate between the source, destination, and the link delay. While [37] is a proposal called Time-relevant Deep reinforcement learning (TIDE), an intelligent routing algorithm for SDN that implements recurrent ANNs and then evaluated in a real transmitting network topology. As a result, TIDE demonstrated the relevance of DRL in routing optimization and compared with the traditional routing algorithms, the RL strategies can process large amounts of data and respond with a proper output following network changes. In terms of delay, TIDE improves network delay by about 9% compared with SPF. [35] uses a spatiotemporal deterministic policy gradient agent where the temporal attention mechanism helps to learn better from the transitions. The experimental results show that this method rapidly adapts to the current network environment and achieves robust convergence time. this technique is compared with state-of-the-art DRL algorithms showing a better end-to-end delay. Also, DDPG Routing Optimization Mechanism (DROM) [36] uses DDPG in SDN, improving network performance regarding delay and throughput by optimizing in a black-box method in continuous time. The results show a better convergence time and effectiveness than the existing solutions in RL and heuristics solutions. Zhang, L. et .al [41] proposes an DRL based routing framework called DSOQR a QoS-routing framework based on DRL and SDN and on-policy learner algorithm called SDN Asynchronous Advantage Actor-critic Routing (SA3CR) based in asynchronous actor-critic, and as a result, proves the validity of the framework and the better performance of SA3CR in delay, throughput, and packet loss rate compared to ECMP and a DDPG implementation. To contrast DQN and DDPG, citeIntelligent-routing develops both DRL models to optimize the packet transmission rate. Both models perform better than OSPF concerning this metric. On the other hand, when comparing the performance between these algorithms, it becomes clear that the performance of DDPG exceeds DQN, with 47% and 40% optimization in packet transmission rate, respectively. In the proposals mentioned earlier, the agent's reward exclusively involves metrics associated with the links state (e.g., packet transmission rate, delay, and packet loss) but not the network device state. Unconsidering the device state supposes a limitation regarding the global characterization of the network environment.

3.4 Gaps

Classical routing proposals in SDN use the global view of the controller to optimize the performance of conventional routing algorithms comparable to their implementations in traditional network architectures, yet, the previous network state is dismissed. On the other hand, in modern routing, there are proposals based on RL and DRL, where the initial network state generates adaptive and cognitive routing. The literature shows the need for implementations that consider the state of the devices for routing decisions. In this scenario, routing strategies capable of learning from the network environment in a globalized way are necessary. Cognitive routing algorithms are needed to provide routing efficiency [3].

Based on the encountered gaps, SDN's centralized network management, adaptive learning, and DRL's high-volume data processing capabilities, together with global sizing of network and device status, would yield decision-making to a decisive, effective, and cognitive routing. In addition, it is evident that implementing a network topology, which uses real traffic matrices, is necessary when generating an environment consistent with current networks. To be specific, the gaps that aid us in assembling a fulfilled direction have been that by using a novel ML approach, we can exploit the past information for the future to create a feedback process that supports decision-making. And that the use of Q-learning-only strategies gets short storage-wise because of the large Q-learning tables. That's why an Algorithm with that benefits merely of Q-learning can be limiting in certain action spaces. Finally, dimensioning the network dismissing device status metrics can not be seen as a global view. Our proposal aims to exploit the unlimited status, device, and link status, conveying a full global view.

Table 3.1 shows the works' gaps, separated by their respective categories. It is important to note that the acronym CR corresponds to the aforementioned classic routing.

Ref	Type of Routing			Used algorithm	Gap
	CR	RL	DRL		
[12]	✓			It implements OSPF over SDN evaluating its performance in terms of convergence time and QoS.	These algorithms do not update the state of the network in real time. They miss the previous information of the network.
[16]	✓			It proposes a method for hybrid networks based on RIP using an SDN controller in order to achieve policy-based routing.	
[18]	✓			It uses SDN to optimize and discover routes within BGP.	
[22]		✓		This algorithm fuses RL Multi-Agent techniques with wireless routing techniques.	Optimizing storage is required due to the large Q-Learning tables. These algorithms implement fixed traffic or bandwidth, resulting in invariant environments.
[23]		✓		A QAR algorithm is designed in SDN, whose Control Plane is distributed and hierarchical to minimize delay in large-scale networks.	
[24]		✓		SDN and RL are used in order to provide optimal routes from the internet environment of vehicles.	
[25]		✓		It proposes an image processing method using Q-Learning and a CNN for their classification.	
[28]		✓		RL is used to enable adaptive routing and intelligent network management to supply application-driven quality of service.	
[40]			✓	Propose a novel space of states and actions model that characterizes the state of the paths obtaining results outperforming Dijkstra and RL approaches.	The metrics used reflect only the state of the link, it is required to globalize the network state
[30]			✓	Select the best path from a DRL agent that, together with a simple heuristic algorithm, allows traffic demand to be predicted, to optimize latency, packet loss and packet transmission rate.	
[33]			✓	A routing scheme called DRL-R is proposed, which uses DQN and DDPG to obtain network routes and intelligent resource allocation depending on the types of existing flows.	
[37]			✓	A routing scheme is proposed that uses DDPG to improve the QoS of the network by modifying the reward parameters, depending on the flow requirements.	

Table 3.1: Research gaps in DRL algorithms.

Chapter 4

Cognitive Flow Routing Algorithm for Programmable Control Plane

Chapter 4 presents Cognitive Routing Algorithm (CoRA), a DDPG-based solution with novel approaches involving SDN, KDN, and global characterization for network metrics. To disclose our solution, first, we present section 4.1 an Overview, then, section 4.2 the summarized network architecture planes; Data, Control, Management, and Knowledge. Furthermore, section 4.3 depicts the operation of our DDPG-based cognitive agent with its components and features. And, finally, section 4.4 presents CoRA's process to reach the expected outcome.

4.1 Overview

In this chapter, we present our solution facing the shortcomings found in the literature regarding routing involving ML over SDN. In the Journey of researching novel approaches to enhanced routing algorithms, we have decided to use the KDN paradigm to ease management and monitoring, jointly with DRL performing over an SDN network. We engaged with the importance of global network characterization through link state and device state metrics to train the algorithm adequately, achieving efficiency and enhanced performance. As the current routing solutions using DRL agents do not directly consider the device metrics, we recover device metrics such as the switch queue´s occupancy as well as the link state, available bandwidth, delay, and packet loss to find the optimal path policy. To create a final result, first; the Control Plane collects the performance metrics, which are later processed by the Management Plane. The processed data is delivered to the Knowledge Plane, creating an environment in which a DDPG agent learns and ultimately provides the optimal route for each pair of nodes. CoRA is a proposal with a novel objective to reach a cognitive DRL agent that outperforms approaches related to traditional routing and modern routing. DRSIR, a modern routing proposal that implements KDN over DQN, was the base for our own. Furthermore, each node delivers directly the path to follow for the packets, avoiding using classic routing algorithms. DRSIR overperforms all traditional approaches [40].

4.2 Architecture

CoRA's architecture is built in SDN, which supports the network's automated management and control. At the same time, following the KDN paradigm to compute network information (i.e., link state and device state metrics) in the intelligent DDPG agent to execute cognitive decision-making in the flow routing. Below, we explain the solution's sequential operation, and Fig 4.1 depicts it in detail.

❶ The Control Plane periodically queries the Data Plane to collect network information.

❷ The Management Plane receives *Topology discovery* and *Statistics* information from the Control Plane to Process and stores the network state.

❸ The Knowledge Plane receives information from the Management Plane.

❹ The DDPG agent explores and exploits the possible routes for each source-destination node pair. Eventually gets the best routing path for all pairs of nodes in the network.

❺ The Knowledge Plane stores in the *Routes data repository* the data about the routes computed by the DDPG agent.

❻ The Control Plane installs the best routes in the flow tables of the switches.

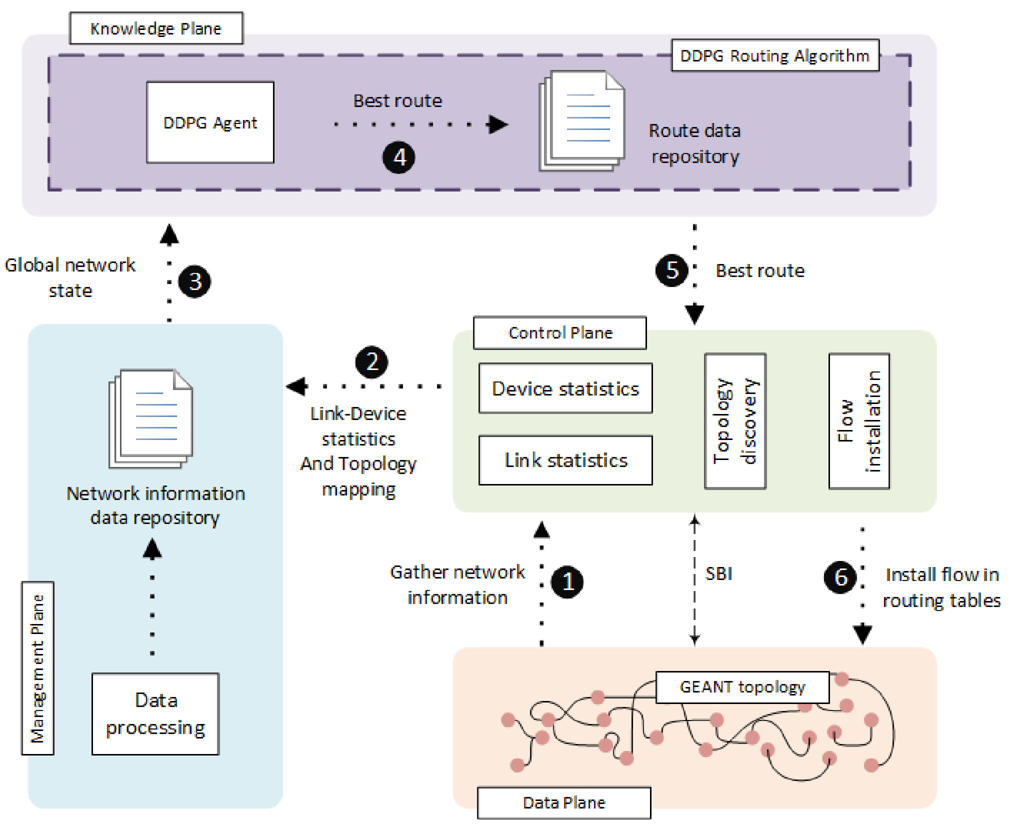


Figure 4.1: CoRA architecture

4.2.1 Data Plane

The Data Plane holds the forwarding devices and the links that connect them. Performs basic tasks like responding to queries with messages containing information about network topology and state. Essentially CoRA's Data Plane is not running complicated errands to manage the limited network resources like link bandwidth or buffer size in the devices. Instead, those resources are closely monitored to secure adequate performance and benefit the network operation. The ideal performance of the Data Plane lies in developing a sophisticated routing policy through the Knowledge Plane that is further installed in the routing tables.

4.2.2 Control Plane

The Control Plane requests queries to gather information from the Data Plane to construct the global view of the network; it is also in charge of installing the routing tables.The Control Plane comprises four modules, device statistics, link statistics, topology discovery, and flow installation.

The *topology discovery module* sends the *feature-request* message from OpenFlow to the Data Plane Switches. The Devices respond with a *feature-reply* message containing the individual switch identifier and port information such as port number and state. This module sends an OpenFlow *packet-out* with a payload containing a Link Layer Discovery Protocol (LLDP) packet to all switch ports. The first switch will send LLDP packets through the port to a neighbor switch. When the neighbor device receives the LLDP packet, it will be sent to the controller a of an OpenFlow *packet-in*. The message has the origin switch's id and the port number, besides the destination switch's id and port number, so the module can identify which devices are neighbors and from which port communicate.

The *link statistics module* sends OpenFlow's *Multipart Messages* to the Data Plane devices each monitoring period every t seconds. The *Multipart Messages* have messages types to hand over specific switch information like port statistics, flow, and flow tables. Due to the need for link statistics, the port statistics employ *port-stats* messages to consult the transmitted and received packet and byte amount in the port.

During each monitoring period, the statistics collection modules collected information on the network state and the Management Plane and carried it to feed the agent in the Knowledge Plane. The *device statistics module* uses Data Plane petitions each monitoring period; the petition's reply contains the packets' load at the switch queue. The queue at the t moment is stored with the link statistics metrics and the network topology to be subsequently processed in the Management Plane.

The Knowledge Plane delivers to the *flow installation module* the optimal path for each node pair generated according to the current network conditions. The path installation is proactive; the paths are installed periodically and change depending on the network state. This module uses the information obtained by the *topology discovery module* to configure which switch port should send the incoming flow properly.

4.2.3 Management Plane

The Management Plane comprises two components as seen in the Figure 4.2, the data processing module and the network information data repository.The data processing module receives raw data gathered by the Control Plane through the link, device statistics and topology discovery modules. This module processes the raw data to calculate every path metric (e.g., delay, packet loss, available bandwidth, and queue occupation) that gets delivered to the Knowledge Plane to enter the algorithm.

The network information data repository stores the metrics calculated by the previous module containing the tuple source destination node and its respective metrics. An example of an entry is: (source = $node_1$, destination = $node_2$, av_bw = $100Kbps$, delay = $1.3ms$, loss =0.5%, queue_occu= 10pkts). So the Management Plane ensures the optimal performance of the network over time.

4.2.4 Knowledge Plane

CoRA implements a decoupled architecture, situating the Knowledge Plane above the Control Plane and communicating directly with it and the Management Plane, avoiding overloading the Control Plane. The Knowledge Plane contains the DDPG-based routing algorithm where the DDPG-agent and the *route data repository* lie. Thanks to the Agent, the Knowledge Plane transforms information into knowledge, gathering the global view of the network and computing it to achieve the algorithm's objective. The *route data repository* holds path information; each entry is a tuple of

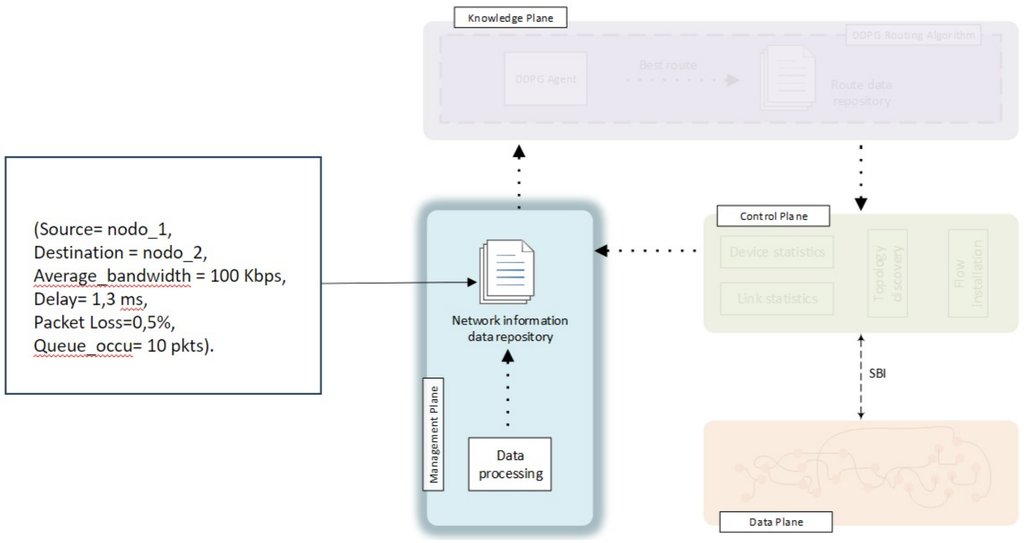


Figure 4.2: Management Plane of CoRA

source, destination, and best path, with source and destination being node numbers, and the best path is a series of nodes that shape the path under consideration.

4.3 Cognitive Routing Agent

This subsection explains how the agent can find the best route from all possible network paths. CoRA's agent uses DDPG a DRL technique that merges DQN and Deterministic Policy Gradient (DPG), an optimal combination that, along with experience replay and slow learning target ANNs make well-developt decisions, stable and efficient learning when developing cognitive policies 4.3. It is demonstrated that DDPG robustly solves problems related to complex action spaces [96], like the one that gathers our commitment in this undergraduate work. Aspiring to take a step forward in ML routing algorithms and improving the previous similar proposes in this field, we choose DDPG as CoRA's agent technique. Interacting with the network environment, the agent converges to an optimal policy for the best route for each node pair. The Knowledge Plane builds the environment using the network state information gathered by the Management Plane. Each iteration makes the agent act on the current state and choose the action that minimizes the reward; then, the agent delivers the next state. The reward corresponding to the action taken conveys the path cost, taking into account the available bandwidth, packet loss, delay, and queue occupation. That's to say, the agent's decision will be influenced by the path with greater available bandwidth, least packet loss, delay, and queue occupation. The following subsections explain the fundamental features of a DDPG agent, state space, action space, and reward process.

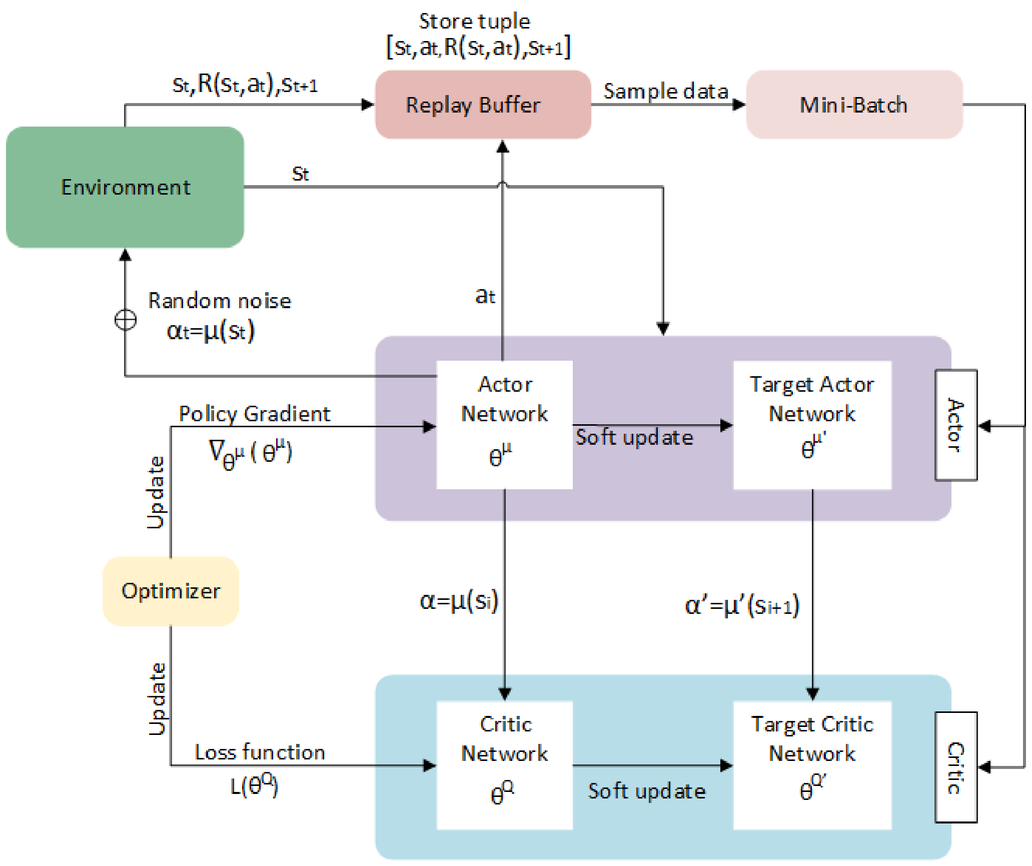


Figure 4.3: DDPG-agent of CoRA based in [4]

4.3.1 Deep Deterministic Policy Gradient

DDPG contains four DNN, the first two are the Q-Network represented with Q and the deterministic policy network represented with μ. The next two are the target Q, represented with Q' and the target policy network represented with μ'. The last two networks are a mechanism original for DDPG to stabilize the learning of the DDPG's agent.

Experience Replay

A fundamental factor in DRL algorithms is experience replay. The experience replay in DDPG updates the neural networks during training to prevent the correlation in the data used to update the neural networks. And seek independent distribution among the dataset, so the algorithm achieves a well-developed learning process. The experience replay overcomes those limitations by creating a finite cache to save data regarding each learning episode's state, action, reward, and next-state tuples. Next, a random sample of these stored tuples nourishes the training; usually, the experience replay evicts the oldest episodes of information, giving way to save the most recent ones. Is proven that the use of experience replay improves efficiency and stability by storing a finite number of the most recent tuples of training [97].

Actor And Critic Network Updates

Starting from reliable data the algorithm can introduce a method to grow the learning process. In this case, the method is the Actor-Critic method. The actor network is in charge of nailing which action should be taken. At the same time, the critic network reports how good the action was by computing the value function and then delivering the information to adjust future decision-making.

Thus the Q networks represent the critic networks and the deterministic policy (μ) represent the actor's networks.

We use the Bellman equation, similar to Q-learning to update the critic ANN to obtain the optimal action value.

$$y_i = r_i + \gamma Q'(S_{i+1}, \mu'(S_{i+1}|\theta^{\mu'})|\theta^{Q'}) \tag{4.1}$$

In the equations 4.1, $\theta^{Q'}$ is the weight of neurons in the target critic network, $\theta^{\mu'}$ is the weight of neurons in target actor network.

The target actor and the target critic networks calculate the next-state Q values.

Then the target updated values are used to minimize the mean squared loss between the updated Q value and the original Q value:

$$L = \frac{1}{N} \sum (y_i - Q(s_i, a_i|\theta^Q))^2 \tag{4.2}$$

The actor network's policy function aims to minimize the expected return.

$$J(\theta) = \mathbb{E}[-Q(s, a)|_{s=s_t, a_t=\mu(s_t)}] \tag{4.3}$$

Then we derivate the function to the parameters of the policy. Since DDPG is an off-policy driven algorithm, the summation mean of the gradients from the experience batch is added as follows.

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum \nabla_a (-Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}) \tag{4.4}$$

Optimizer

Another vital part of the mechanism is the **optimizer**. The optimizer is an algorithm used by different ML techniques to reduce loss or cost functions by updating the weights of the DNN [98]. The optimizers have a parameter called the learning rate that determines how much the weights of the DNN are updated, improving or worsening the agent's performance. There are different types of optimizers, but Adam is the most used in DL [99].

Target Network Updates

The value returned by the target network act as an objective value for the Q-function to minimize the mean squared loss. For the sake of learning stability, the parameters of the target network cannot be the same as those used to train the main networks. That is why we update target networks with a time delay once per major network update. The DDPG algorithms update the target network using the Polyak average.

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \tag{4.5}$$

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \tag{4.6}$$

Where the parameter τ called target network updater is usually near 1 $\tau << 1$.

Exploration

For discrete action spaces, the exploration is achieved with techniques such as ϵ-greedy that allows the agent to explore potentially promising actions while exploiting their current knowledge [100]. In continuous action spaces like the DDPG's case, introducing noise to the action guarantees exploration by bringing in randomness. To add noise to the action, the original DDPG paper [96] use and recommends the Ornstein-Uhlenbeck Process. This process generates random values for a mean and standard deviation specified, that furthermore depends on the previously generated random value.

4.3.2 State Space

In the DRL agent, the state space corresponds to all the pairs of nodes that can establish communication in the network. For example, a state s_i can be node x as the source and node y as the destination. Another state s_j could be the inverse of state s_i, with node y as the source and node x as the destination. Therefore, the number of possible states is the permutation of the network's total number of existing nodes.

$$|S| = |P(N,k)| = \frac{N!}{(N-k)!} \tag{4.7}$$

Where N is the total number of nodes in the network, and k is equal to 2, given that one source node and one destination node.

4.3.3 Action Space

The action space is a set of valid choices to continue the interaction and advancement inside the environment, that is, the state space. The agent has a set of actions A_t for any given state. Thus, the Action States is a list of the possible k paths per state S_t. The *Network Information Repository* saves the possible k paths for state S_t. DDPG is a DRL technique for continuous actions; therefore, we discretize the action generated by the agent to obtain one path out of k paths. This approximation is valid since [33] uses the same action space for a DDPG and DQN agent. In addition, in [101], they discretize the action delivered by the DDPG agent to make a decision based on their problem, obtaining good results.

4.3.4 Reward

The reward function incentivizes the algorithm to converge to the optimal policy in the long term. Consequently, the value of the reward function represents the cost of a potential path in the Action Space for every state. The equation 4.8 defines the Reward Function inversely proportional to the mean available bandwidth in the path bwa_{path} and directly proportional to the path delay d_{path}, path packet loss ratio l_{path}, and queue device's occupation qo_{path}.

$$R = \beta_1 * \frac{1}{bwa_{path}} + \beta_2 * d_{path} + \beta_3 * l_{path} + \beta_4 * qo_{path} \tag{4.8}$$

The values β_1, β_2, β_3, and $\beta_4 \in [0,1]$ can be modified to furnish weight to a specific metric into the Reward Function.

In the Reward Function, the metrics must be all normalized using equation 4.9 known as the Min-Max, advised in [102] to improve accuracy. Since the metrics in the algorithm are in different units, so one metric is not prevalent over the others, delivering an exact outcome.

$$\hat{x_i} = a + \frac{(x_i - min(X)) * (b - a)}{max(X) - min(X)} \tag{4.9}$$

The Min-Max method involves scaling the values of the metrics to an arbitrary interval; [a,b]. Where $\hat{x_i}$ is the value to normalized, and X is a set of values, the equation 4.10 is the normalized version of 4.8

$$\hat{R} = \beta_1 * \frac{1}{bw\hat{a}_{path}} + \beta_2 * \hat{d_{path}} + \beta_3 * \hat{l_{path}} + \beta_4 * q\hat{o_{path}} \tag{4.10}$$

Link Metrics The Management Plane is in charge of calculating both link and device state metrics. This plane computes link throughput and loss using the number of packets that pass through the links connected to the switch port since it samples the number of bytes transmitted or received.

Comparing the retrieved values at two different instants is possible to discover the instantaneous throughput. When the controller sends to the Data Plane a *port-stats* message at time t_1, the number of bytes received b_{t_1} is replied. A second request message is sent, and the reply b_{t_2} contains the number of bytes received at t_2, the duration of the interval that separates times t_1 and t_2 is the period p.

$$bwu_{link} = [\frac{(b_{t_2} - b_{t_1})}{p}] \tag{4.11}$$

Then to acquire the available link bandwidth, we make the difference between the link capacity cap_{link} and the instantaneous throughput.

$$bwa_{link} = cap_{link} - bwu_{link} \tag{4.12}$$

The controller sends *port-stats* of the respective ports of the switches belonging to a link. The number of bits sent btx_i by the port is observed in the response, and the number of bits received by the other port of the neighboring switch brx_j is observed. With these data, equation 4.13 is applied to calculate the instantaneous loss ratio.

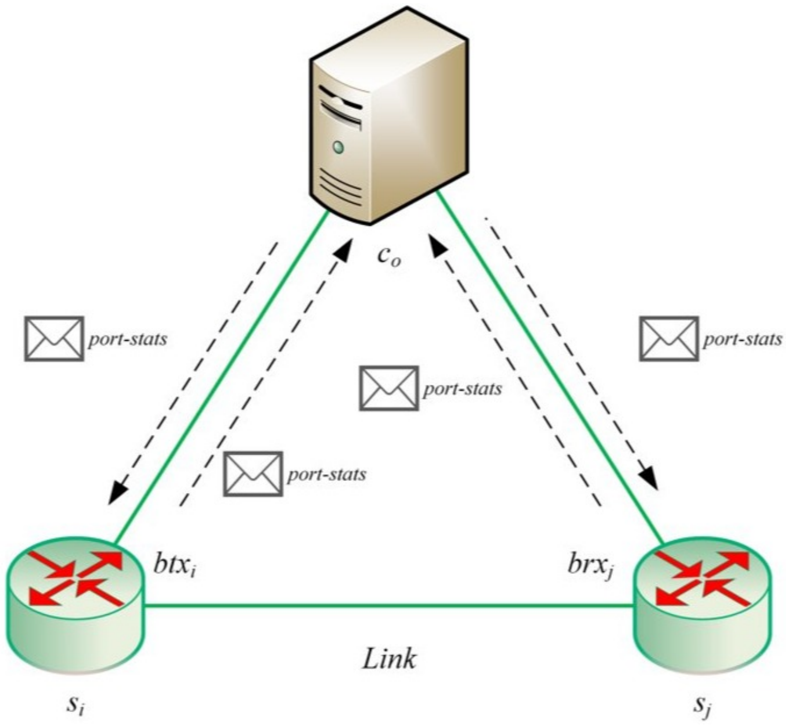


Figure 4.4: Procedure to find packet loss

$$l_{link} = \frac{btx_i - brx_j}{btx_i} \tag{4.13}$$

Following the process described in [103], we compute the instantaneous delay using LLDP and OpenFlow messages. A LLDP message sent by the controller c_0 does the path c_0-s_i-s_j-c_0 with $(s_i,\ s_j)$ being the link that connect the switches s_i and s_j. The time between the transmission and reception of the LLDP message is captured in the message's time stamp, that is, $d_{lldp_{cij}}$. Then, time taken from c_0 to the s_i's port is estimated as half the time that passed between the transmission time and the reception of the OpenFlow *echo-request* and *echo-reply* messages sent by c_0 to s_i, that is $d_{c_0-s_i}$. Similarly, the time elapsed between s_j to c_0, is $d_{c_0-s_j}$, finally the equation 4.14 depicts the instantaneous delay in the link $(s_i,\ s_j)$.

$$d_{s_i-s_j} = d_{lldp_{cij}} - d_{c_0-s_i} - d_{c_0-s_j} \tag{4.14}$$

The Management Plane processes the link metrics to find the path metrics needed for the overall network status. The link metrics (e.i., bwa_{link}, l_{link}, $d_{s_i - s_j}$) are processed to find the path metrics as follows. The Management Plane obtains the lower bandwidth available from all belonging links to the path P to fetch P's available bandwidth then.

$$bwa_{path} = \min_{i \in P}(bwa_{link_i}) \tag{4.15}$$

The sum of the delay from all the links that reside in the path P is equal to the total delay of the path.

$$d_{path} = \sum_{i \in P} d_{link_i} \tag{4.16}$$

Estimating the path loss can be seen as the failure probability in a system of series-coupled components since every link is independent. At the same time, the links are arranged contiguously to construct a path. Thus, the path loss responds to the equation below.

$$l_{path} = 1 - \prod_{i \in P}(1 - l_{link_i}) \tag{4.17}$$

Device Metrics. As show in figure 4.5, the switch ingress queue occupation is the device metric selected to achieve the global network state to acquire an optimal routing policy. The Data Plane has an HTTP server that employs Linux Traffic Control (TC) tool to extract the queue occupation from the devices. TC is a utility that enables configuring the kernel packet scheduler likewise model packet delay, loss, bandwidth, and switch queue with a set packet space [104]. TC requires the *Device Statistics Module* in the Control Plane to send an HTTP type GET request to the Data Plane server. The GET request first identifies each switch network interface, and then, the queue occupation is ready to be consulted. TC finds the number of packets queued on each network interface and returns the GET request response in a JavaScript Object Notation (JSON) format to the *Device Statistics Module* joining

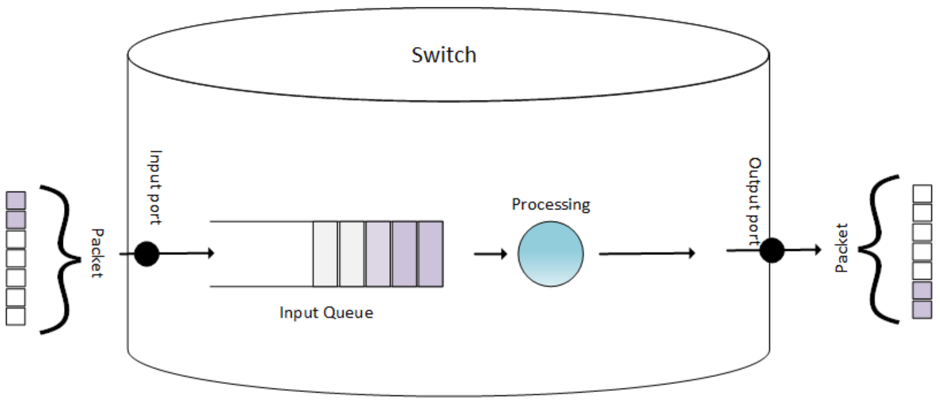


Figure 4.5: Switch ingress queue occupation

the interface's name with the number of packets queued. The interface's name contains the switch and the port number to identify which link the information came through in the process.

To find the complete path queue occupation, the Management Plane makes the sum between the queue of the links integrating the path, as shown beneath;

$$qo_{path} = \sum_{i \in P} qo_{link_i} \tag{4.18}$$

4.4 Cognitive Routing Algorithm

The algorithm 1 describes the process of finding the optimal path between a source-destination node pair sequentially while considering the link metrics; available bandwidth, packet loss, delay, and as a device metric, the switch queue occupation. The algorithm inputs are; learning episode number n which is a sequence of states, actions and rewards, that ends with terminal state in our case the convergence to a optimal policy. Other inputs are the k possible paths list per state , the network path state, the discontinuity factor, the learning rate from the optimizer, the target network update value, number of step by episode and the batch size, that's to say, the number of samples processed before the network update. The algorithm delivers to the *flow installation module* the best route from every node pair for its installation.

To start the algorithm, the Management Plane processes the path metrics to build the environment. The environment is created with the states, and actions with their respective reward. In the second line, the Target ANNs begin with the same weight as the Actor and Critic ANNs, aiming to stabilize the learning process. Then the Replay Memory will record the agent's experiences.

The agent trains by interacting with the environment previously created in a set of n episodes (lines 6 and 18). In line 6, the agent initializes the state with a node pair chosen randomly. The agent iterates m-times each episode (lines 8 to 17). Subsequently, the agents start to take an action regarding the actual state and take a path from the k paths using the actor NN. The environment acquires this action, and the equation 4.10 conveys the reward and next state s_{t+1}. Afterward, the replay memory stores a tuple assembled with the current state s_t, the action a_t taken by the actor NN, the reward r_t, and s_{t+1}. A strategy to train the agent in an off-policy manner is to extract a size N mini-batch of information from past experiences (line 12).

We calculate the expected value with equation 4.1 (line 13) using the mini-batch of information and the critic target ANN. This computation reduces the standard deviation in the learning process and achieves faster, more efficient, and more stable convergence. Then, we calculate the mean squared loss 4.2 to update the critic ANN (line 14).

Conveying the policy gradient (line 15) using the chain rule with the critic ANN, we update the actor ANN. The actor ANN computes the actions regarding the mini-batch states, then, using the critic ANN we find the expected values generated with the tuple from the mini-batch. To minimize the reward, the loss function to find the policy gradient from the actor would be the mean of the expected values. Using the soft update, the agent state, the actor, and the critic target ANNs are actualized (line 16).

At the end of the learning process, we use the actor weights to find a path that generates the smaller reward from each node pair's k path lists (line 20). Those paths are stored and sent to the Control Plane so the *flow installation module* configures the routes in the Data Plane. Then, we verify if the learning time was longer than the monitorization time of the network; if that's the case, we wait for the control plane to obtain the new path states. Finally, we use the new path states to reconstruct the environment.

Algorithm 1: CoRA algorithm

```
Input    : Number of learning episodes: $n$
           Number of steps by episode: $m$
           List of "k" paths per state: $k_{paths}$
           Network path-state
           Discount factor: $\gamma$
           Learning rate
           Target network updater: $\tau$
           Batch size: N
Output   : Set with the best routing path for all pairs of nodes in the network
1  Build Environment Network with Network path-state
2  Initialize critic and actor targets NNs with weights $\theta^{Q'} \leftarrow \theta^{Q}, \theta^{\mu'} \leftarrow \theta^{\mu}$
3  Initialize Replay Memory
4  while $true$ do
5      for $episode$ to $n$ do
6          Initial state $S_t$
7          for $step$ to $m$ do
8              Select action $a_t$
9              Execute action on environment
10             Get reward $r_t$ and next state $s_{t+1}$
11             Store tuple ($s_t$, $a_t$, $r_t$, $s_{t+1}$) into Replay Memory
12             Sample a random mini-batch from Replay Memory
13             Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
14             Update critic $L = \frac{1}{N}\sum(y_i - Q(s_i, a_i|\theta^{Q}))^2$
15             Update actor policy to minimize reward
                 $\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum \nabla_a(-Q(s,a|\theta^{Q})|_{s=s_i,a=\mu(s_i)} \nabla_{\theta^{\mu}}\mu(s|\theta^{\mu})|_{s_i})$
16             Update the target networks
                   $$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$$
                   $$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$$
17             Update state $s_t = s_{t+1}$
18         end
19     end
20     Use final $\theta^{\mu}$ to retrieve the path from $k_{paths}$ that corresponds to the action with the
         lowest reward for each state
21     Store the set of paths for all pair of nodes in the controller to install on data plane;
22     if $timelearning < timemonitoring$ then
23         wait $timemonitoring - timelearning$
24     end
25     Retrieve new network path-state
26     Rebuild Environment network with new network path-state
27 end
```

Chapter 5

Evaluation and Analysis

5.1 Test Environment

The agent works on the 2004s GÉANT network topology [105]. This topology comprises 23 nodes with 37 links. The capacity of the links is: 19 links of 10Gbps, 14 links of 2.5Gbps, and 4 links with 155 Mbps. Emulating the actual link capacity is not possible due to computational resource limitations, mainly CPU restrictions. Therefore the capacity of the links was scaled in a 100 ratio. The links with 10 Gbps, 2.5 Gbps, and 155 Mbps of capacity tuned into 100 Mbps, 25 Mbps, and 1.55 Mbps, respectively. Since the links have diminished capacity, the traffic must shrink in the same proportion. The network topology was be emulated with Mininet version 2.2.2 [106], and the Switches with Open Virtual switch 2.13.8.

To model the links, we used TC tool from Linux, which allowed us to specify the available bandwidth and the input queue size in packet number. In commercial switches the queue size in depends on the model, the manufacturer, and the traffic. For our case we defined an input queue size of 100 packets since it produced a reasonable balance between delay and loss [107].

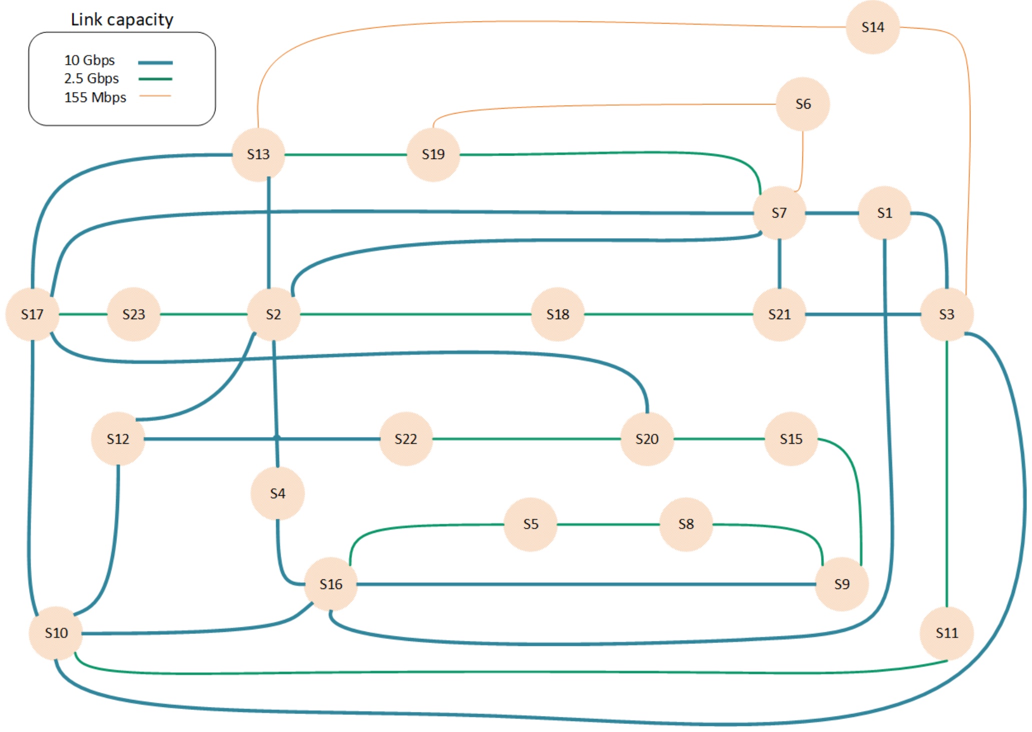


Figure 5.1: GÉANT topology [5]

5.2 Prototype

Figure 5.2 portrays CoRA's prototype where the Data Plane that comprises the GÉANT network topology was emulated with Mininet. Furthermore in the Control Planewe rendered the controller, *Link State Statistics*, *Device State Statistics*, *Topology Discovery*, and *Flow installation modules* additionally with RYU API, and to gather device and link status data Ryu's tools allow the process. The Management, Knowledge Planes, and the DDPG agent were deployed with Python 3.7, Keras 2.9.0, and TensorFlow 2.9.1 to carry out the ML workforce. For the data processing, we used libraries specialized in data manipulation, NumPy 1.23.1 and Pandas. Every plane was executed on Ubuntu Desktop 20.04.5 LTS with a Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz using 8 cores and 16 GB of RAM.

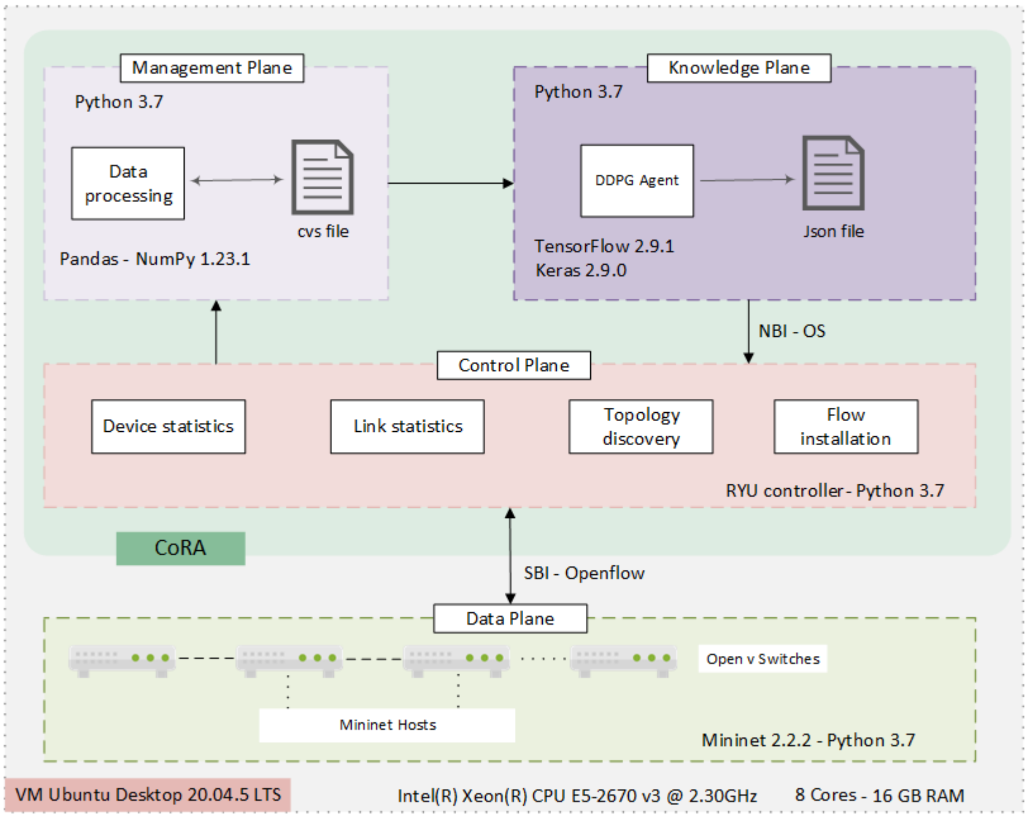


Figure 5.2: CoRA Prototype

5.3 Performance Metrics

As shown in the literature in section 3.1, the most common metrics for evaluating a routing algorithm's performance are delay, packet loss, or throughput. We consider essential to include each of those metrics for performance evaluation, including the queue occupation as a fundamental part of the network state. The modules in charge of the network metrics are *link state statistics* and *device state statistics* that sample the network state every 10s. Likewise, the *data processing module* processes the metrics to the path level for the agent to receive them. We confront the significance of the link and device metrics with DRSIR, which uses the link state metrics only to generate the best policy.

5.4 Traffic Generation

To generate the traffic in the mininet emulation, we used an informatic network tool, *iperf2*. The *iperf2* scripts produced User Datagram Protocol (UDP) traffic according to a traffic matrix. The traffic matrix corresponds to a set of sixteen public intra-domain traffic matrices [108] of the GÉANT Paneuropean topology. We delimited the traffic matrix of the GÉANT topology to accommodate the level of congestion to verify the device metrics' essential role in the network's global view. Thus, we selected the heavier traffic day of the year sample and processed it, diminishing atypical values.

5.5 Hyperparameters Setup

There are different hyperparameters in the DDPG technique, which can modify the behavior of the reward, improving or depreciating the agent's performance. The hyperparameters that we are going to evaluate are:

- Number of hidden layers.
- Number of neurons per hidden layer.
- Discount factor (γ).
- Learning rate.
- Target network updater (τ).
- Batch size.

For the step number to finish an episode, DRSIR initially implemented 30 steps. Still, 30 made our algorithm converge more slowly, and with a couple of trials, we arrived at 45, which we proved to be the most accurate value. In the context of

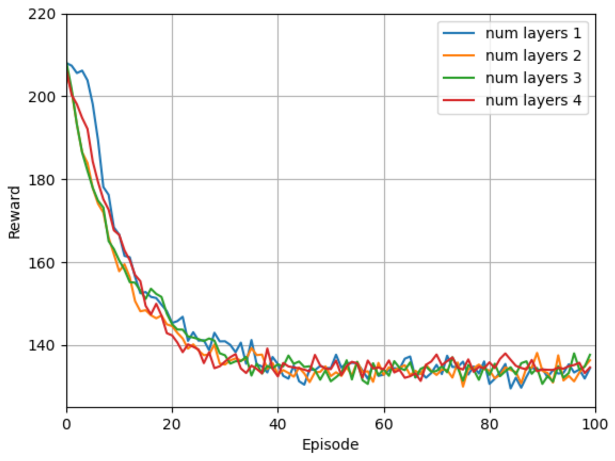


Figure 5.3: Varying number of hidden layers.

our state spaces, we need to monitor the tendency of the reward at the end of the episode, which makes sense when using 45 allows us to know the trend.

In figure 5.3 we vary the number of hidden layers in the neural network of the actor and the critic. We observe that modifying the number of hidden layers does not noticeably affect the behavior of the reward. However, it significantly affects the agent's convergence time, which is the 50th episode. Therefore, we will use only one hidden layer in the neural networks in actor and critic.

In the same way that was varying the number of hidden layers, varying the number of neurons in the hidden layers of the networks does not significantly affect the reward behavior of the agent, as can be seen in figure 5.4. Varying the number of neurons exclusively influences the agent's convergence time. The more neurons, the longer the convergence time will be. Consequently, we took the lower number of neurons per layer, the reason why if we increased the neuron number, the convergence time increases, and the reward makes no better.

In Figure 5.5, we vary the discontinuity factor (γ) with 0.1, 0.5 y 0.9. The behavior of the reward changes slightly by varying this hyperparameter. However, the γ value with a lower average reward is 0.1. The above may be due to the DRL agent design; the agent delivers the path to each pair of nodes, so the agent considers the

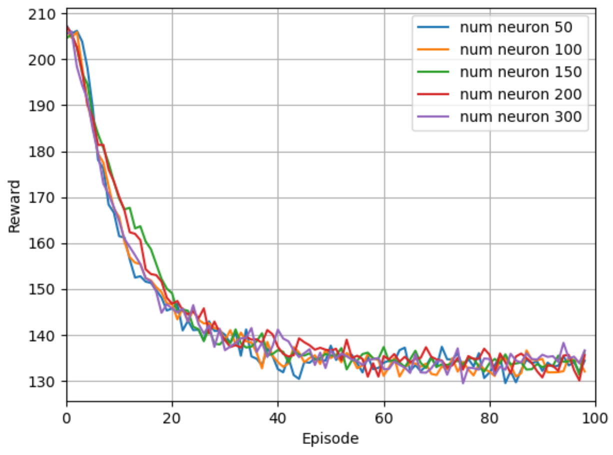


Figure 5.4: Varying number of neurons per layer.

immediate reward.

In figure 5.6, we vary the learning rate in the optimizer. We start to vary from the value 0.001 to 0.05 to observe the impact on the reward. A high value of 0.05 in the learning rate causes the agent not to be able to converge; this is because updating the weights in the neural networks can be abrupt, causing the agent not to find the optimal weight for convergence. On the contrary, a low value of 0.001 in the learning rate means that the agent needs more training episodes to converge because the update weights of the neural networks are small. Whereby the update weights of the neural networks are small, it would need more steps to converge. The optimal value of the learning rate for the agent is 0.01.

Modifying the batch size does not imply a significant change in the behavior of the reward, except for a size of 10, as can be seen in the figure 5.7. The size of 30 generates a slightly lower reward than the other batch sizes. Also, if this value is increased, the convergence time increases because the batch size is the number of times the agent has to relearn from that previous experience to avoid learning instability.

In Figure 5.8, we analyze the effect of the target network updater (τ) on the behavior of the reward. τ value has to be considerably smaller than 1. Hence, we vary the

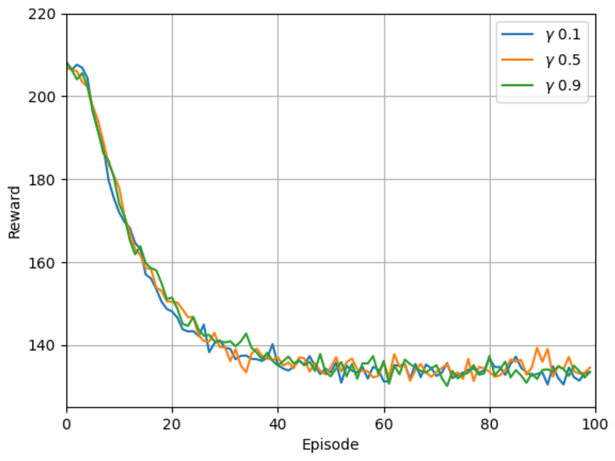


Figure 5.5: Varying discount factor.

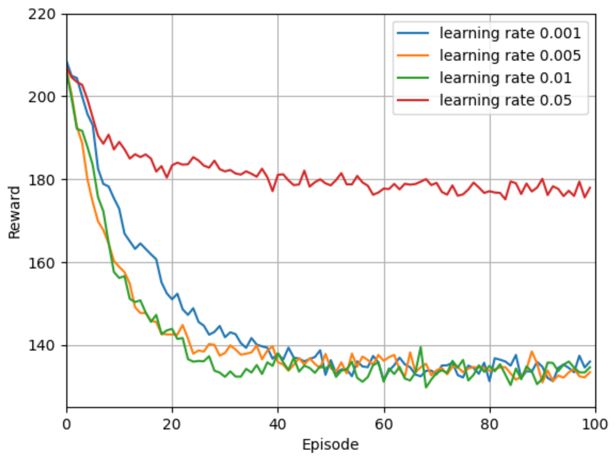


Figure 5.6: Varying learning rate.

τ value to 0.001, 0.005, 0.01, and 0.05. The behavior of the reward related to the variation of τ value is not notorious. Either way, with a value below 0.001, we obtain a minor reward average.

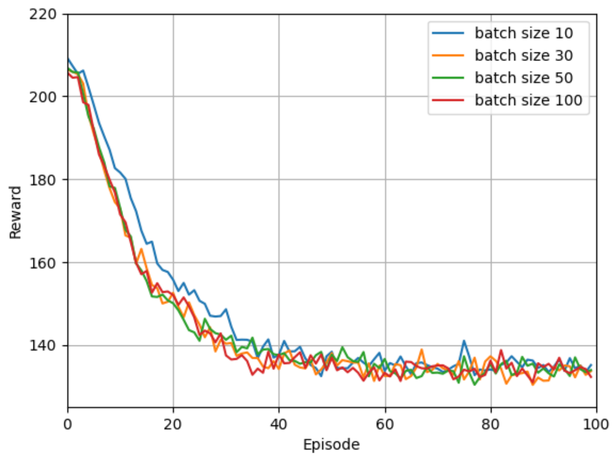


Figure 5.7: Varying batch size.

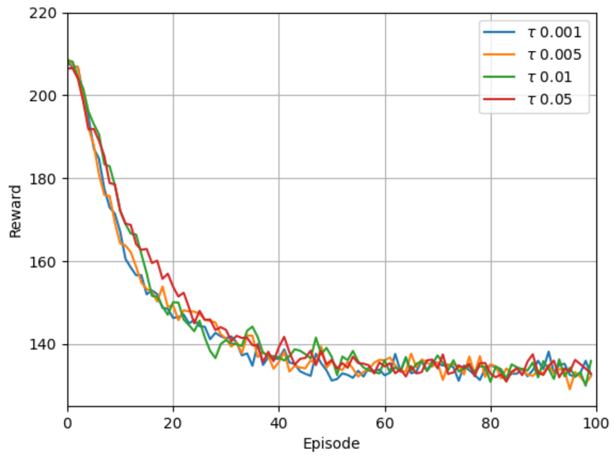


Figure 5.8: Varying target network updater.

5.6 Results And Analysis

In this section, we compare CoRA against DRSIR, the routing algorithm that was our approach's starting point. DRSIR was born to address RL and conventional algorithms shortcomings. DRSIR outperforms the conventional routing algorithms like Dijisktra, and RL algorithms as example RSIR, DRSIR previous proposal. DRSIR is a DQN algorithm that considers path-state metrics to produce proactive, efficient routing that adapts dynamically to network changes. DRSIR is evaluated on the GEANT topology with real and synthetic traffic matrices. We confront the two algorithms performance-wise, considering link metrics instantaneous throughput, average delay, loss rate, and the device metric average queue occupancy of the switches.

Figure 5.9 show the average queue packet occupancy on the Data Plane switches throughout the busiest hours of the day. On the heaviest traffic hours, we observe an increase number of packets dammed in the input queue of the switches due to the increase in packet forwarding on the links. CoRA reduced queue occupancy by 12.7% in regard to DRSIR. The decrease in queue occupancy is due to CoRA explicitly considering the queue occupancy of the switches in the reward function. Therefore, DNNs train to optimize the policy selecting less queue occupy path, that's to say the agent takes into account the queue occupancy. In addition, DDPG as learning technique is sophisticated when learning a policy that finds the best paths for each pair of nodes.

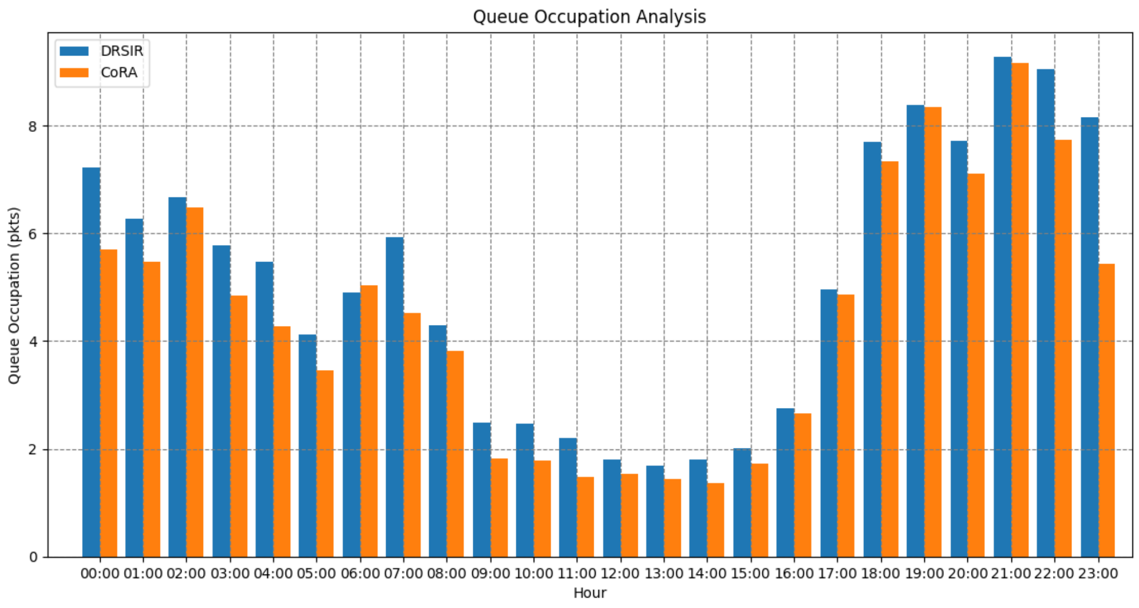


Figure 5.9: Queue occupation Analysis

Figure 5.10 shows the average delay of all links during the most congested hours of the day. As well as the queue occupancy, in the hours with the highest traffic generated by the nodes, the delay of the links begins to increase due to congestion.

The CoRA algorithm obtains a delay improvement of 17% compared to DRSIR. When choosing the path, CoRA considers the number of packets waiting to be processed by the switches. This consideration avoids the links that present congestion in the input queue, reducing the packet queue time and consequently making a more efficient routing to decrease delay.

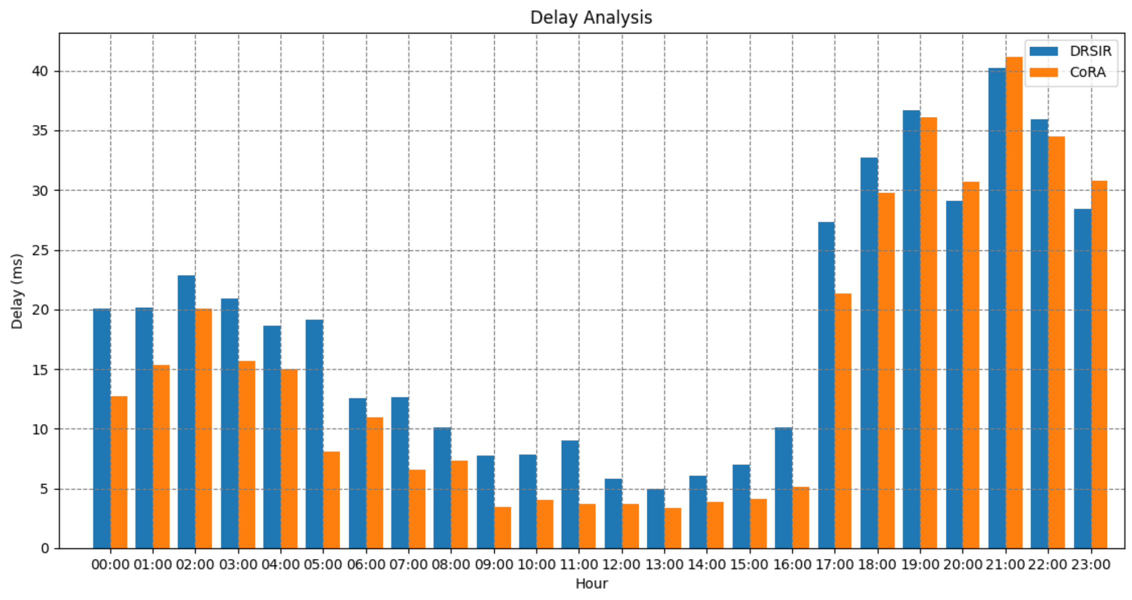


Figure 5.10: Delay Analysis

Figure 5.11 shows the average loss of all the links throughout the hours of the day with the most congestion. When the traffic increases, the congestion in the links becomes present, which induces an occupation in the input queue of the switch; when the number of packets exceeds the size of the queue, the next packet arriving will be discarded.

CoRA obtains 29.44% of loss improvement compared to DRSIR. We attribute this to CoRA's awareness of the queue state of the switches, preventing traffic from being sent over links where the queues have fewer packets waiting, preventing them from filling up to their maximum capacity and start discarding.

Figure 5.12 shows the average instantaneous throughput of all the links during the hours of the day of greatest congestion. The instantaneous throughput indicates how

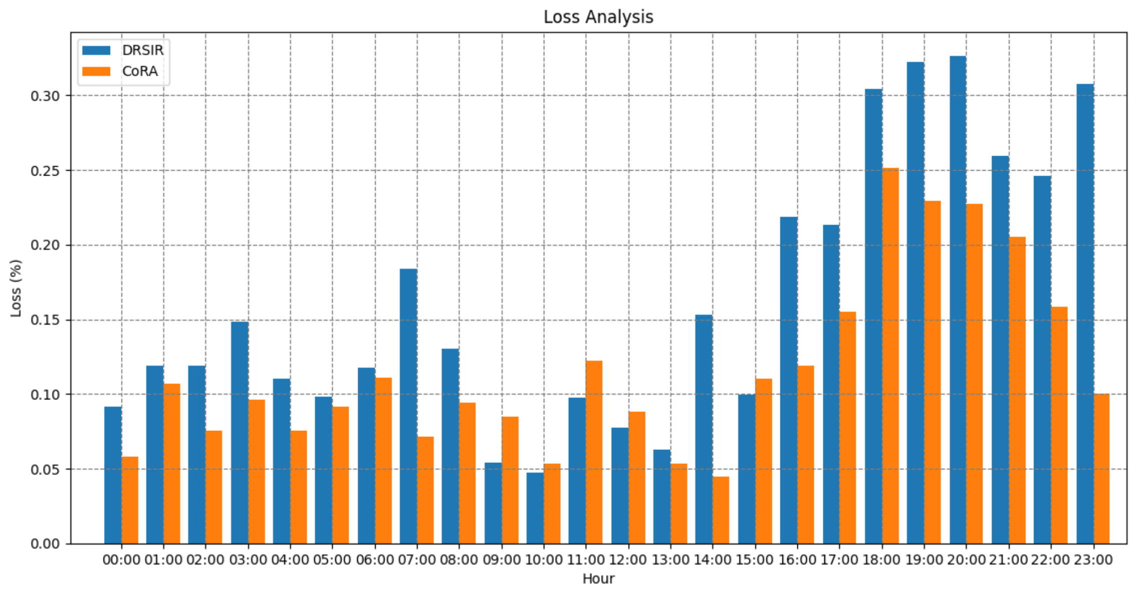


Figure 5.11: Loss Analysis

the routing strategy adequately distributes the traffic generated by all the links. The more traffic, the more needed strategy to avoid overloading the same links.

The CoRA algorithm obtains a 12.5% reduction in instantaneous throughput compared to DRSIR; this means that CoRA distributes the traffic along less congestioned paths, avoiding the overuse of links, therefore improving the performance of the above metrics.

Figure 5.13shows the average stretch produced by the agents throughout the day. The stretch is calculated by the relationship between the size of the path chosen by the agents against the path with the fewest possible hops. CoRA gets slightly longer paths (<2%) than DRSIR. This result means that CoRA distributes the traffic in the paths regarding the metrics considered, not the shorter path. That way, it achieves an improvement in the previous metrics.

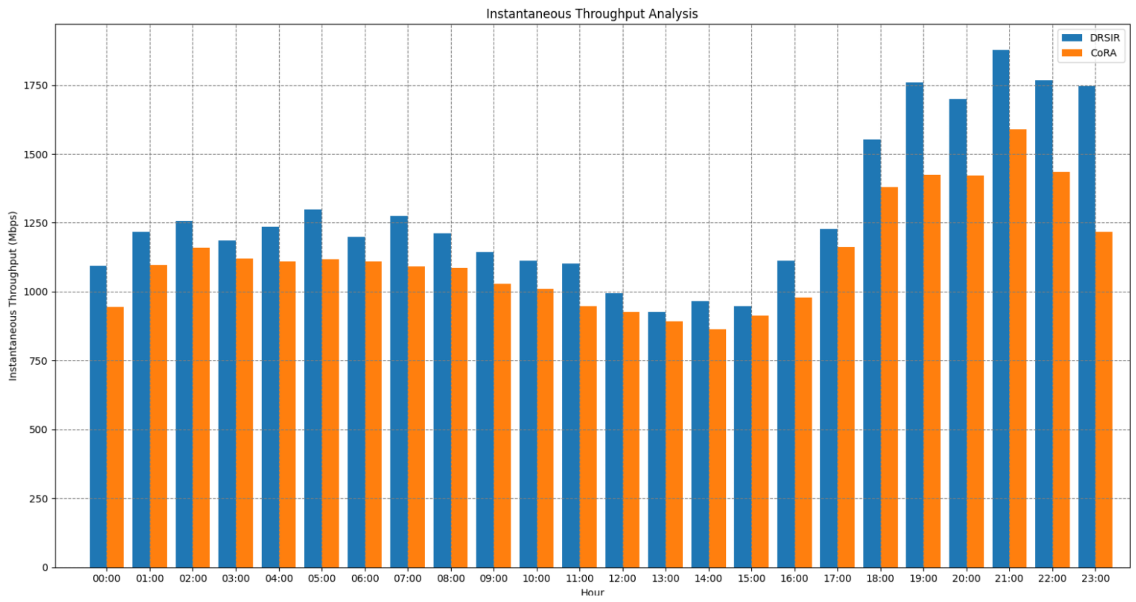


Figure 5.12: Instantaneous throughput Analysis

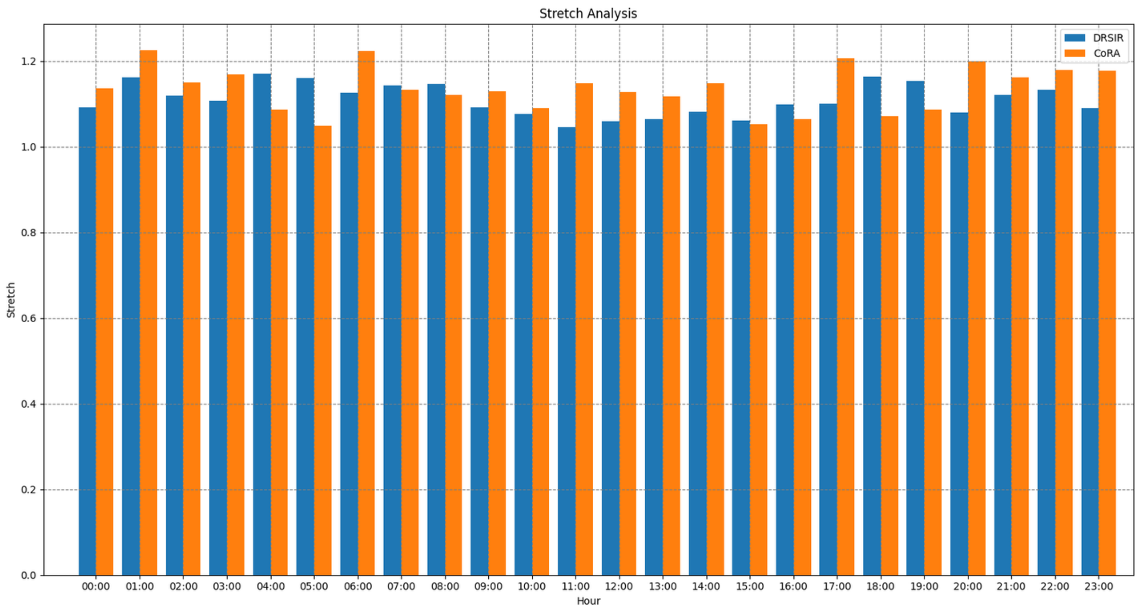


Figure 5.13: Stretch Analysis

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This undergraduate work presents the answer to the question: **How to obtain a cognitive and efficient routing of packets in SDN considering link and device metrics from Control Plane?**

To solve this question, this undergraduate work presented the investigation carried out to verify the hypothesis: **Cognitive routing in SDN can be achieved by analyzing link and device state metrics using DRL techniques.** Based on the hypothesis, we designed the CoRA routing mechanism that implements the DRL technique, DDPG, and the link and device state data. Futhermore, CoRA was implemented on top of the GEANT topology by injecting it with real traffic matrices. CoRa performance was evaluated regarding stretch, loss, delay, instantaneous throughput, and queue occupancy.

CoRA improves the network performance compared to the DRSIR proposal. 12.7% reduced in queue occupancy, the delay is reduced by 17%, 29.4% improved loss, and a 12.5% reduced on the instantaneous throughput. In addition, CoRA gets 2% more stretch than DRSIR. These performance improvements are since the CoRA mechanism fully knows the network state. Making routing decisions, CoRa is aware of the link and device state, namely delay, packet loss, available bandwidth, and input queue switch occupation.

The CoRA mechanism reduces the number of packets waiting in the switch input, and this means a reduction in the link delay. The decrease in packets in the switch input queue reduces the packet loss in the link because the mechanism prevents the number of packets from exceeding the maximum capacity of the switch queue, altogether avoiding queue overflow meaning packet discard. CoRA distributes traffic over less congested links, reducing the average instantaneous throughput. In addition, CoRA obtains better results without sacrificing stretch too much.

To our knowledge, the proposed mechanism is the only one that uses a DRL agent without relying upon classical routing protocols. While monitoring the device's status together with the link status to nourish cognitive routing decisions to reinforce the reward getting optimal performance.

6.2 Future work

- Research other device metrics, such as CPU and RAM consumption, to witness the impact of those device metrics on the network status.
- Implement different DRL techniques.
- Implement different ML techniques for the efficient configuration of the weights of the metrics in the agent's reward before traffic variations.

Bibliography

[1] F. Estrada-Solano, A. Ordonez, L. Z. Granville, and O. M. Caicedo Rendon, "A framework for sdn integrated management based on a cim model and a vertical management plane," *Computer Communications*, 2017.

[2] G. Kim, Y. Kim, and H. Lim, "Deep reinforcement learning-based routing on software-defined networks," *IEEE Access*, 2022.

[3] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A knowledge plane for the internet," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, 2003, pp. 3–10.

[4] G. Kim, Y. Kim, and H. Lim, "Deep reinforcement learning-based routing on software-defined networks," *IEEE Access*, vol. 10, pp. 18 121–18 133, 2022.

[5] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. da Fonseca, "Intelligent routing based on reinforcement learning for software-defined networking," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 870–881, 2020.

[6] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, 2014.

[7] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in software defined networks," *Computer Networks*, vol. 71, pp. 1–30, 2014.

[8] D. Gopi, S. Cheng, and R. Huck, "Comparative analysis of SDN and conventional networks using routing protocols," *IEEE CITS 2017 - 2017 International Conference on Computer, Information and Telecommunication Systems*, pp. 108–112, 2017.

[9] D. Awduche, J. Malcolm, J. Agogbua, M. D. O'Dell, and J. McManus, "Requirements for traffic engineering over mpls," *RFC*, vol. 2702, pp. 1–29, 1999.

[10] Y. Li, X. Li, and O. Yoshie, "Traffic engineering framework with machine learning based meta-layer in software-defined networks," *Proceedings of 2014 4th IEEE International Conference on Network Infrastructure and Digital Content, IEEE IC-NIDC 2014*, pp. 121–125, 2014.

[11] H. Zhang and J. Yan, "Performance of SDN Routing in Comparison with Legacy Routing Protocols," *Proceedings - 2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, CyberC 2015*, pp. 491–494, 2015.

[12] A. Rego, S. Sendra, J. M. Jimenez, and J. Lloret, "Ospf routing protocol performance in software defined networks," in *2017 Fourth International Conference on Software Defined Systems (SDS).* IEEE, 2017, pp. 131–136.

[13] A. A. Khan, M. Hussain, M. Zafrullah, and M. S. Zia, "A convergence time optimization paradigm for ospf based networks through sdn spf protocol computer communications and networks (ccn)/delay tolerant networks," in *Proceedings of the International Conference on Future Networks and Distributed Systems*, 2017.

[14] S. Abdallah, A. Kayssi, I. H. Elhajj, and A. Chehab, "Network convergence in sdn versus ospf networks," in *2018 Fifth International Conference on Software Defined Systems (SDS).* IEEE, 2018.

[15] R. Adrian, A. Dahlan, and K. Anam, "OSPF cost impact analysis on SDN network," in *2017 2nd International conferences on Information Technology, Information Systems and Electrical Engineering (ICITISEE).* IEEE, Nov. 2017. [Online]. Available: https://doi.org/10.1109/icitisee.2017.8285494

[16] E. Amiri, M. R. Hashemi, and K. R. Lejjy, "Policy-based routing in rip-hybrid network with sdn controller," in *4th National Conference on Applied Research in Electrical, Mechanical, Computer and IT Engineering*, 2018, pp. 1–8.

[17] P. Lin, J. Bi, and H. Hu, "Btsdn: Bgp-based transition for the existing networks to sdn," *Wireless Personal Communications*, vol. 86, pp. 1829–1843, 2016.

[18] L. M. Elguea and F. Martinez-Rios, "A new method to optimize BGP routes using SDN and reducing latency," *Procedia Computer Science*, vol. 135, pp. 163–169, 2018. [Online]. Available: https://doi.org/10.1016/j.procs.2018.08.162

[19] V. Kotronis, A. Gämperli, and X. Dimitropoulos, "Routing centralization across domains via SDN: A model and emulation framework for BGP evolution," *Computer Networks*, vol. 92, 2015.

[20] W. Duan, L. Xiao, D. Li, Y. Zhou, R. Liu, L. Ruan, Y. Xia, and M. Zhu, "Ofbgp: A scalable, highly available bgp architecture for sdn," in *2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems*, 2014, pp. 557–562.

[21] G. Xu, Y. Mu, and J. Liu, "Inclusion of Artificial Intelligence in Communication Networks and Services," *ITU Journal: ICT Discoveries, Special Issue*, no. 1, pp. 1–6, 2017.

[22] T. Hendriks, M. Camelo, and S. Latré, "Q 2-routing: A qos-aware q-routing algorithm for wireless ad hoc networks," in *2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 2018, pp. 108–115.

[23] S.-C. Lin, I. F. Akyildiz, P. Wang, and M. Luo, "Qos-aware adaptive routing in multi-layer hierarchical software defined networks: A reinforcement learning approach," in *2016 IEEE International Conference on Services Computing (SCC)*. IEEE, 2016, pp. 25–33.

[24] C. Wang, L. Zhang, Z. Li, and C. Jiang, "SDCoR: Software Defined Cognitive Routing for Internet of Vehicles," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3513–3520, 2018.

[25] T. Mahboob, Y. R. Jung, and M. Y. Chung, "Optimized Routing in Software Defined Networks - A Reinforcement Learning Approach," in *Proceedings of the 13th International Conference on Ubiquitous Information Management and Communication (IMCOM) 2019.* Springer International Publishing, 2019.

[26] S. Kim, J. Son, A. Talukder, and C. S. Hong, "Congestion prevention mechanism based on q-leaning for efficient routing in sdn," in *2016 International Conference on Information Networking (ICOIN)*, 2016, pp. 124–128.

[27] Y.-H. Lu and F.-Y. Leu, "Dynamic routing and bandwidth provision based on reinforcement learning in SDN networks," in *Advanced Information Networking and Applications.* Springer International Publishing, 2020, pp. 1–11.

[28] M. B. Hossain and J. Wei, "Reinforcement learning-driven QoS-aware intelligent routing for software-defined networks," in *2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP).* IEEE, Nov. 2019. [Online]. Available: https://doi.org/10.1109/globalsip45357.2019.8969320

[29] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y. C. Liang, and D. I. Kim, "Applications of Deep Reinforcement Learning in Communications and Networking: A Survey," *IEEE Communications Surveys and Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[30] E. H. Bouzidi, A. Outtagarts, and R. Langar, "Deep reinforcement learning application for network latency management in software defined networks," in *2019 IEEE Global Communications Conference (GLOBECOM).* IEEE, Dec. 2019. [Online]. Available: https://doi.org/10.1109/globecom38437.2019.9013221

[31] Y.-R. Chen, A. Rezapour, W.-G. Tzeng, and S.-C. Tsai, "RL-routing: An SDN routing algorithm based on deep reinforcement learning," *IEEE*

Transactions on Network Science and Engineering, pp. 1–1, 2020. [Online]. Available: https://doi.org/10.1109/tnse.2020.3017751

[32] Q. Fu, E. Sun, K. Meng, M. Li, and Y. Zhang, “Deep q-learning for routing schemes in SDN-based data center networks,” *IEEE Access*, vol. 8, pp. 103 491–103 499, 2020. [Online]. Available: https://doi.org/10.1109/access.2020.2995511

[33] W. xi Liu, “Intelligent routing based on deep reinforcement learning in software-defined data-center networks,” in *2019 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, Jun. 2019. [Online]. Available: https://doi.org/10.1109/iscc47284.2019.8969579

[34] G. Stampa, M. Arias, D. Sánchez-Charles, V. Muntés-Mulero, and A. Cabellos, “A deep-reinforcement learning approach for software-defined networking routing optimization,” *arXiv preprint arXiv:1709.07080*, 2017.

[35] J. Chen, Z. Xiao, H. Xing, P. Dai, S. Luo, and M. A. Iqbal, “Stdpg: A spatio-temporal deterministic policy gradient agent for dynamic routing in sdn,” 2020.

[36] C. Yu, J. Lan, Z. Guo, and Y. Hu, “DROM: Optimizing the routing in software-defined networks with deep reinforcement learning,” *IEEE Access*, vol. 6, pp. 64 533–64 539, 2018. [Online]. Available: https://doi.org/10.1109/access.2018.2877686

[37] P. Sun, Y. Hu, J. Lan, L. Tian, and M. Chen, “TIDE: Time-relevant deep reinforcement learning for routing optimization,” *Future Generation Computer Systems*, vol. 99, pp. 401–409, Oct. 2019. [Online]. Available: https://doi.org/10.1016/j.future.2019.04.014

[38] Q. T. A. Pham, Y. Hadjadj-Aoul, and A. Outtagarts, “Deep Reinforcement Learning based QoS-aware Routing in Knowledge-defined networking,” in *Qshine 2018 - 14th EAI International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, Ho Chi Minh City, Vietnam, Dec. 2018, pp. 1–13. [Online]. Available: https://hal.inria.fr/hal-01933970

[39] Y. Hu, Z. Li, J. Lan, J. Wu, and L. Yao, “EARS: Intelligence-driven experiential network architecture for automatic routing in software-defined networking,” *China Communications*, vol. 17, no. 2, pp. 149–162, Feb. 2020. [Online]. Available: https://doi.org/10.23919/jcc.2020.02.013

[40] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. S. da Fonseca, “Drsir: A deep reinforcement learning approach for routing in software-defined networking,” *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 4807–4820, 2022.

[41] L. Zhang, Y. Lu, D. Zhang, H. Cheng, P. Dong *et al.*, “Dsoqr: Deep reinforcement learning for online qos routing in sdn-based networks,” *Security and Communication Networks*, vol. 2022, 2022.

[42] C. Zhao, M. Ye, X. Xue, J. Lv, Q. Jiang, and Y. Wang, “Drl-m4mr: An intelligent multicast routing approach based on dqn deep reinforcement learning in sdn,” *Physical Communication*, vol. 55, p. 101919, 2022.

[43] J. F. K. K. Ross, *Computer Networking: A Top-Down Approach*, 7th ed. Pearson, 2016.

[44] J. Macfarlane, *Network routing basics: Understanding IP routing in Cisco systems.* John Wiley & Sons, 2007.

[45] D. Medhi, “Network routing: An overview,” in *Network Routing Algorithms, Protocols, and Architectures.* ELSEVIER Inc, 2007, ch. 1.2, pp. 5–7.

[46] M. Jayakumar, N. R. S. Rekha, and B. Bharathi, “A comparative study on RIP and OSPF protocols,” in *2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS).* IEEE, Mar. 2015. [Online]. Available: https://doi.org/10.1109/iciiecs.2015.7193275

[47] L. Lan, L. Li, and C. Jianya, “A multipath routing algorithm based on ospf routing protocol,” in *2012 Eighth International Conference on Semantics, Knowledge and Grids*, 2012, pp. 269–272.

[48] A. A. Noman and A. Chowdhury, “Performance analysis of EIGRP and OSPF for different applications using OPNET,” *Australasian Journal of Computer Science*, vol. 1, no. 1, pp. 1–8, 2014.

[49] G. S. Kalyan and D. V. V. Prasad, “Optimal selection of dynamic routing protocol with real time case studies,” in *2012 International Conference on Recent Advances in Computing and Software Systems*. IEEE, 2012, pp. 219–223.

[50] C. L. Hedrick, “RFC1058: Routing information protocol,” 1988.

[51] T. M. Thomas, *OSPF network design solutions.* Cisco Press, 1998.

[52] Y. Rekhter, T. Li, and S. Hares, “RFC 4271: A border gateway protocol 4 (bgp-4),” 2006.

[53] P. Rakheja, P. kaur, A. gupta, and A. Sharma, “Performance analysis of rip, ospf, igrp and eigrp routing protocols in a network,” *International Journal of Computer Applications*, vol. 48, pp. 6–11, 06 2012.

[54] B. Mao, Z. M. Fadlullah, F. Tang, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, “A tensor based deep learning technique for intelligent packet routing,” in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, 2017, pp. 1–6.

[55] R. Amin, E. Rojas, A. Aqdus, S. Ramzan, D. Casillas-Perez, and J. M. Arco, “A survey on machine learning techniques for routing optimization in sdn,” *IEEE Access*, 2021.

[56] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, “A machine learning approach to routing,” *arXiv preprint arXiv:1708.03074*, 2017.

[57] F. Hu, Q. Hao, and K. Bao, “A survey on software-defined network and openflow: From concept to implementation,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[58] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, “Are we ready for SDN? Implementation

challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[59] ONF, "Sdn architecture v1.1, technical reference tr-504," Nov. 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN-ARCH-Overview-1.1-11112014.02.pdf

[60] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. S. D. Meyer, and O. Koufopavlou, "Software-defined networking (sdn): Layers and architecture terminology, informational rfc 7426, ietf," Ene. 2015. [Online]. Available: https://tools.ietf.org/html/rfc7426

[61] ITU, "Framework of software-defined networking, recommendation y.3300, international telecommuncation union," Jun. 2014. [Online]. Available: https://www.itu.int/rec/T-REC-Y.3300

[62] K. Benzekki, A. El Fergougui, and A. El Belrhiti El Alaoui, "Software-defined networking (sdn): A survey," *Security and Communication Networks*, vol. 9, 02 2017.

[63] O. S. Specification, "Version 1.5. 0," *Open Networking Foundation*, 2014.

[64] T. Liu, "Implementing open flow switch using fpga based platform," Master's thesis, Institutt for telematikk, 2014.

[65] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *ACM SIGCOMM computer communication review*, vol. 38, no. 3, pp. 105–110, 2008.

[66] D. Erickson, "The beacon openflow controller," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 13–18.

[67] Z. Cai, A. L. Cox, and T. Ng, "Maestro: A system for scalable openflow control," Tech. Rep., 2010.

[68] The Linux Foundation. Opendaylight. [Online]. Available: https://www.opendaylight.org/

[69] R. Wallner and R. Cannistra, “An sdn approach: quality of service using big switch’s floodlight open-source controller,” in *Proceedings of the Asia-Pacific Advanced Network*, vol. 35, no. 14-19, 2013, pp. 10–7125.

[70] Getting started — ryu 4.34 documentation. [Online]. Available: https://ryu.readthedocs.io/en/latest/getting_started.html#what-s-ryu

[71] J. A. Wickboldt, W. P. De Jesus, P. H. Isolani, C. B. Both, J. Rochol, and L. Z. Granville, “Software-defined networking: management requirements and challenges,” *IEEE Communications Magazine*, vol. 53, no. 1, pp. 278–285, 2015.

[72] M. Jarschel, T. Zinner, T. Hoßfeld, P. Tran-Gia, and W. Kellerer, “Interfaces, attributes, and use cases: A compass for sdn,” *IEEE Communications Magazine*, vol. 52, no. 6, pp. 210–217, 2014.

[73] M. B. Al-Somaidai and E. B. Yahya, “Survey of software components to emulate openflow protocol as an sdn implementation,” *American Journal of Software Engineering and Applications*, vol. 3, no. 6, pp. 74–82, 2014.

[74] B. Mao, Z. M. Fadlullah, F. Tang, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, “Routing or computing? the paradigm shift towards intelligent computer network packet transmission based on deep learning,” *IEEE Transactions on Computers*, 2017.

[75] R. S. Sutton and A. G. Barto, *Reinforcement Learning.* The MIT Press, 2014, ch. 1.1, pp. 2–5.

[76] Y. Zhao, Y. Li, X. Zhang, G. Geng, W. Zhang, and Y. Sun, “A Survey of Networking Applications Applying the Software Defined Networking Concept Based on Machine Learning,” *IEEE Access*, vol. 7, pp. 95 397–95 417, 2019.

[77] G. Tesauro, “Reinforcement learning in autonomic computing: A manifesto and case studies,” *IEEE Internet Computing*, vol. 11, no. 1, pp. 22–30, 2007.

[78] Z. Mammeri, “Reinforcement learning based routing in networks: Review and classification of approaches,” *IEEE Access*, vol. 7, pp. 55 916–55 950, 2019.

[79] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, “A comprehensive survey on machine learning for networking: evolution, applications and research opportunities,” *Journal of Internet Services and Applications*, vol. 9, no. 1, Jun. 2018. [Online]. Available: https://doi.org/10.1186/s13174-018-0087-2

[80] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, “State-of-the-art deep learning: Evolving machine intelligence toward tomorrow’s intelligent network traffic control systems,” *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2432–2455, 2017.

[81] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, “Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning,” in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018.

[82] O. Abiodun, A. Jantan, O. Omolara, K. Dada, N. Mohamed, and H. Arshad, “State-of-the-art in artificial neural network applications: A survey,” *Heliyon*, vol. 4, p. e00938, 11 2018.

[83] C. Aggarwal, “An introduction to neural networks,” in *Neural Networks and Deep Learning*. Springer International Publishing AG, 2018, ch. 1.1, pp. 1–3.

[84] A. Gulli, “Introduction to neural networks,” in *Deep Learning with TensorFlow 2 and Keras*, 2nd ed. Packt Publishing, 2019, ch. 1.4, pp. 5–8.

[85] H. Dong, Z. Ding, and S. Zhang, *Deep Q-Networks*. Springer, 2020, ch. 4, pp. 135–161.

[86] S. Mousavi, M. Schukat, and E. Howley, “Deep reinforcement learning: An overview,” *Lecture Notes in Networks and Systems*, pp. 426–440, 06 2018.

[87] B. Brown, “Learning to pick the best policy: Policy gradient methods,” in *Deep Reinforcement Learning in Action*. Manning Publications Co, 2020, ch. 4, pp. 90–111.

[88] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[89] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16. AAAI Press, 2016, p. 2094–2100.

[90] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, p. 1995–2003.

[91] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarcón, M. Solé, V. Muntés-Mulero, D. Meyer, S. Barkai, M. J. Hibbett, G. Estrada, K. Ma'ruf, F. Coras, V. Ermagan, H. Latapie, C. Cassar, J. Evans, F. Maino, J. Walrand, and A. Cabellos, "Knowledge-defined networking," *Computer Communication Review*, vol. 47, pp. 1–10, 2017.

[92] S. Ashtari, I. Zhou, M. Abolhasan, N. Shariati, J. Lipman, and W. Ni, "Knowledge-defined networking: Applications, challenges and future work," *Array*, vol. 14, 2022.

[93] S. Tomovic, N. Prasad, and I. Radusinovic, "Sdn control framework for qos provisioning," in *2014 22nd Telecommunications Forum Telfor (TELFOR)*. IEEE, 2014.

[94] J. Park, J. Hwang, and K. Yeom, "NSAF: An Approach for Ensuring Application-Aware Routing Based on Network QoS of Applications in SDN," *Mobile Information Systems*, vol. 2019, 2019.

[95] J.-R. Jiang, H.-W. Huang, J.-H. Liao, and S.-Y. Chen, "Extending dijkstra's shortest path algorithm for software defined networking," in *The 16th Asia-Pacific Network Operations and Management Symposium*. IEEE, 2014.

[96] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 09 2015.

[97] W. Fedus, P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland, and W. Dabney, "Revisiting fundamentals of experience replay," in *International Conference on Machine Learning.* PMLR, 2020, pp. 3061–3071.

[98] H. M. El Misilmani, T. Naous, and S. K. Al Khatib, "A review on the design and optimization of antennas using machine learning algorithms and techniques," *International Journal of RF and Microwave Computer-Aided Engineering*, vol. 30, no. 10, p. e22356, 2020.

[99] E. Okewu, P. Adewole, and O. Sennaike, "Experimental comparison of stochastic optimizers in deep learning," in *Computational Science and Its Applications–ICCSA 2019: 19th International Conference, Saint Petersburg, Russia, July 1–4, 2019, Proceedings, Part V 19.* Springer, 2019, pp. 704–715.

[100] A. D. Tijsma, M. M. Drugan, and M. A. Wiering, "Comparing exploration strategies for q-learning in random stochastic mazes," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2016, pp. 1–8.

[101] R. Zhong, Y. Liu, X. Mu, Y. Chen, X. Wang, and L. Hanzo, "Hybrid reinforcement learning for star-riss: A coupled phase-shift model based beamformer," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 9, pp. 2556–2569, 2022.

[102] L. Al Shalabi and Z. Shaaban, "Normalization as a preprocessing engine for data mining and the approach of preference matrix," in *2006 International conference on dependability of computer systems.* IEEE, 2006, pp. 207–214.

[103] L. Liao and V. C. M. Leung, "Lldp based link latency monitoring in software defined networks," in *2016 12th International Conference on Network and Service Management (CNSM)*, 2016.

[104] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. Schroeder, J. Spaans, and P. Larroy, "Linux advanced routing & traffic control," in *Ottawa Linux Symposium*, vol. 213. sn, 2002.

[105] P. T. Kirstein, "European international academic networking: A 20 year perspective." in *TERENA Networking Conference*, 2004.

[106] K. Kaur, J. Singh, and N. S. Ghumman, "Mininet as software defined networking testing platform," in *International conference on communication, computing & systems (ICCCS)*, 2014, pp. 139–42.

[107] Y. Lu and S. Zhu, "Sdn-based tcp congestion control in data center networks," in *2015 IEEE 34th international performance computing and communications conference (IPCCC)*. IEEE, 2015, pp. 1–7.

[108] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, p. 83–86, jan 2006. [Online]. Available: https://doi.org/10.1145/1111322.1111341

Cognitive routing of flows in software-defined networks from the control plane

ANNEXES

Undergraduate degree

Sofia Rubin Castillo
Brayam David Otero Pomeo

Advisor: PhD. Oscar Mauricio Caicedo Rendón
Co-Advisor: PhD.Cristhian Nicolás Figueroa Martínez

Department of Telematics
Faculty of Electronic and Telecommunications Engineering
Universidad del Cauca
Popayán, March 2023

Annexes A

Algorithm

Annex A presents the redirect link to the repository on GitHub, where the respective CoRA source codes are located.

`https://github.com/BrayamOtero/CoRA.git`

Annexes B

Publicaciones

Annex B presents the scientific paper written during the undergraduate work development.

- **Brayam David Otero Pomeo**, **Sofia Rubin Castillo**,Oscar Mauricio Caicedo Rendón, Cristhian Nicolás Figueroa Martínez. **Cognitive Routing Of Flows In Software-Defined Networks From The Control Plane**. Elsevier - Computer Networks.
  - Status: Writtend and ready to be sent.
  - Quartile: Q1.
  - Impact Factor: 5.493
  - H-index: 169 Scimagp

CoRA: A Cognitive Routing Algorithm for Routing in SDN on the Control Plane

Brayam David Otero Pomeo, Sofía Rubin-Castillo, Cristian Nicolás Figueroa, and Oscar Mauricio Caicedo Rendon
Department of Telematics Engineering, University of Cauca
Popayán, Cauca
Email: davidotero, nataliaquino, cfigmart, omcaicedo@unicauca.edu.co

***Abstract*—Flow routing algorithms are a fundamental part of telecommunication networks since they allow all nodes to communicate with each other. Routing algorithms must be efficient to avoid network's degradation. Traditional routing protocols usually use fixed link weight assignment and shortest path routing which that cause link overuse resulting in the service degradation. The appearance of Software Defined Networking (SDN) offered multiple advantages over traditional networks, such as a global view of the network and a programmable control and data planes entitling the introduction of new technologies.**

In a first attempt, traditional routing protocols were implemented on the top of SDN, improving routing convergence. Still, they inherited limitations such as link overuse and did not consider the network's historical information to nourish decision-making. SDN's programmability made it possible to integrate Machine Learning (ML) techniques to take advantage of the network's historical information opening the way for novel routing strategies. However, ML-based solutions are still dependent on conventional routing protocols, therefore, do not fully exploit the network status. We propose a routing algorithm whose cognitive learning improves network performance and takes advantage of the broad potential of SDN.

***Index Terms*—Deep Reinforcement Learning, Routing, Software Defined Networking**

I. Introduction

SDN manages networks effectively, reduces the operation costs, and promotes the development of networks through programmability [1], [2]. Therefore, SDN has been the focus of growing attention due to its efficiency and scalability to handle high traffic volumes in highly complex networks [3]. The potential of SDN compared to its conventional counterparts highlights the importance of reassessing how networks are implemented and the algorithms used in them for their operation since the search for effective routing plays an essential role in increasing SDN performance.

The use of Shortest Path First (SPF) [4] or link state routing algorithms within conventional routing protocols in SDN favors situations of congestion and degradation of the network environment by using a rigid allocation of weights in the calculation of routes [5]. Some solutions implement conventional routing protocols such as Open Shortest Path First (OSPF) [6]–[10], Routing Information Protocol (RIP) [11] or Border Gateway Protocol (BGP) [12]–[15]. Although these solutions achieve improvements regarding delay and packet loss compared to implementations in traditional network architectures, these solutions incur limitations when making routing decisions in the network. Fixed metrics in rigid weight assignments for route calculation are a drawback when facing intrinsic network variability in current environments. Therefore, the service ends up degrading due to link congestion. Furthermore, these protocols do not use prior experience in making routing decisions.

With time and the contants search for new ways to exploit SDN in particular, the programmability of SDN was seen as a routing solution toghether with ML capabilities in data processing, and decision-making [16]. Some studies [17]–[23] propose routing solutions based on Reinforcement Learning (RL) within SDN to find an optimal routing policy in terms of delay, packet transmission rate, packet loss rate, among others. Despite seeing an optimal policy in an RL algorithm requires many iterations, which can hamper proper routing algorithm performance in continuously time-varying environments with large node and link number [24].

Works like [25]–[37] are based on Deep Reinforcement Learning (DRL). These works create routing algorithms that adapt themselves to the state of the network and improve the distribution of resources, optimizing network performance in terms of delay and packet transmission rate.

The contributions above show different approaches related to routing, and thus, we can spot a couple of common factors to address. For example, we have seen a need for real topology usage and appropriate traffic matrices to nourish the implementation and evaluation of the routing algorithms. On the other hand, the routing decisions are prone to dismiss device state metrics. Hence, the network loses its global view by neglecting important device status metrics, using only link metrics.

We propose CoRA a **Co**gnitive **R**outing **A**lgorithm that uses the DRL technique Deep Deterministic Policy Gradient (DDPG) and the Knowledge-Defined Networking (KDN) paradigm to route flows on the SDN architecture, altogether with the link and device state metrics usage to dimension the network state globally. The enriched learning process of CoRA's agent creates routing policies based on efficient and cognitive decision-making to install optimal routes on the forwarding devices, thus permitting rapid adaptation to the current network's always-changing environment. In summary, this paper contributes to a novel routing algorithm that employs device and link metrics computed by a DDPG agent to convey the optimal routes for a node pair. The results show that CoRA performance is superior to Deep Reinforcement Learning and Software-defined networking Intelligent Routing (DRSIR); a Deep Q-Networks (DQN) proposal, therefore, is a promising solution proving the significance of global dimensioning of the network and the effectiveness of DRL for cognitive solutions in the routing matter.

The rest of the paper is organized as follows. Sections II describe the related work, section III details our approach by introducing the DDPG routing architecture, CoRA's agent, and the routing algorithm. Section IV presents the CoRA prototype and the results of its evaluation. Section V offers the conclusions and suggestions for future work.

II. Related Work

This chapter describes the most representative works regarding routing based on SDN, ML, and device metrics. Among these are mechanisms addressed through different models and strategies that allow confronting the diverse solutions and approaches we have developed for this undergraduate work.

A. Classical Routing

The proposes [6]–[10] use of OSPF in SDN to improve the QoS of a large-scale networks in terms of delay, packet transmission rate, and packet loss. Researches like [12]–[15] implement BGP over SDN. Particularly, [13] uses SDN's programmability to adapt BGP so that the distance traveled is less, resulting in reduced latency. [15] proposes a BGP architecture called OFBGP, implemented as an SDN application, improving scalability and availability regarding the BGP. However, SPF-based protocols have difficulty when facing network variability, incurring congestion, and the subsequent service degradation.

The Dijkstra link-state routing algorithm traditionally considers the shortest path or performance metrics (e.g., delay rate, packet transmission rate, and packet loss) to assign costs to network links. Dijkstra is used in solutions like [38]–[40] within SDN. These proposals create strategies that take advantage of the flexibility of SDN, such as dynamic weight assignment to links, or nodes of the network, benefiting performance by reducing network latency and congestion. Nevertheless, even when speaking about dynamic monitoring in routing algorithms, the weight assigned to the links or nodes does not update in real-time, creating a flaw in representing the network in its current state. The routing decisions of the works mentioned in this section are calculated through conventional routing algorithms, which take into account only the network's current state, missing the opportunity to exploit the previous state—subsequently missing knowledge about previous decisions to generate more intelligent choices in the future. These shortcomings impact when aiming to reach prediction capabilities or a more competent global view.

B. Reinforcement Learning-based Routing

The RL trend is prevalent within routing in SDN because RL does not need prior information on the network state. Proposals like [17]–[23], [41] improve routing capabilities by using RL over SDN. These proposals surpass conventional algorithms in terms of delay, congestion, adaptability, and intelligent management of networks with more excellent QoS provisioning, managing to obtain routes on demand even in large-scale networks. However, the high number of iterations necessary to find the optimal routing strategy and the extensive tables in Q-learning based solutions result in long convergence times and make essential efficiency improvements, increased processing speed, and storage optimization. [20] suggests that in the future, implementing other intelligence techniques (e.g., DRL, DNN) will be essential to overcome the limitations mentioned above. In [21], the need for adequate flexibility and scalability becomes evident when it is evaluated exclusively in an environment with a fixed traffic rate and bandwidth.

C. Deep Reinforcement Learning-based Routing

[25]–[27], [35] use DQN for the routing process. [25] uses DRL to minimize latency and packet loss, achieving its goal well above the performance among other ML algorithms used for this purpose (e.g., ARMA - *Auto- Regressive Integrated Moving Average*). In [26], the authors manage a variant of DQN called DDQN, where the agent's reward is associated with the delay and the packet transmission rate, surpassing the performance of OSPF in these metrics.

In [27] algorithm uses two DNNs to treat mouse and elephant flows. Then, they set a reward for each flow according to the specific metrics required. Mouse flows take into account packet loss and average delay. Besides, elephant flows bear the loss rate and the transmission of packets as metrics. [27] optimizes delay, packet transmission rate, and packet loss metrics, outperforming algorithms such as ECMP (Equal-Cost MultiPath Routing).

The works [29]–[34] propose DRL routing algorithms, employing the critical actor technique in conjunction with DDPG to obtain the optimal policy in a single step. In [29] and [30], the DRL agent only considers delay for reward maximization, resulting in improvements compared to traditional algorithms like SPF.

In [34], the maximization of the reward is given by the packet transmission rate between the source, destination, and the link delay. Furthermore, [33] uses packet loss and latency. Besides, [31], [32], [35]–[37] better dimension the link state, including all the above metrics, packet transmission, packet loss, and delay. Especially [35] introduce DRSIR, DRSIR proposes a novel space of states and actions model that characterizes the state of the paths obtaining results outperforming Dijkstra, and RL approaches.

[28] develops two DRL models, DQN and DDPG to optimize the packet transmission rate. Both models show better performance than OSPF concerning this metric. On the other hand, when comparing the performance of these DRL algorithms, it becomes clear that the performance of DDPG exceeds DQN, with 47% and 40% optimization in packet transmission rate, respectively.

In the proposals mentioned earlier, the agent's reward exclusively involves metrics associated with the links state (e.g., packet transmission rate, delay, and packet loss) but not the network device state. Unconsidering the device state supposes a limitation regarding the global characterization of the network environment.

III. CoRA

This section shows the architecture and the components of CoRA, as well as how the DRL agent is constituted.

A. Overview

We present our solution facing the shortcomings found in the literature regarding routing involving ML over SDN. In the Journey of researching novel approaches to enhanced routing algorithms, we have decided to use the KDN paradigm to ease management and monitoring, jointly with DRL performing over an SDN network. We engaged with the importance of global network characterization through link state and device state metrics to train the algorithm adequately, achieving efficiency and enhanced performance. As the current routing solutions using DRL agents do not directly consider the device metrics, we recover device metrics such as the switch queue´s occupancy as well as the link state, available bandwidth, delay, and packet loss to find the optimal path policy. To create a final result, first; the Control Plane collects the performance metrics, which are later processed by the Management Plane. The processed data is delivered to the Knowledge Plane, creating an environment in which a DDPG agent learns and ultimately provides the optimal route for each pair of nodes. CoRA is a proposal with a novel objective to reach a cognitive DRL agent that outperforms approaches related to traditional routing and modern routing. DRSIR, a modern routing proposal that implements KDN over DQN, was the base for our own. Furthermore, each node delivers directly the path to follow for the packets, avoiding using classic routing algorithms. DRSIR overperforms all traditional approaches [35].

B. Architecture

CoRA's architecture is built in SDN, which supports the network's automated management and control. At the same time, following the KDN paradigm to compute network information (i.e., link state and device state metrics) in the intelligent DDPG agent to execute cognitive decision-making in the flow routing. Below, we explain the solution's sequential operation, and Fig 1 depicts it in detail.

❶ The Control Plane periodically queries the Data Plane to collect network information.
❷ The Management Plane receives *Topology discovery* and *Statistics* information from the Control Plane to Process and stores the network state.
❸ The Knowledge Plane receives information from the Management Plane.
❹ The DDPG agent explores and exploits the possible routes for each source-destination node pair. Eventually gets the best routing path for all pairs of nodes in the network.
❺ The Knowledge Plane stores in the *Routes data repository* the data about the routes computed by the DDPG agent.

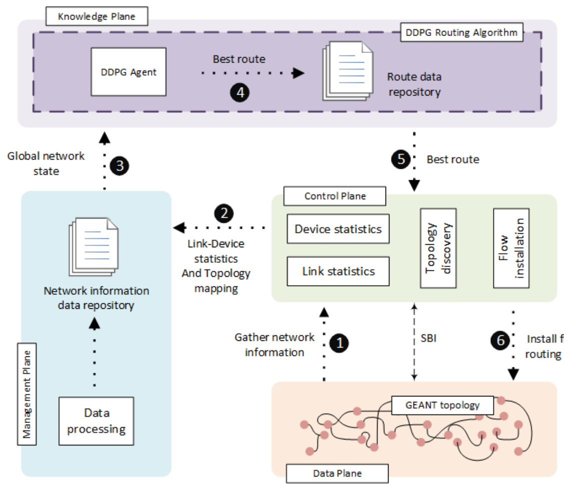


Fig. 1: CoRA Architecture

❻ The Control Plane installs the best routes in the flow tables of the switches.

CoRA have the next planes with their components:

The **Data Plane** holds the forwarding devices and the links that connect them. Performs basic tasks like responding to queries with messages containing information about network topology and state. Essentially CoRA's Data Plane is not running complicated errands to manage the limited network resources like link bandwidth or buffer size in the devices. Instead, those resources are closely monitored to secure adequate performance and benefit the network operation. The ideal performance of the Data Plane lies in developing a sophisticated routing policy through the Knowledge Plane that is further installed in the routing tables.

The **Control Plane** requests queries to gather information from the Data Plane to construct the global view of the network; it is also in charge of installing the routing tables.The Control Plane comprises four modules, device statistics, link statistics, topology discovery, and flow installation.

The *topology discovery module* sends the *feature-request* message from OpenFlow to the Data Plane Switches. The Devices respond with a *feature-replay* message containing the individual switch identifier and port information such as port number and state. This module sends an OpenFlow *packet-out* with a payload containing a Link Layer Discovery Protocol (LLDP) packet to all switch ports. The first switch will send LLDP packets through the port to a neighbor switch. When the neighbor device receives the LLDP packet, it will be sent to the controller a of an OpenFlow *packet-in*. The message has the origin switch's id and the port number, besides the destination switch's id and port number, so the module can identify which devices are neighbors and from which port communicate.

The *link statistics module* sends OpenFlow's *Multipart Messages* to the Data Plane devices each monitoring period every *t* seconds. The *Multipart Messages* have messages types to hand over specific switch information like port statistics, flow, and flow tables. Due to the need for link statistics, the port statistics employ *port-stats* messages to consult the transmitted and received packet and byte amount in the port.

During each monitoring period, the statistics collection modules collected information on the network state and the Management Plane and carried it to feed the agent in the Knowledge Plane. The *device statistics module* uses Data Plane petitions each monitoring period; the petition's reply contains the packets' load at the switch queue. The queue at the *t* moment is stored with the link statistics metrics and the network topology to be subsequently processed in the Management Plane. The Knowledge Plane delivers to the *flow installation module* the optimal path for each node pair generated according to the current network conditions. The path installation is proactive; the paths are installed periodically and change depending on the network state. This module uses the information obtained by the *topology discovery module* to configure which switch port should send the incoming flow properly.

The **Management Plane** comprises two components, the *data processing module* and the *network information data repository*.The *data processing module* receives raw data gathered by the Control Plane through the link, device statistics and topology discovery modules. This module processes the raw data to calculate every path metric (e.g., delay, packet loss, available bandwidth, and queue occupation) that gets delivered to the Knowledge Plane to enter the algorithm. The *network information data repository* stores the metrics calculated by the previous module containing the tuple source destination node and its respective metrics. An example of an entry is: (source = $node_1$, destination = $node_2$, av_bw = $500Kbps$, delay = $5.1ms$, loss =0.1%, queue_occu= 10pkts). So the Management Plane ensures the optimal performance of the network over time.

The **Knowledge Plane** contains the DDPG-based routing algorithm where the DDPG-agent and the *route data repository* lie. Thanks to the Agent, the Knowledge Plane transforms information into knowledge, gathering the global view of the network and computing it to achieve the algorithm's objective. The *route data repository* holds path information; each entry is a tuple of source, destination, and best path, with source and destination being node numbers, and the best path is a series of nodes that shape the path under consideration.

C. Cognitive Routing Agent

This subsection explains how the agent can find the best route from all possible network paths. CoRA's agent uses DDPG a DRL technique that merges DQN and Deterministic Policy Gradient (DPG), an optimal combination that, along with experience replay and slow learning target Deep Neural Network (DNN)s make well-developt decisions, stable and efficient learning when developing cognitive policies 2. It is demonstrated that DDPG robustly solves problems related to complex action spaces [42], like the one that gathers our commitment in this undergraduate work. Aspiring to take a step forward in ML routing algorithms and improving the previous similar proposes in this field, we choose DDPG as CoRA's agent technique. Interacting with the network environment, the agent converges to an optimal policy for the best route for each node pair. The Knowledge Plane builds the environment using the network state information gathered by the Management Plane. Each iteration makes the agent act on the current state and choose the action that minimizes the reward; then, the agent delivers the next state. The reward corresponding to the action taken conveys the path cost, taking into account the available bandwidth, packet loss, delay, and queue occupation. That's to say, the agent's decision will be influenced by the path with greater available bandwidth, least packet loss, delay, and queue occupation. The following subsections explain the fundamental features of a DDPG agent, state space, action space, and reward process.

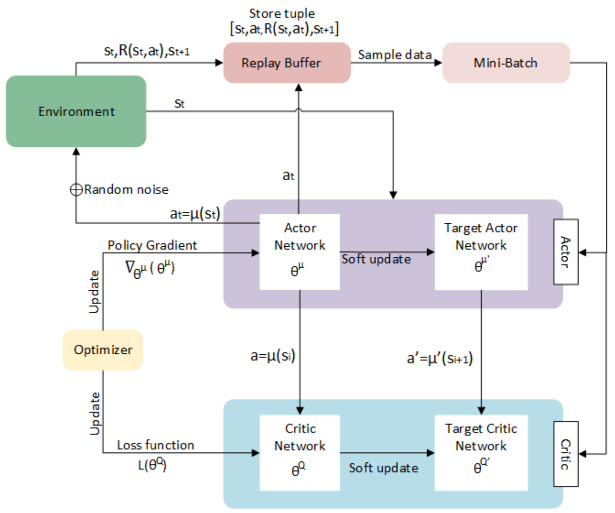


Fig. 2: DDPG-agent of CoRA based in [43]

1) Deep Deterministic Policy Gradient: DDPG contains four DNN, the first two are the Q-Network or critic network represented with Q and the deterministic policy network or actor network represented with μ. The next two are the target Q, represented with Q' and the target policy network represented with μ'. The last two networks are a mechanism original for DDPG to stabilize the learning of the DDPG's agent. DDPG can be divided into experience replay, actor and critic network updates, optimizer, target network updates and exploration.

The **experience replay** in DDPG updates the neural networks during training to prevent the correlation in the data used to update the neural networks. And seek independent distribution among the dataset, so the algorithm achieves a well-developed learning process. The experience replay overcomes those limitations by creating a finite cache to save data regarding each learning episode's state, action, reward, and next-state tuples. Next, a random sample of these stored tuples nourishes the training; usually, the experience replay evicts the oldest episodes of information, giving way to save the most recent ones. Is proven that the use of experience replay improves efficiency and stability by storing a finite number of the most recent tuples of training [44].

The following methods use the **actor and critic network updates**. We use the Bellman equation, similar to Q-learning to update the critic DNN to obtain the optimal action value.

$$y_i = r_i + \gamma Q'(S_{i+1}, \mu'(S_{i+1}|\theta^{\mu'})|\theta^{Q'}) \quad (1)$$

In the equations 1, $\theta^{Q'}$ is the weight of neurons in the target critic network, $\theta^{\mu'}$ is the weight of neurons in target actor network. The target actor and the target critic networks calculate the next-state Q values. Then the target updated values are used to minimize the mean squared loss between the updated Q value and the original Q value:

$$L = \frac{1}{N}\sum(y_i - Q(s_i, a_i|\theta^Q))^2 \tag{2}$$

Where N is batch size. The actor network's policy function aims to minimize the expected return.

$$J(\theta) = \mathbb{E}[-Q(s,a)|_{s=s_t, a_t=\mu(s_t)}] \tag{3}$$

Then we derivate the function to the parameters of the policy. Since DDPG is an off-policy driven algorithm, the summation mean of the gradients from the experience batch is added as follows.

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum \nabla_a(-Q(s,a|\theta^Q)|_{s=s_i,a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}) \tag{4}$$

Another vital part of the mechanism is the **optimizer**. The optimizer is an algorithm used by different ML techniques to reduce loss or cost functions by updating the weights of the DNN [45]. The optimizers have a parameter called the learning rate that determines how much the weights of the DNN are updated, improving or worsening the agent's performance. There are different types of optimizers, but Adam is the most used in Deep Learning (DL) [46].

Updating the target networks is critical for learning stability. The parameters of the target network cannot be the same as those used to train the main networks. That is why we update target networks with a time delay once per major network update. The DDPG algorithms update the target network using the Polyak average.

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \tag{5}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \tag{6}$$

Where the parameter τ called target network updater is usually near 1 $\tau << 1$.

To **explore** potentially beneficial actions, noise is introduced to incorporate randomness. To add noise to the action, the original DDPG paper [42] use and recommends the Ornstein-Uhlenbeck Process. This process generates random values for a mean and standard deviation specified, that furthermore depends on the previously generated random value.

2) State Space: In the DRL agent, the state space corresponds to all the pairs of nodes that can establish communication in the network. For example, a state s_i can be node x as the source and node y as the destination. Another state s_j could be the inverse of state s_i, with node y as the source and node x as the destination. Therefore, the number of possible states is the permutation of the network's total number of existing nodes.

$$|S| = |P(N,k)| = \frac{N!}{(N-k)!} \tag{7}$$

Where N is the total number of nodes in the network, and k is equal to 2, given that one source node and one destination node.

3) Action Space: The action space is a set of valid choices to continue the interaction and advancement inside the environment, that is, the state space. The agent has a set of actions A_t for any given state. Thus, the Action States is a list of the possible k paths per state S_t. The *Network Information Repository* saves the possible k paths for state S_t. DDPG is a DRL technique for continuous actions; therefore, we discretize the action generated by the agent to obtain one path out of k paths. This approximation is valid since [28] uses the same action space for a DDPG and DQN agent. In addition, in [47], they discretize the action delivered by the DDPG agent to make a decision based on their problem, obtaining good results.

4) Reward: The reward function incentivizes the algorithm to converge to the optimal policy in the long term. Consequently, the value of the reward function represents the cost of a potential path in the Action Space for every state. The equation 8 defines the Reward Function inversely proportional to the mean available bandwidth in the path bwa_{path} and directly proportional to the path delay d_{path}, path packet loss ratio l_{path}, and queue device's occupation qo_{path}.

$$R = \beta_1 * \frac{1}{bwa_{path}} + \beta_2 * d_{path} + \beta_3 * l_{path} + \beta_4 * qo_{path} \tag{8}$$

The values β_1, β_2, β_3, and $\beta_4 \in$ [0,1] can be modified to furnish weight to a specific metric into the Reward Function.

In the Reward Function, the metrics must be all normalized using equation 9 known as the Min-Max, advised in [48] to improve accuracy. Since the metrics in the algorithm are in different units, so one metric is not prevalent over the others, delivering an exact outcome.

$$\hat{x_i} = a + \frac{(x_i - min(X)) * (b-a)}{max(X) - min(X)} \tag{9}$$

The Min-Max method involves scaling the values of the metrics to an arbitrary interval; [a,b]. Where $\hat{x_i}$ is the value to normalized, and X is a set of values, the equation 10 is the normalized version of 8

$$\hat{R} = \beta_1 * \frac{1}{bw\hat{a}_{path}} + \beta_2 * d\hat{_{path}} + \beta_3 * l\hat{_{path}} + \beta_3 * qo\hat{_{path}} \tag{10}$$

Link Metrics The Management Plane is in charge of calculating both link and device state metrics. This plane computes link throughput and loss using the number of packets that pass through the links connected to the switch port since it samples the number of bytes transmitted or received.

Comparing the retrieved values at two different instants is possible to discover the instantaneous throughput. When the controller sends to the Data Plane a *port-stats* message at time t_1, the number of bytes received b_{t_1} is replied. A second request message is sent, and the reply b_{t_2} contains the number of bytes received at t_2, the duration of the interval that separates times t_1 and t_2 is the period p.

$$bwu_{link} = [\frac{(b_{t_1} - b_{t_2})}{p}] \tag{11}$$

Then to acquire the available link bandwidth, we make the difference between the link capacity cap_{link} and the instantaneous throughput.

$$bwa_{link} = cap_{link} - bwu_{link} \tag{12}$$

The controller sends *port-stats* of the respective ports of the switches belonging to a link. The number of bits sent btx_i by the port is observed in the response, and the number of bits received by the other port of the neighboring switch brx_j is observed. With these data, equation 13 is applied to calculate the instantaneous loss ratio.

$$l_{link} = \frac{btx_i - brx_j}{btx_i} \tag{13}$$

Following the process described in [49], we compute the instantaneous delay using LLDP and OpenFlow messages. A LLDP message sent by the controller c_0 does the path c_0-s_i-s_j-c_0 with (s_i, s_j) being the link that connect the switches s_i and s_j. The time between the transmission and reception of the LLDP message is captured in the message's time stamp, that is, $d_{lldp_{cij}}$. Then, time taken from c_0 to the s_i's port is estimated as half the time that passed between the transmission time and the reception of the OpenFlow *echo-request* and *echo-reply* messages sent by c_0 to s_i, that is $d_{c_0-s_i}$. Similarly, the time elapsed between s_j to c_0, is$d_{c_0-s_j}$, finally the equation 14 depicts the instantaneous delay in the link (s_i, s_j).

$$d_{s_i-s_j} = d_{lldp_{cij}} - d_{c_0-s_i} - d_{c_0-s_j} \tag{14}$$

The Management Plane processes the link metrics to find the path metrics needed for the overall network status. The link metrics (e.i., bwa_{link}, l_{link}, $d_{s_i-s_j}$) are processed to find the path metrics as follows. The Management Plane obtains the lower bandwidth available from all belonging links to the path P to fetch P's available bandwidth then.

$$bwa_{path} = \min_{i \in P}(bwa_{link_i}) \tag{15}$$

The sum of the delay from all the links that reside in the path P is equal to the total delay of the path.

$$d_{path} = \sum_{i \in P} d_{link_i} \tag{16}$$

Estimating the path loss can be seen as the failure probability in a system of series-coupled components since every link is independent. At the same time, the links are arranged contiguously to construct a path. Thus, the path loss responds to the equation below.

$$l_{path} = 1 - \prod_{i \in P}(1 - l_{link_i}) \tag{17}$$

Device Metrics. As show in figure 3, the switch ingress queue occupation is the device metric selected to achieve the global network state to acquire an optimal routing policy. The

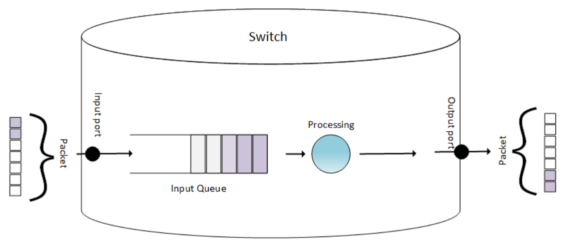


Fig. 3: Switch ingress queue occupation

Data Plane has an HTTP server that employs Linux Traffic Control (TC) tool to extract the queue occupation from the devices. TC is a utility that enables configuring the kernel packet scheduler likewise model packet delay, loss, bandwidth, and switch queue with a set packet space [50]. TC requires the *Device Statistics Module* in the Control Plane to send an HTTP type GET request to the Data Plane server. The GET request first identifies each switch network interface, and then, the queue occupation is ready to be consulted. TC finds the number of packets queued on each network interface and returns the GET request response in a JavaScript Object Notation (JSON) format to the *Device Statistics Module* joining the interface's name with the number of packets queued. The interface's name contains the switch and the port number to identify which link the information came through in the process.

To find the complete path queue occupation, the Management Plane makes the sum between the queue of the links integrating the path, as shown beneath;

$$qo_{path} = \sum_{i \in P} qo_{link_i} \tag{18}$$

D. Cognitive Routing Agent

The algorithm 1 describes the process of finding the optimal path between a source-destination node pair sequentially while considering the link metrics; available bandwidth, packet loss, delay, and as a device metric, the switch queue occupation. The algorithm inputs are; learning episode number n which is a sequence of states, actions and rewards, that ends with terminal state in our case the convergence to a optimal policy. Other inputs are the possible paths list per state k, the network path state, the discontinuity factor, the learning rate from the optimizer, the target network update value, number of step by episode, and the batch size, that's to say, the number of samples processed before the network update. The algorithm delivers to the *flow installation module* the best route from every node pair for its installation.

To start the algorithm, the Management Plane processes the path metrics to build the environment. The environment is created with the states, and actions with their respective reward. In the second line, the Target DNNs begin with the same weight as the Actor and Critic DNNs, aiming to stabilize the learning process. Then the Replay Memory will record the agent's experiences.

The agent trains by interacting with the environment previously created in a set of n episodes (lines 6 and 18). In line 6, the agent initializes the state with a node pair chosen randomly. The agent iterates m-times each episode (lines 8 to 17). Subsequently, the agents start to take an action regarding the actual state and take a path from the k paths using the actor DNN. The environment acquires this action, and the equation 10 conveys the reward and next state s_{t+1}. Afterward, the replay memory stores a tuple assembled with the current state s_t, the action a_t taken by the actor DNN, the reward r_t, and s_{t+1}. A strategy to train the agent in an off-policy manner is to extract a size N mini-batch of information from past experiences (line 12).

We calculate the expected value with equation 1 (line 13) using the mini-batch of information and the critic target DNN. This computation reduces the standard deviation in the learning process and achieves faster, more efficient, and more stable convergence. Then, we calculate the mean squared loss 2 to update the critic DNN (line 14).

Conveying the policy gradient (line 15) using the chain rule with the critic DNN, we update the actor DNN. The actor DNN computes the actions regarding the mini-batch states, then, using the critic DNN we find the expected values generated with the tuple from the mini-batch. To minimize the reward, the loss function to find the policy gradient from the actor would be the mean of the expected values. Using the soft update, the agent state, the actor, and the critic target DNNs are actualized (line 16).

At the end of the learning process, we use the actor weights to find a path that generates the smaller reward from each node pair's k path lists (line 20). Those paths are stored and sent to the Control Plane so the *flow installation module* configures the routes in the Data Plane. Then, we verify if the learning time was longer than the monitorization time of the network; if that's the case, we wait for the control plane to obtain the new path states. Finally, we use the new path states to reconstruct the environment.

Algorithm 1: CoRA algorithm

```
Input  : Number of learning episodes: $n$
         Number of steps by episode: $m$
         List of "k" paths per state: $k_{paths}$
         Network path-state
         Discount factor: $\gamma$
         Learning rate
         Target network updater: $\tau$
         Batch size: N
Output : Set with the best routing path for all pairs of nodes in
         the network
1  Build Environment Network with Network path-state
2  Initialize critic and actor targets NNs with weights
     $\theta^{Q'} \leftarrow \theta^{Q}, \theta^{\mu'} \leftarrow \theta^{\mu}$
3  Initialize Replay Memory
4  while $true$ do
5      for $episode$ to $n$ do
6          Initial state $S_t$
7          for $step$ to $m$ do
8              Select action $a_t$
9              Execute action on environment
10             Get reward $r_t$ and next state $s_{t+1}$
11             Store tuple ($s_t$, $a_t$, $r_t$, $s_{t+1}$) into Replay Memory
12             Sample a random mini-batch from Replay Memory
13             Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
14             Update critic $L = \frac{1}{N}\sum(y_i - Q(s_i, a_i|\theta^{Q}))^2$
15             Update actor policy to minimize reward $\nabla_{\theta^{\mu}} J \approx$
                 $\frac{1}{N}\sum \nabla_a(-Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}}\mu(s|\theta^{\mu})|_{s_i})$
16             Update the target networks
                   $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$
                   $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$
17             Update state $s_t = s_{t+1}$
18         end
19     end
20     Use final $\theta^{\mu}$ to retrieve the path from $k_{paths}$ that corresponds
         to the action with the lowest reward for each state
21     Store the set of paths for all pair of nodes in the controller to
         install on data plane;
22     if $timelearning < timemonitoring$ then
23         wait $timemonitoring - timelearning$
24     end
25     Retrieve new network path-state
26     Rebuild Environment network with new network path-state
27 end
```

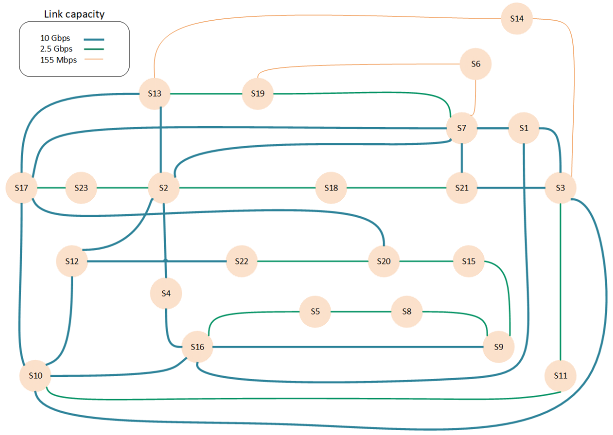


Fig. 4: GÉANT topology [41]

IV. Evaluation

This section presents in subsection A the test environment where CoRA was evaluated and the prototype. Subsection B shows how the traffic for the evaluation was generated. Subsection C presents how the hyperparameters of the agent are configured. Finally, subsection D presents the results and analysis of CoRA versus DRSIR.

A. Test Environment and Prototype

The agent works on the 2004s GÉANT network topology [51]. This topology comprises 23 nodes with 37 links. The capacity of the links is: 19 links of 10Gbps, 14 links of 2.5Gbps, and 4 links with 155 Mbps. Emulating the actual link capacity is not possible due to computational resource limitations, mainly CPU restrictions. Therefore the capacity of the links was scaled in a 100 ratio. The links with 10 Gbps, 2.5 Gbps, and 155 Mbps of capacity tuned into 100 Mbps, 25 Mbps, and 1.55 Mbps, respectively. Since the links have diminished capacity, the traffic must shrink in the same proportion. The network topology was be emulated with Mininet version 2.2.2 [52], and the Switches with Open Virtual switch 2.13.8.

To model the links, we used TC tool from Linux, which allowed us to specify the available bandwidth and the input queue size in packet number. In commercial switches the queue size in depends on the model, the manufacturer, and the traffic. For our case we defined an input queue size of 100 packets since it produced a reasonable balance between delay and loss [53].

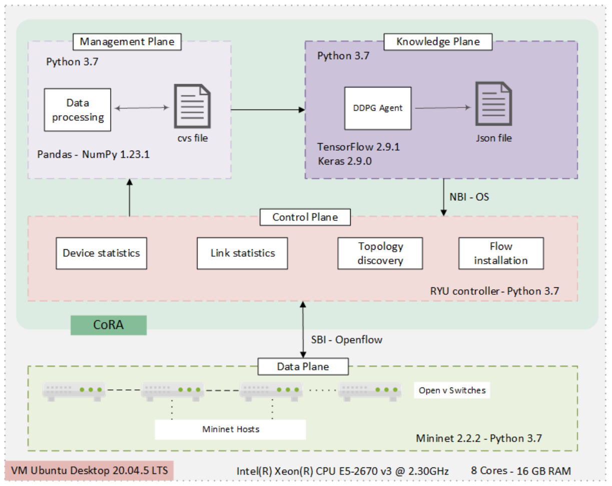


Fig. 5: CoRA Prototype

Figure 5 portrays CoRA's prototype where the Data Plane that comprises the GÉANT network topology was emulated with Mininet. Furthermore in the Control Planewe rendered the controller, *Link State Statistics*, *Device State Statistics*, *Topology Discovery*, and *Flow installation modules* additionally with RYU API, and to gather device and link status data Ryu's tools allow the process. The Management, Knowledge Planes, and the DDPG agent were deployed with Python 3.7, Keras 2.9.0, and TensorFlow 2.9.1 to carry out the ML workforce. For the data processing, we used libraries specialized in data manipulation, NumPy 1.23.1 and Pandas. Every plane was executed on Ubuntu Desktop 20.04.5 LTS with a Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz using 8 cores and 16 GB of RAM.

B. Traffic Generation

To generate the traffic in the mininet emulation, we used an informatic network tool, *iperf2*. The *iperf2* scripts produced User Datagram Protocol (UDP) traffic according to a traffic matrix. The traffic matrix corresponds to a set of sixteen public intra-domain traffic matrices [54] of the GÉANT Paneuropean topology. We delimited the traffic matrix of the GÉANT topology to accommodate the level of congestion to verify the device metrics' essential role in the network's global view. Thus, we selected the heavier traffic day of the year sample and processed it, diminishing atypical values.

C. Learning Parameters Setup

There are different hyperparameters in the DDPG technique, which can modify the behavior of the reward, improving or depreciating the agent's performance. The hyperparameters that we are going to evaluate are:

- Number of hidden layers.
- Number of neurons per hidden layer.
- Discount factor (γ).
- Learning rate.
- Target network updater (τ).
- Batch size.

For the step number to finish an episode, DRSIR initially implemented 30 steps. Still, 30 made our algorithm converge more slowly, and with a couple of trials, we arrived at 45, which we proved to be the most accurate value. In the context of our state spaces, we need to monitor the tendency of the reward at the end of the episode, which makes sense when using 45 allows us to know the trend.

In figure 6a we vary the number of hidden layers in the neural network of the actor and the critic. We observe that modifying the number of hidden layers does not noticeably affect the behavior of the reward. However, it significantly affects the agent's convergence time, which is the 50th episode. Therefore, we will use only one hidden layer in the neural networks in actor and critic.

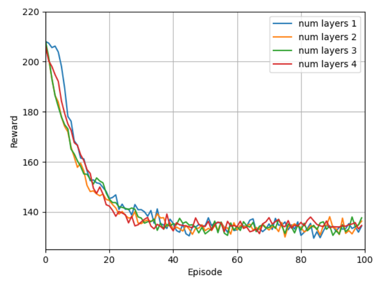
In the same way that was varying the number of hidden layers, varying the number of neurons in the hidden layers of the networks does not significantly affect the reward behavior of the agent, as can be seen in figure 6b. Varying the number of neurons exclusively influences the agent's convergence time. The more neurons, the longer the convergence time will be. Consequently, we took the lower number of neurons per layer, the reason why if we increased the neuron number, the convergence time increases, and the reward makes no better.

In Figure 6c, we vary the discontinuity factor (γ) with 0.1, 0.5 y 0.9. The behavior of the reward changes slightly by varying this hyperparameter. However, the γ value with a lower average reward is 0.1. The above may be due to the DRL agent design; the agent delivers the path to each pair of nodes, so the agent considers the immediate reward.

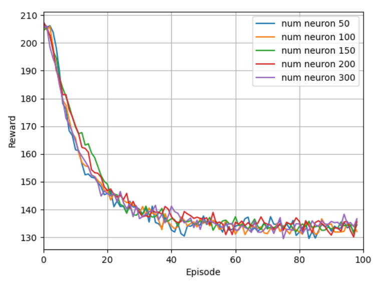
In figure 6d, we vary the learning rate in the optimizer. We start to vary from the value 0.001 to 0.05 to observe the impact on the reward. A high value of 0.05 in the learning rate causes the agent not to be able to converge; this is because updating the weights in the neural networks can be abrupt, causing the agent not to find the optimal weight for convergence. On the contrary, a low value of 0.001 in the learning rate means that the agent needs more training episodes to converge because the update weights of the neural networks are small. Whereby the update weights of the neural networks are small, it would need more steps to converge. The optimal value of the learning rate for the agent is 0.01.

Modifying the batch size does not imply a significant change in the behavior of the reward, except for a size of 10, as can be seen in the figure 6e. The size of 30 generates a slightly lower reward than the other batch sizes. Also, if this value is increased, the convergence time increases because the batch size is the number of times the agent has to relearn from that previous experience to avoid learning instability.

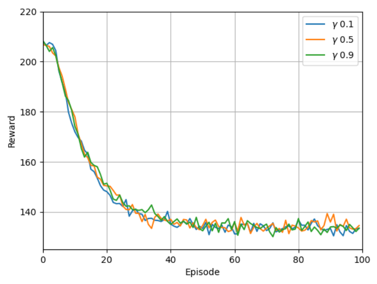
In Figure 6f, we analyze the effect of the target network updater (τ) on the behavior of the reward. τ value has to be considerably smaller than 1. Hence, we vary the τ value to 0.001, 0.005, 0.01, and 0.05. The behavior of the reward



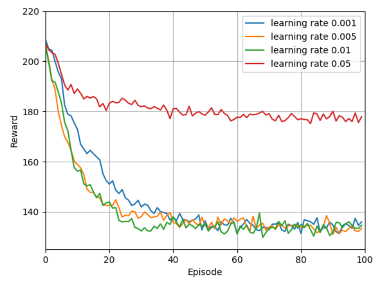
(a) Varying number of hidden layers.



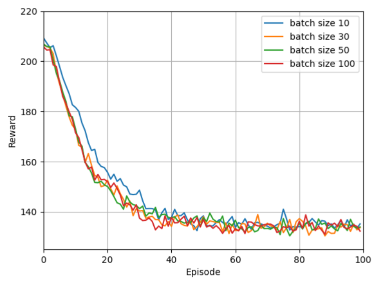
(b) Varying number of neurons per layer.



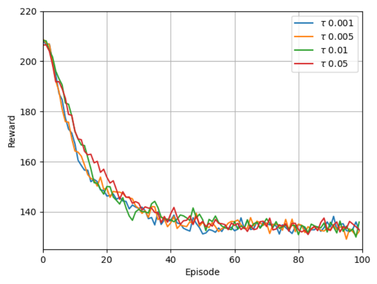
(c) Varying discount factor.



(d) Varying learning rate.



(e) Varying batch size.



(f) Varying target network updater.

Fig. 6: Learning parameters figures

related to the variation of τ value is not notorious. Either way, with a value below 0.001, we obtain a minor reward average.

D. *Results and Analysis*

In this section, we compare CoRA against DRSIR, the routing algorithm that was our approach's starting point. DRSIR was born to address RL and conventional algorithms shortcomings. DRSIR outperforms the conventional routing algorithms like Dijisktra, and RL algorithms as example Reinforcement Learning and Software-defined networking Intelligent Routing (RSIR), DRSIR previous proposal. DRSIR is a DQN algorithm that considers path-state metrics to produce proactive, efficient routing that adapts dynamically to network changes. DRSIR is evaluated on the GEANT topology with real and synthetic traffic matrices. We confront the two algorithms performance-wise, considering link metrics instantaneous throughput, average delay, loss rate, and the device metric average queue occupancy of the switches.

Figure 7a show the average queue packet occupancy on the Data Plane switches throughout the busiest hours of the day. On the heaviest traffic hours, we observe an increase number of packets dammed in the input queue of the switches due to the increase in packet forwarding on the links. CoRA reduced queue occupancy by 12.7% in regard to DRSIR. The decrease in queue occupancy is due to CoRA explicitly considering the queue occupancy of the switches in the reward function. Therefore, DNNs train to optimize the policy selecting less queue occupy path, that's to say the agent takes into account the queue occupancy. In addition, DDPG as learning technique is sophisticated when learning a policy that finds the best paths for each pair of nodes.

Figure 7b shows the average delay of all links during the most congested hours of the day. As well as the queue occupancy, in the hours with the highest traffic generated by the nodes, the delay of the links begins to increase due to congestion.

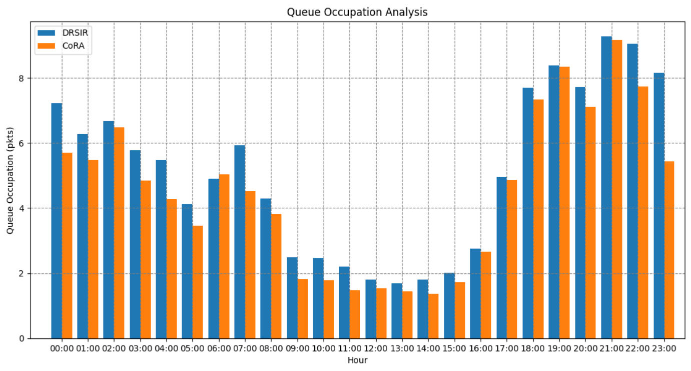
The CoRA algorithm obtains a delay improvement of 17% compared to DRSIR. When choosing the path, CoRA considers the number of packets waiting to be processed by the switches. This consideration avoids the links that present congestion in the input queue, reducing the packet queue time and consequently making a more efficient routing to decrease delay.

Figure 7c shows the average loss of all the links throughout the hours of the day with the most congestion. When the traffic increases, the congestion in the links becomes present, which induces an occupation in the input queue of the switch; when the number of packets exceeds the size of the queue, the next packet arriving will be discarded.

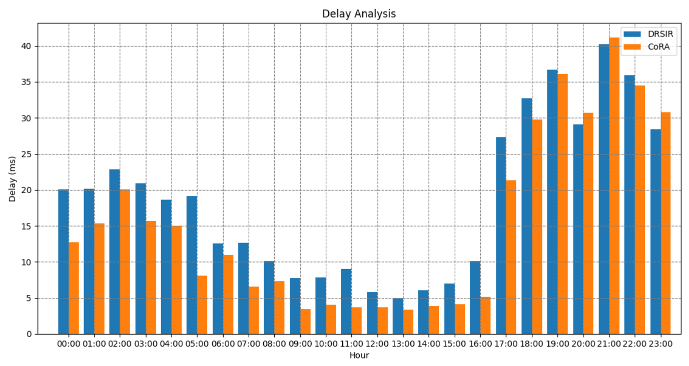
CoRA obtains 29.44% of loss improvement compared to DRSIR. We attribute this to CoRA's awareness of the queue state of the switches, preventing traffic from being sent over links where the queues have fewer packets waiting, preventing them from filling up to their maximum capacity and start discarding.

Figure 7d shows the average instantaneous throughput of all the links during the hours of the day of greatest congestion. The instantaneous throughput indicates how the routing strategy adequately distributes the traffic generated by all the links. The more traffic, the more needed strategy to avoid overloading the same links.

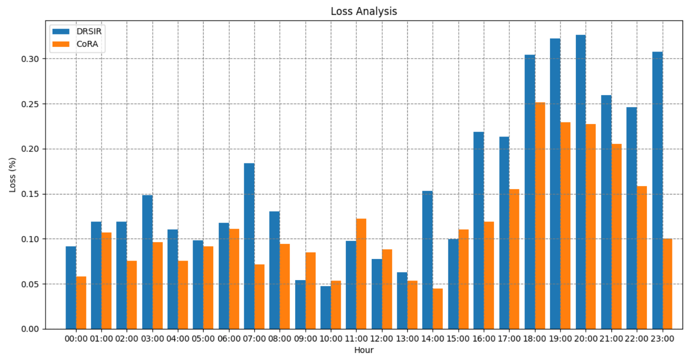
The CoRA algorithm obtains a 12.5% reduction in instantaneous throughput compared to DRSIR; this means that CoRA



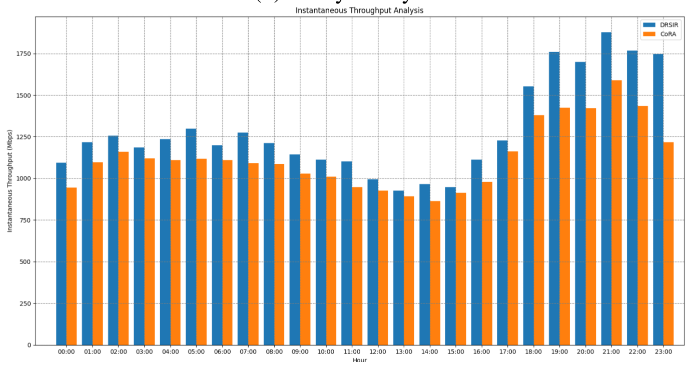
(a) Queue occupation Analysis



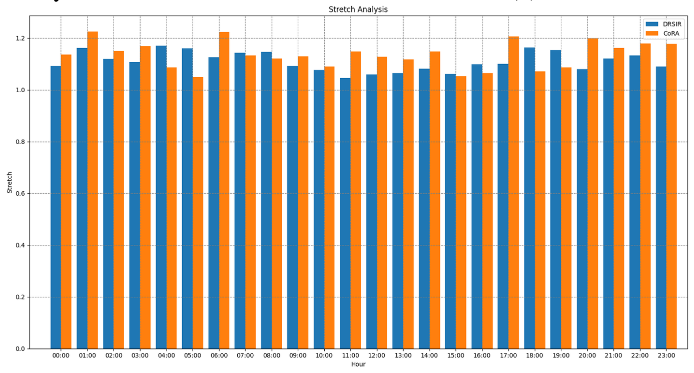
(b) Delay Analysis



(c) Loss Analysis



(d) Instantaneous throughput Analysis



(e) Stretch Analysis

Fig. 7: Performance metrics of CoRA and DRSIR

distributes the traffic along less congestioned paths, avoiding the overuse of links, therefore improving the performance of the above metrics.

Figure 7e shows the average stretch produced by the agents throughout the day. The stretch is calculated by the relationship between the size of the path chosen by the agents against the path with the fewest possible hops. CoRA gets slightly longer paths (<2%) than DRSIR. This result means that CoRA distributes the traffic in the paths regarding the metrics considered, not the shorter path. That way, it achieves an improvement in the previous metrics. sectionComparative Analysis

V. Conclusions

In this paper, we have introduced CoRA , an routing mechanism that implements the DRL technique, DDPG, and the link and device state data. Futhermore, CoRA was implemented on top of the GEANT topology by injecting it with real traffic matrices. CoRa performance was evaluated regarding stretch, loss, delay, instantaneous throughput, and queue occupancy.

CoRA improves the network performance compared to the DRSIR proposal. 12.7% reduced in queue occupancy, the delay is reduced by 17%, 29.4% improved loss, and a 12.5% reduced on the instantaneous throughput. These performance improvements are since the CoRA mechanism fully knows the network state. Making routing decisions, CoRa is aware of the link and device state, namely delay, packet loss, available bandwidth, and input queue switch occupation. The CoRA mechanism reduces the number of packets waiting in the switch input, and this means a reduction in the link delay. The decrease in packets in the switch input queue reduces the packet loss in the link because the mechanism prevents the number of packets from exceeding the maximum capacity of the switch queue, altogether avoiding queue overflow meaning packet discard. CoRA distributes traffic over less congested links, reducing the average instantaneous throughput.

To our knowledge, the proposed mechanism is the only one that uses a DRL agent without relying upon classical routing protocols. While monitoring the device's status together with the link status to nourish cognitive routing decisions to reinforce the reward getting optimal performance.

VI. Future work

- Research other device metrics, such as CPU and RAM consumption, to witness the impact of those device metrics on the network status.
- Implement different DRL techniques.
- Implement different ML techniques for the efficient configuration of the weights of the metrics in the agent's reward before traffic variations.

References

[1] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, 2014.

[2] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in software defined networks," *Computer Networks*, vol. 71, pp. 1–30, 2014.

[3] D. Gopi, S. Cheng, and R. Huck, "Comparative analysis of SDN and conventional networks using routing protocols," *IEEE CITS 2017 - 2017 International Conference on Computer, Information and Telecommunication Systems*, pp. 108–112, 2017.

[4] D. Awduche, J. Malcolm, J. Agogbua, M. D. O'Dell, and J. McManus, "Requirements for traffic engineering over mpls," *RFC*, vol. 2702, pp. 1–29, 1999.

[5] Y. Li, X. Li, and O. Yoshie, "Traffic engineering framework with machine learning based meta-layer in software-defined networks," *Proceedings of 2014 4th IEEE International Conference on Network Infrastructure and Digital Content, IEEE IC-NIDC 2014*, pp. 121–125, 2014.

[6] H. Zhang and J. Yan, "Performance of SDN Routing in Comparison with Legacy Routing Protocols," *Proceedings - 2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, CyberC 2015*, pp. 491–494, 2015.

[7] A. Rego, S. Sendra, J. M. Jimenez, and J. Lloret, "Ospf routing protocol performance in software defined networks," in *2017 Fourth International Conference on Software Defined Systems (SDS)*. IEEE, 2017, pp. 131–136.

[8] A. A. Khan, M. Hussain, M. Zafrullah, and M. S. Zia, "A convergence time optimization paradigm for ospf based networks through sdn spf protocol computer communications and networks (ccn)/delay tolerant networks," in *Proceedings of the International Conference on Future Networks and Distributed Systems*, 2017.

[9] S. Abdallah, A. Kayssi, I. H. Elhajj, and A. Chehab, "Network convergence in sdn versus ospf networks," in *2018 Fifth International Conference on Software Defined Systems (SDS)*. IEEE, 2018.

[10] R. Adrian, A. Dahlan, and K. Anam, "OSPF cost impact analysis on SDN network," in *2017 2nd International conferences on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*. IEEE, Nov. 2017. [Online]. Available: https://doi.org/10.1109/icitisee.2017.8285494

[11] E. Amiri, M. Reza Hashemi, and K. Raeisi Lejjy, "Policy-Based Routing in RIP-Hybrid Network with SDN Controller," no. September, pp. 1–8, 2018.

[12] "BTSDN: BGP-Based Transition for the Existing Networks to SDN," *Wireless Personal Communications*, vol. 86, no. 4, pp. 1829–1843, 2016.

[13] L. M. Elguea and F. Martinez-Rios, "A new method to optimize BGP routes using SDN and reducing latency," *Procedia Computer Science*, vol. 135, pp. 163–169, 2018. [Online]. Available: https://doi.org/10.1016/j.procs.2018.08.162

[14] V. Kotronis, A. Gämperli, and X. Dimitropoulos, "Routing centralization across domains via SDN: A model and emulation framework for BGP evolution," *Computer Networks*, vol. 92, pp. 227–239, Dec. 2015. [Online]. Available: https://doi.org/10.1016/j.comnet.2015.07.015

[15] W. Duan, L. Xiao, D. Li, Y. Zhou, R. Liu, L. Ruan, Y. Xia, and M. Zhu, "Ofbgp: A scalable, highly available bgp architecture for sdn," in *2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems*, 2014, pp. 557–562.

[16] G. Xu, Y. Mu, and J. Liu, "Inclusion of Artificial Intelligence in Communication Networks and Services," *ITU Journal: ICT Discoveries, Special Issue*, no. 1, pp. 1–6, 2017.

[17] T. Hendriks, M. Camelo, and S. Latré, "Q 2 -routing : A qos-aware q-routing algorithm for wireless ad hoc networks," 10 2018, pp. 108–115.

[18] S.-C. Lin, I. F. Akyildiz, P. Wang, and M. Luo, "Qos-aware adaptive routing in multi-layer hierarchical software defined networks: A reinforcement learning approach," 2016.

[19] C. Wang, L. Zhang, Z. Li, and C. Jiang, "SDCoR: Software Defined Cognitive Routing for Internet of Vehicles," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3513–3520, 2018.

[20] T. Mahboob, Y. R. Jung, and M. Y. Chung, "Optimized Routing in Software Defined Networks - A Reinforcement Learning Approach," in *Proceedings of the 13th International Conference on Ubiquitous Information Management and Communication (IMCOM) 2019*, S. Lee, R. Ismail, and H. Choo, Eds. Cham: Springer International Publishing, 2019, pp. 267–278.

[21] S. Kim, J. Son, A. Talukder, and C. S. Hong, "Congestion prevention mechanism based on q-leaning for efficient routing in sdn," in *2016 International Conference on Information Networking (ICOIN)*, 2016, pp. 124–128.

[22] Y.-H. Lu and F.-Y. Leu, "Dynamic routing and bandwidth provision based on reinforcement learning in SDN networks," in *Advanced Information Networking and Applications*. Springer International Publishing, 2020, pp. 1–11.

[23] M. B. Hossain and J. Wei, "Reinforcement learning-driven QoS-aware intelligent routing for software-defined networks," in *2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, Nov. 2019. [Online]. Available: https://doi.org/10.1109/globalsip45357.2019.8969320

[24] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y. C. Liang, and D. I. Kim, "Applications of Deep Reinforcement Learning in Communications and Networking: A Survey," *IEEE Communications Surveys and Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[25] E. H. Bouzidi, A. Outtagarts, and R. Langar, "Deep reinforcement learning application for network latency management in software defined networks," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, Dec. 2019. [Online]. Available: https://doi.org/10.1109/globecom38437.2019.9013221

[26] Y.-R. Chen, A. Rezapour, W.-G. Tzeng, and S.-C. Tsai, "RL-routing: An SDN routing algorithm based on deep reinforcement learning," *IEEE Transactions on Network Science and Engineering*, pp. 1–1, 2020. [Online]. Available: https://doi.org/10.1109/tnse.2020.3017751

[27] Q. Fu, E. Sun, K. Meng, M. Li, and Y. Zhang, "Deep q-learning for routing schemes in SDN-based data center networks," *IEEE Access*, vol. 8, pp. 103 491–103 499, 2020. [Online]. Available: https://doi.org/10.1109/access.2020.2995511

[28] W. xi Liu, "Intelligent routing based on deep reinforcement learning in software-defined data-center networks," in *2019 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, Jun. 2019. [Online]. Available: https://doi.org/10.1109/iscc47284.2019.8969579

[29] G. Stampa, M. Arias, D. Sánchez-Charles, V. Muntés-Mulero, and A. Cabellos, "A deep-reinforcement learning approach for software-defined networking routing optimization," *arXiv preprint arXiv:1709.07080*, 2017.

[30] J. Chen, Z. Xiao, H. Xing, P. Dai, S. Luo, and M. A. Iqbal, "Stdpg: A spatio-temporal deterministic policy gradient agent for dynamic routing in sdn," 2020.

[31] C. Yu, J. Lan, Z. Guo, and Y. Hu, "DROM: Optimizing the routing in software-defined networks with deep reinforcement learning," *IEEE Access*, vol. 6, pp. 64 533–64 539, 2018. [Online]. Available: https://doi.org/10.1109/access.2018.2877686

[32] P. Sun, Y. Hu, J. Lan, L. Tian, and M. Chen, "TIDE: Time-relevant deep reinforcement learning for routing optimization," *Future Generation Computer Systems*, vol. 99, pp. 401–409, Oct. 2019. [Online]. Available: https://doi.org/10.1016/j.future.2019.04.014

[33] Q. T. A. Pham, Y. Hadjadj-Aoul, and A. Outtagarts, "Deep Reinforcement Learning based QoS-aware Routing in Knowledge-defined networking," in *Qshine 2018 - 14th EAI International*

Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness, Ho Chi Minh City, Vietnam, Dec. 2018, pp. 1–13. [Online]. Available: https://hal.inria.fr/hal-01933970

[34] Y. Hu, Z. Li, J. Lan, J. Wu, and L. Yao, "EARS: Intelligence-driven experiential network architecture for automatic routing in software-defined networking," *China Communications*, vol. 17, no. 2, pp. 149–162, Feb. 2020. [Online]. Available: https://doi.org/10.23919/jcc.2020.02.013

[35] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. S. da Fonseca, "Drsir: A deep reinforcement learning approach for routing in software-defined networking," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 4807–4820, 2022.

[36] L. Zhang, Y. Lu, D. Zhang, H. Cheng, P. Dong *et al.*, "Dsoqr: Deep reinforcement learning for online qos routing in sdn-based networks," *Security and Communication Networks*, vol. 2022, 2022.

[37] C. Zhao, M. Ye, X. Xue, J. Lv, Q. Jiang, and Y. Wang, "Drl-m4mr: An intelligent multicast routing approach based on dqn deep reinforcement learning in sdn," *Physical Communication*, vol. 55, p. 101919, 2022.

[38] S. Tomovic, N. Prasad, and I. Radusinovic, "Sdn control framework for qos provisioning," in *2014 22nd Telecommunications Forum Telfor (TELFOR)*. IEEE, 2014.

[39] J. Park, J. Hwang, and K. Yeom, "NSAF: An Approach for Ensuring Application-Aware Routing Based on Network QoS of Applications in SDN," *Mobile Information Systems*, vol. 2019, 2019.

[40] J.-R. Jiang, H.-W. Huang, J.-H. Liao, and S.-Y. Chen, "Extending dijkstra's shortest path algorithm for software defined networking," in *The 16th Asia-Pacific Network Operations and Management Symposium*. IEEE, Sep. 2014. [Online]. Available: https://doi.org/10.1109/apnoms.2014.6996609

[41] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. da Fonseca, "Intelligent routing based on reinforcement learning for software-defined networking," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 870–881, 2020.

[42] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, 09 2015.

[43] G. Kim, Y. Kim, and H. Lim, "Deep reinforcement learning-based routing on software-defined networks," *IEEE Access*, vol. 10, pp. 18 121–18 133, 2022.

[44] W. Fedus, P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland, and W. Dabney, "Revisiting fundamentals of experience replay," in *International Conference on Machine Learning*. PMLR, 2020, pp. 3061–3071.

[45] H. M. El Misilmani, T. Naous, and S. K. Al Khatib, "A review on the design and optimization of antennas using machine learning algorithms and techniques," *International Journal of RF and Microwave Computer-Aided Engineering*, vol. 30, no. 10, p. e22356, 2020.

[46] E. Okewu, P. Adewole, and O. Sennaike, "Experimental comparison of stochastic optimizers in deep learning," in *Computational Science and Its Applications–ICCSA 2019: 19th International Conference, Saint Petersburg, Russia, July 1–4, 2019, Proceedings, Part V 19*. Springer, 2019, pp. 704–715.

[47] R. Zhong, Y. Liu, X. Mu, Y. Chen, X. Wang, and L. Hanzo, "Hybrid reinforcement learning for star-riss: A coupled phase-shift model based beamformer," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 9, pp. 2556–2569, 2022.

[48] L. Al Shalabi and Z. Shaaban, "Normalization as a preprocessing engine for data mining and the approach of preference matrix," in *2006 International conference on dependability of computer systems*. IEEE, 2006, pp. 207–214.

[49] L. Liao and V. C. M. Leung, "Lldp based link latency monitoring in software defined networks," in *2016 12th International Conference on Network and Service Management (CNSM)*, 2016.

[50] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. Schroeder, J. Spaans, and P. Larroy, "Linux advanced routing & traffic control," in *Ottawa Linux Symposium*, vol. 213. sn, 2002.

[51] P. T. Kirstein, "European international academic networking: A 20 year perspective." in *TERENA Networking Conference*, 2004.

[52] K. Kaur, J. Singh, and N. S. Ghumman, "Mininet as software defined networking testing platform," in *International conference on communication, computing & systems (ICCCS)*, 2014, pp. 139–42.

[53] Y. Lu and S. Zhu, "Sdn-based tcp congestion control in data center networks," in *2015 IEEE 34th international performance computing and communications conference (IPCCC)*. IEEE, 2015, pp. 1–7.

[54] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, p. 83–86, jan 2006. [Online]. Available: https://doi.org/10.1145/1111322.1111341