الجمهورية الجزَائِرية الديمقرَاطية الشعبية
PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
وزَارة التعليم العَالي و البحث العلمي
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
جَامعة عمَار ثليجي بِالأَغوَاط
UNIVERSITY OF LAGHOUAT

FACULTY OF SCIENCES
DEPARTMENT OF COMPUTER SCIENCE

Field : Mathematics and Computer Science
Option : Computer Science
Specialization : Distributed Networks, Systems, and Applications.

Dissertation Submitted in fulfillment of the requirements for the Masters degree in Computer Science

Submitted by: AMEUR Mazene

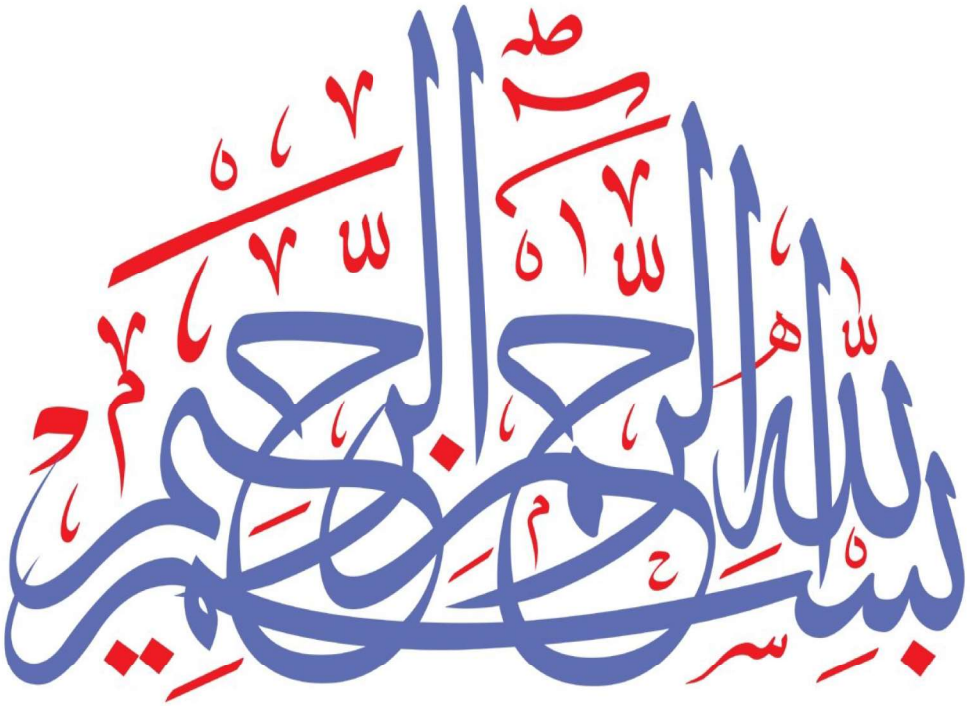
THEME

Machine Learning for Resource Allocation in Partially Observable Networks

Jury members:

Mr.	Ouinten Youcef	Professor	(University of Laghouat)	President
Mrs.	Bousbaa Fatima Zohra	M.C.(A)	(University of Laghouat)	Examiner
Mr.	Bensaad Mohamed Lahcen	M.C.(A)	(University of Laghouat)	Examiner
Mr.	Lagraa Nasreddine	Professor	(University of Laghouat)	Advisor
Mr.	Bradai Abbas	M.C.(A)	(University of Poitiers)	Co-advisor

2022-2023


بسم الله الرحمن الرحيم

Acknowledgments

First of all, I must thank the almighty Allah, who has guided me and given me the ability and the perseverance for the accomplishment of this work. All praise and thanks are due to the almighty Allah, the Lord of the World.

The Prophet Muhammad, peace and blessings be upon him, said, ***"Whoever does not thank people has not thanked Allah"****. So I would like to appreciate and express my utmost gratitude to my respected supervisor* ***Pr. Nasreddine Lagraa*** *for his endless counsel, support, and valuable guidance throughout this year.*

I would like to extend my deepest appreciation to my supervisor ***Dr. Abbas Bradai*** *for providing me with the invaluable opportunity to participate in an online internship. I am appreciative to the entire* ***Xlim Laboratory research team*** *for their guidance, expertise, and unwavering support throughout this internship.*

I would like to extend my sincere thanks and gratitude to the jury members ***Pr. Youcef Ouinten****,* ***Dr. Fatima Zohra Bousbaa****, and* ***Dr. Mohamed Lahcen Bensaad*** *who kindly accepted to allocate time and find the energy to read and evaluate my dissertation, also for their valuable support during this journey.*

Special thanks go to all professors and administrative staff of the Computer Science department at Laghouat University. Last but not least, special regards to all my friends, for their moral support.

Mazene, June 2023

Dedications

This thesis is specifically devoted to honoring the cherished memory of my beloved father, ***Professor Messaoud Ameur****, who passed away before completing this work. He was and still will be a source of inspiration for me forever. May Allah grant him paradise.*

To my dear mother, that no dedication can express my sincere feelings, for her love, support, patience, and her sacrifice throughout my studies, may the almighty Allah always protects her.

To my dear sister and brothers for their endless love and unwavering support.

Finally, to all my family and whoever cares about me.

I dedicate this humble work.

Ameur Mazene

Abstract

The rising need for dependable and efficient networks has sparked a surge of interest in implementing Machine Learning (ML) methods for Resource Allocation in Software-Defined Wide Area Networks (SD-WAN) and Partially Observable Networks (PONs) that requires reliable and high-performance communication. Meanwhile, Deep Reinforcement Learning (DRL) is a type of ML that has gained significant attention due to its ability to learn through trial and error in complex environments. One application of DRL is in the path selection optimization field where Quality of Service (QoS) requirements play an important role in influencing the path decisions. This problem is considered one of the crucial resource allocation problems in networking. To this end, this thesis proposes a Multi-Layered Hierarchical SD-WAN control architecture that can effectively deploy our Multi-Agent DRL-based Hierarchical Teacher-Students (HTS) framework. The latter tackles the QoS Path Optimisation problem by leveraging the capabilities of DRL to learn optimal path selection policies while addressing the challenges of the distributed decision-making process in SD-WAN. The experimental findings demonstrate the efficacy of the suggested HTS framework in improving network performance and reliability, outperforming some of the existing approaches. Specifically, HTS achieves better average delay, throughput ratio, and packet loss ratio results compared to two other routing schemes including Shortest Path Routing (SPR) and QoS Routing (QoSR). Additionally, to justify the important role of the teacher in our framework, our scheme achieves superior performance compared to a DRL-based Path Optimisation solution, while demonstrating excellent ability to adapt and maintain reliable performance in the face of link failures.

Keywords: Multi-agent DRL, Resource Allocation, SD-WAN, Teacher-Students Framework, QoS Path Optimisation.

ملخص

أدى الطلب المتزايد على شبكات موثوقة وعالية الأداء إلى اهتمام متزايد بتطبيق تقنيات التعلم الآلي (ML) لإدارة الموارد في الشبكات الملاحظة جزئيا (PONs). تهدف هذه الأطروحة على استخدام التعلم المعزز العميق (DRL) لاختيار مسار حزمة المعلومات مع الأخذ بعين الاعتبار جودة الخدمة (QoS)، بحيث تعتبر مشكلة من مشكلات إدارة الموارد في الشبكات الواسعة النطاق القابلة للبرمجة (SD-WAN)، وعليه تقترح هذه الأطروحة بنية هرمية متعددة الطبقات مخصصة للشبكات الواسعة النطاق القابلة للبرمجة. في حين تتمثل المساهمة الثانية في تقديم إطار عمل هرمي يسمى معلم - تلاميذ (HTS) حيث نستعمل فيه تقنيات التعلم المعزز العميق المتعدد الوكلاء (MA-DRL) لاختيارأفضل مسار لحزمة المعلومات يحقق شروط الجودة المطلوبة. يستعمل الحل المقترح مزايا التعلم المعزز العميق من أجل تعلم افضل سياسة ممكنة لاختيار الطرق التي تسلكها المعلومات عبر أجهزة التوجيه المتواجدة على الحافة. توضح النتائج التجريبية فعالية الإطار المقترح في تحسين أداء الشبكة وموثوقيتها، متفوقا على بعض من أفضل الحلول المقترحة سابقا حيث يتعلق الأمر بالخورزميتين SPR و QoSR. اضافة على ذلك، لإثبات دور المعلم في إطار عملنا، قارنا HTS مع حل يعتمد فقط على تقنية التعليم المعزز العميق بغياب الأستاذ، النتائج بينت تفوق مخططنا المقترح مع إظهار قدرة ممتازة على التكيف والحفاظ على أداء موثوق في مواجهة أعطال الروابط.

الكلمات المفتاحية: التعلم المعزز العميق المتعدد الوكلاء، إدارة الموارد، الشبكات الواسعة النطاق القابلة للبرمجة، إطار العمل معلم - تلاميذ، إختيار مسار حزمة المعلومات مع جودة الخدمة.

Résumé

Le besoin croissant de réseaux fiables et efficaces a suscité un intérêt croissant pour la mise en œuvre des méthodes d'apprentissage automatique (Machine Learning - ML) pour la gestion des ressources dans les réseaux étendus à définition logicielle (Software-Defined Wide Area Networks - SD-WAN) et les réseaux partiellement observables (Partially Observable Networks - PONs) nécessitant une communication fiable et performante. De plus, l'apprentissage par renforcement profond (Deep Reinforcement Learning - DRL) est un type de ML qui a suscité une attention significative en raison de sa capacité à apprendre par essais et erreurs dans des environnements complexes. Une application de DRL est l'optimisation de la sélection des chemins de qualité de service (Quality of Service - QoS), qui est un problème crucial de gestion des ressources en réseau. Pour résoudre ce problème, cette thèse propose principalement un framework hiérarchique d'apprentissage par renforcement profond à base de multiples agents (Hierarchical Teacher-Students - HTS) pour l'optimisation des chemins de QoS déployés dans une architecture SD-WAN hiérarchique à plusieurs niveaux. Les résultats expérimentaux démontrent l'efficacité du HTS suggéré dans l'amélioration des performances et de la fiabilité du réseau, surpassant certain approches existantes. En particulier, HTS obtient de meilleurs résultats en termes de délai, débit, et de perte de paquets par rapport à deux autres schémas de routage, notamment le routage du chemin le plus court (Shortest Path Routing - SPR) et le routage de QoS (QoS Routing - QoSR). De plus, pour justifier le rôle important de Teacher dans notre framework, notre solution atteint des performances supérieures par rapport à une solution d'optimisation de chemin basée sur DRL, tout en démontrant une excellente capacité à s'adapter et à maintenir des performances fiables en cas de défaillance de lien.

Mots clés: : **DRL Multi-Agent, Gestion des ressources, SD-WAN, Teacher-students framework, Optimisation des chemins avec QoS.**

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

AI Artificial Intelligence. 26

ANNs Artificial Neural Networks. 28

APIs Application Programming Interfaces. 13

AS Autonomous Systems. 6

CNNs Convolutional Neural Networks. 29

DRL Deep Reinforcement Learning. iv, 2, 3, 25

ETSI European Telecommunications Standards Institute. 14

FNNs Feedforward Neural Networks. 29

HTS Hierarchical Teacher-Students. iv, 3, 53, 60

IDE Integrated Development Environment. 54, 55

MAMDP Multi-Agent Markov Decision Process. 51, 52, 53, 56

MDP Markov Decision Process. 26, 29, 31

ML Machine Learning. iv, 1

MLHS Multi-Layered Hierarchical SD-WAN. 3

NBI Northbound Interface. 13

NFV Network Function Virtualization. 7, 14, 15

NFVI Network Functions Virtualisation Infrastructure. 14

NOS Network Operating System. 16

ONF Open Networking Foundation. 6, 8

POMDP Partially Observable Markov Decision Process. 26

PONs Partially Observable Networks. iv, 2

QoE Quality of Experience. 6

QoS Quality of Service. iv, 1

QoSR QoS Routing. iv, 64

RL Reinforcement Learning. 26

RNNs Recurrent Neural Networks. 29

SBI Southbound Interface. 13

SBI Westbound Interface. 13

SBI Eastbound Interface. 13

SDN Software Defined Network. 2, 3, 5, 15

SD-WAN Software-Defined Wide Area Networks. iv, 2, 3, 17

SHR Stateless Hierarchical Routing. 48, 49, 56

SPR Shortest Path Routing. iv, 64

VMs Virtual Machines. 14

VNF Virtualized Network Functions. 14

WAN Wide Area Network. 2

1 Introduction

Contents

1.1 Context

Machine Learning (ML) has become an increasingly popular approach in networking due to its ability to process large amounts of data and extract meaningful patterns and insights. Machine learning paradigms refer to the different methodologies and approaches used to develop intelligent systems capable of learning and making decisions based on data [2]. One significant and new paradigm is Deep Reinforcement Learning (DRL), the latter combines the power of deep learning and reinforcement learning to enable artificial agents to make sequential decisions in complex and dynamic environments. In the realm of DRL, an agent engages with its environment and obtains responses in the shape of rewards or penalties, contingent upon the actions it takes. The agent learns to optimize its decision-making process by utilizing deep neural networks to approximate value or policy functions [15].

Machine learning techniques have been applied in many fields and networking is not the exception. By analyzing network data, ML algorithms can learn to make predictions and recommendations, optimize resource allocation, and improve network performance. Some examples of machine learning applications in networking include traffic classification [16], anomaly detection [17], network optimization [18], and QoS management [19]. ML can also be used to automate network allocation tasks, such as network configuration, fault detection, and network security.

As the complexity of modern networks continues to grow, machine learning is becoming an essential tool for network engineers and researchers to manage and optimize network resources effectively [18]. Partially Observable Networks (PONs) are networks where some information is hidden or not available, making it challenging to make accurate decisions based on the available information. As PONs example, we have the Software-Defined Wide Area Networks (SD-WAN), when the network conditions may be uncertain or unknown, and decisions must be made based on incomplete information. SD-WAN uses Software Defined Network (SDN) principles to manage and control network traffic between geographically distributed locations. SD-WAN can provide a more flexible and cost-effective alternative to traditional WANs using SDN principles. However, the dynamic and unpredictable nature of SD-WAN means that network administrators may not have complete information about the state of the network, leading to suboptimal decisions. By using machine learning techniques such as DRL, PONs can enable SD-WAN to make optimal decisions based on incomplete information, leading to more efficient and reliable networks [20].

1.2 Problem Statement and Motivation

The problem of resource allocation in PONs is a significant challenge that needs to be addressed to optimize network performance and reliability. In PONs, some information may be hidden or not available, making it challenging to make accurate decisions based on the available information [21]. This problem is particularly acute and obvious in SD-WAN, which are becoming increasingly popular as a more flexible and cost-effective alternative to traditional Wide Area Network (WAN) [22]. However, the dynamic and unpredictable nature of SD-WANs means that network administrators may not have complete information about the state of the network, leading to suboptimal decisions. To address this problem, we propose to use machine learning techniques, particularly DRL, to enable SD-WANs to make optimal decisions based on incomplete information. In other words, in this dissertation we try to use DRL for Resource Allocation in SD-WAN and especially we address the problem of Path Selection.

The motivation for this research is to improve the performance and reliability of SD-WANs by addressing the problem of QoS Path Selection. By using DRL to make decisions based on incomplete information, we aim to optimize network performance while minimizing the impact of network congestion and other network-related issues. Furthermore, the proposed approach can enable SD-WANs to adapt to changing network conditions and learn from experience, leading to more efficient and reliable networks. The potential benefits of this research are significant, particularly for organizations that rely on SD-WANs for their day-to-day operations. Therefore, the QoS Path Selection problem is one of SD-WAN's critical areas of research that needs to be addressed to enable its widespread adoption across the globe.

1.3 Contributions

The contribution of this thesis can be summarized as follows:

- An extensive examination of the latest advancements in using DRL for path selection and routing in SDN and SD-WAN. This study aims to offer valuable insights into the current methods being used, while also identifying the existing gaps and challenges that require attention.
- A proposed control architecture called Multi-Layered Hierarchical SD-WAN (MLHS) is presented, which effectively manages network resources and enhances the Quality of Service (QoS) provided to end-users.
- A suggested framework called Multi-Agent DRL-based HIerarchical Teacher-Students is proposed for QoS path selection in SD-WAN. In this framework, the teacher represents a domain knowledge expert, and the students represent DRL agents.
- The performance of the proposed Hierarchical Teacher-Students (HTS) framework is evaluated through simulations to determine its effectiveness.

1.4 Organization of the Thesis

This dissertation is organized as shown in Figure 1.1:

- In **Chapter 2**, we provide an overview of SDN and SD-WAN, where we present their architecture, types, and key features.
- In **Chapter 3**, we explore DRL and study some of the related methods that use it for the QoS Path optimization problem in SDN generally and SD-WAN specifically.
- In **Chapter 4**, we present our new Multi-Layered Hierarchical SD-WAN architecture (Multi-Layered Hierarchical SD-WAN (MLHS)). Then, we introduce our novel framework named Hierarchical Teacher-Students (HTS) that can perfectly fit in our architecture, where we formulate our problem and provide the model details.
- In **Chapter 5**, we describe the implementation of HTS, showcase the simulation setup, and finally evaluate and analyze the performance using several scenarios.
- Finally, we provide a general summary of the findings in this study and propose potential directions for future development.

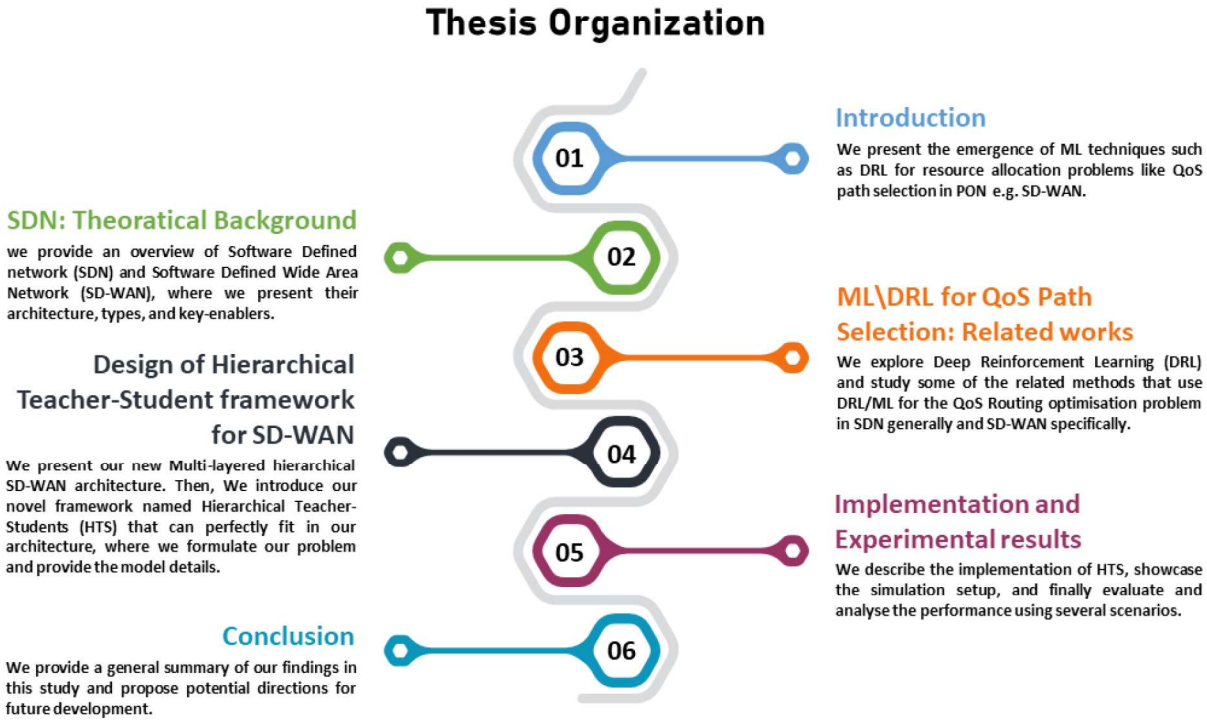


Figure 1.1: Structure of the thesis.

2

Software Defined Network: Theoretical Background

Contents

2.1 Introduction

SDN [23, 24] is a paradigm shift in the way networks are designed, deployed, and managed. It enables network administrators to programmatically control the behavior of the network by separating the control plane from the data plane, making the network more agile, flexible, and scalable [25].

This chapter discusses the theoretical side of SDN, starting from the motivation and the technology behind it, passing through its architecture and the protocols used to deploy it. Last but not least, we shed light on one of its most widely used applications which is the Software-Defined Wide Area Network (SD-WAN).

2.2 The Rise of SDN

The concept of SDN was first introduced in 2011 by the Open Networking Foundation (ONF), a consortium of industry leaders, including Google, Microsoft, and Cisco, aimed at promoting open standards and software-defined networking. Since then, SDN has gained significant momentum and has been adopted by many large organizations, including Facebook, Amazon, and AT&T [26].

According to projections, the SDN market is expected to exceed the value of 112 billion USD by 2028, experiencing a Compound Annual Growth Rate (CAGR) of 19.60% between the years 2021 and 2028. This growth is primarily attributed to the increasing investments made by the major companies towards SDN technology for billion USD in the automation of network infrastructure. As a result, the software-defined networking (SDN) market is expected to expand significantly [27].

More than 4 billion individuals are connected to the Internet through over 500,000 Autonomous Systems (AS) [28]. These numbers are increasing exponentially over the years, posing challenges to the traditional network infrastructure as these users keep asking for a better Quality of Experience (QoE) [29]. To address this challenge, SDN with the separation plan eliminates the need for traditional forwarding devices like switches and routers and instead relies on simple forwarding devices. Consequently, decision-making is centralized, which provides a comprehensive view of the network and programming abstractions. SDN offers a scalable, cost-effective, and holistically manageable networking system that features cloud abstraction and rigorous monitoring [30].

2.3 From Traditional Networking to SDN

The shift from traditional networking to Software-Defined Networking (SDN) has been a significant one in recent years. Traditional networking relied heavily on physical infrastructure, such as switches and routers, to manage network traffic and ensure connectivity. Traditionally, networking devices such as routers and switches have tightly integrated control and data planes (as shown in Figure 2.1). This means that the network devices themselves are responsible for making forwarding decisions and routing packets [13, 31].

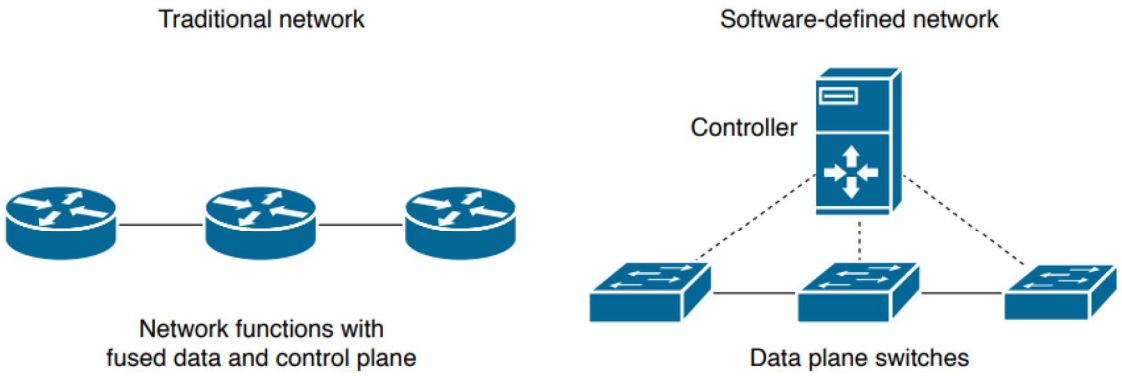


Figure 2.1: Control plane separation: Traditional network vs SDN

Traditional Networking drawbacks

While traditional networking has served its purpose, there are several drawbacks that come with it. Some of the most notable drawbacks of traditional networking [31, 12] are:

- Legacy networking devices are designed to handle a limited amount of traffic, and adding more devices to the network can lead to congestion and slow performance.
- Classic routers and switches require manual configuration and management, which can be time-consuming and complex. This can result in human errors, which can lead to network failures and security breaches.
- This networking equipment can be expensive to purchase, operate, and maintain. Additionally, as the equipment ages, it may become more expensive to maintain, leading to higher operating costs over time.
- Traditional networking devices were not designed with security in mind, and as a result, they may have vulnerabilities that can be exploited by attackers. This can lead to data breaches and other security incidents.
- Traditional routers are often inflexible and cannot adapt to changing network conditions or new technologies quickly. This can slow down innovation and makes it difficult to implement new network services or features.

Overall, traditional networking has served its purpose, but it is becoming increasingly clear that it has several drawbacks that limit its effectiveness in modern networking environments. As a result, many organizations are turning to newer networking technologies, such as SDN (see section 2.4) and Network Function Virtualization (NFV) (see section 2.7), to overcome these limitations. Table 2.3 provides more in-depth details about the main features and the differences between the two schemes [12].

Table 2.1: The Differences between Software Defined Network and Traditional Network [12].

Software Defined Network	Traditional Network
Centralized control.	Distributed control.
Programmable Network.	Non programmable Network.
Open interface.	Closed interface.
Software is utilized to separate the data plane and control plane in Software Defined Networking.	The data plane and control plane in traditional networking are located on the same plane.
Takes less time due to the automatic configuration.	Takes more time because of the static/manual configuration.
We can block and prioritize specific network packets.	All packets are treated in the same way.
It is easy to program as per need.	It is difficult to program again and to replace existing program as per use.
Cost of deployment is low.	Cost of deployment is high.
Low Structural complexity.	High Structural complexity.
Easy troubleshooting and reporting due to the centralized control.	Difficult troubleshooting and reporting due to the distributed control.
Lower maintenance cost.	Higher maintenance cost.

2.4 SDN Definition

The Open Networking Foundation ONF has defined SDN as the separation of the network's control plane from the forwarding plane, with a control plane governing multiple devices [32]. In SDN, the control plane is abstracted and managed by a centralized software controller, which can dynamically configure and manage the network devices through an open and programmable interface named OpenFlow (see section 2.4). This enables greater network flexibility, scalability, and agility, as well as easier automation and management of network resources [33].

OpenFlow

OpenFlow is the initial standardized communication interface that connects the control and forwarding layers in an SDN structure. It permits the direct control and modification of the forwarding plane in various network devices, including both virtual (hypervisor-based) and physical switches and routers. On both ends of the interface connecting network infrastructure devices and SDN control software, the OpenFlow protocol is put into action. This protocol utilizes the notion of "flows" to recognize network traffic by utilizing match rules that are predefined and can be statically or dynamically programmed by the SDN control software [34].

OpenFlow Switch is comprised of a group table and one or more flow tables that carry out packet lookups and forwarding, as well as an OpenFlow channel that links it to an external controller (see Figure 2.2). By means of the OpenFlow protocol, the switch interacts with the

controller, which oversees the switch's operations [1].

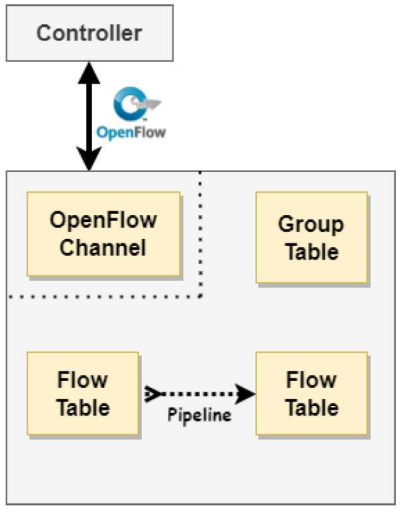


Figure 2.2: OpenFlow switch Key components [1].

In order to gain an understanding of SDN architecture, it is essential to review its fundamental operation. As illustrated in Figure 2.3, OpenFlow-based SDN networks operate by utilizing an OpenFlow protocol to facilitate communication between switches and the software-based controller. Each OpenFlow switch has a flow table, which consists of flow entries that determine how different packets on the data plane are processed. When a packet arrives on the data plane, the OpenFlow switch extracts its header fields and matches them against flow entries in the flow table. If a matching entry is found, the packet is processed according to the actions specified in the flow entry. If there is no matching entry, the switch forwards an OpenFlow PacketIn message (arrows 2 and 5) to the controller, which includes the packet header or the entire packet, if specified. The controller then sends OpenFlow FlowMod messages (arrows 3 and 6) to manage the flow table of the switch by adding flow entries, which can be used to process subsequent packets of the flow. The OpenFlow protocol standardizes the messages transmitted between the switches and the controller [2].

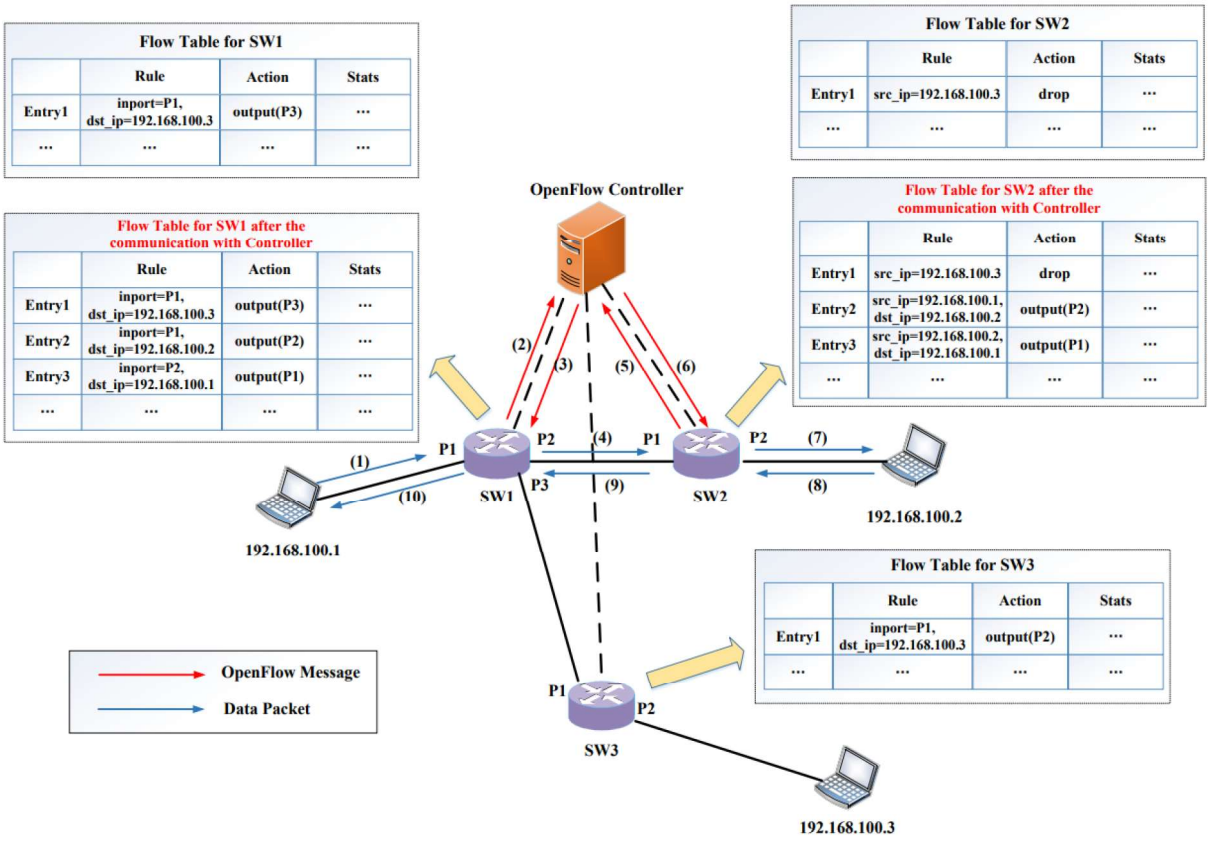


Figure 2.3: OpenFlow-enabled SDN network [2].

The flow tables in switches can be modified by the OpenFlow controller to manage traffic forwarding. By adding two flow entries (Entry2 and Entry3) at switches SW1 and SW2, communication between IP addresses 192.168.100.1 and 192.168.100.2 can be allowed. However, packets from IP address 192.168.100.3 to 192.168.100.2 are denied at switch SW2 due to security policies.

2.5 SDN Key features

It is clear that the demand for a new network architecture implies that the previous technologies and standards were inadequate to fulfill current requirements. The current predicament is that modern technology has introduced new innovations, such as social media, mobile devices, VoIP, and cloud computing, but the networks linking them are restricted by their limitations. The issue that arises is how to utilize and further develop these new technologies and innovations without being constrained by network limitations [34].

In recent times, the use of techniques like virtualization and cloud technology has made updating or modifying software more convenient. However, there are instances where this may not be the case. Additionally, the hardware aspect of the network can pose a significant

challenge. There are situations where altering or replacing parts of the network or outdated hardware might not be achievable without interfering with the infrastructure, which could prove costly and potentially hazardous. This indicates that networks are still constrained in certain aspects such as upgrading, managing, and maintaining them. Consequently, this has resulted in a dearth of advancement and progression in network technologies, as was evident in recent years. SDN can potentially address and prevent the issues outlined earlier by providing a simple way to virtualize and manage various network devices. SDN has the capability to introduce numerous innovations to the network, including [35]:

- **Virtualization:** The utilization of network resources from diverse environments and integrating them in a software-driven network;
- **Orchestration:** The capacity to operate, manage, and regulate a wide range of devices through uncomplicated commands. usually referred to as Network OS (see section 2.7.3);
- **Programmability:** The capability of modifying the actions of a device or a specified segment of the network;
- **Dynamic Scaling:** The ability to modify the size, quality, performance, etc. of the network as required;
- **Automation:** Reducing human intervention to a minimum for a more efficient system;
- **Visibility:** Supervision of resources, connectivity, network state, network topology, etc.
- **Performance:** Improving the network's resource utilization by load balancing and quick failure resolution, and maximizing the utilization of all network resources;
- **Service integration:** The ability to place services, such as firewalls and Intrusion Detection Systems (IDS), dynamically.

2.6 SDN Architecture

With SDN, the control plane is abstracted from the data plane and is implemented in a software controller that communicates with the network devices through the previously mentioned open interface (OpenFlow) [36]. This decoupling of the control and data planes allows network administrators to configure the network centrally and dynamically, based on the network's current state and traffic patterns [18].

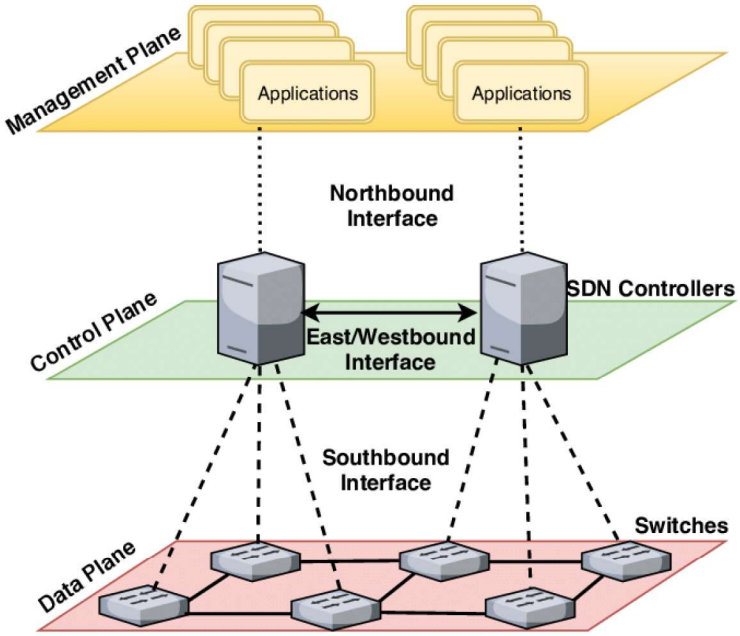


Figure 2.4: SDN Architecture.

As Figure 2.4 illustrates the SDN architecture comprises three main components or planes: Application layer, Control layer, and Data layer. These layers are separated by interfaces (see section 2.6.4) which abstract and help transfer information, data, and commands [34]:

2.6.1 Application Plane

SDN Applications refer to software programs that communicate their network requirements and preferred network behavior to the SDN Controller using a northbound interface (NBI) in a direct, explicit, and programmable manner. Generally, this layer contains the software applications that manage network policies and services, such as firewalls, load balancers, and virtual private networks.

2.6.2 Control Plane

The SDN Controller is a virtual entity that receives directives or demands from the SDN Application layer and transmits them to the network components. Furthermore, it obtains network-related details from the hardware devices and sends back to the SDN Applications a simplified representation of the network that includes information, such as statistics and occurrences of ongoing activities.

2.6.3 Data Plane

The SDN Data layer comprises the network devices, such as switches and routers, that handle the forwarding of data packets.

2.6.4 SDN Interfaces

One of the key components of the SDN architecture is the set of interfaces used to control and manage the network. These interfaces enable network administrators to program and manage the network using software, rather than manually configuring each device. As shown in Figure 2.4, SDN defines the following interfaces: Northbound, Southbound, Eastbound, and Westbound [37].

Northbound Interface

Northbound Interface (NBI) enables the communication between the control plane and the applications or services running on top of it. The NBI provides a standardized set of Application Programming Interfaces (APIs) and protocols that allow applications and services to interact with the network, making it easier to develop and deploy new network services. Commonly used NBI APIs include Ad-hoc API and REST-ful API [38]. With a software-based NBI, network administrators can leverage the power of software to manage the network, automate tasks, and enable faster service delivery. The NBI also facilitates the integration of different networking solutions and helps ensure interoperability between different vendors products, making it a critical component of modern network management [39].

Southbound Interface

The Southbound Interface (SBI), unlike the NBI, has a clearly defined and commonly accepted standardized structure. It consists of physical or virtual devices, such as Open vSwitch, and APIs like OpenFlow, NetConf, and SNMP [40]. Its purpose is to specify the communication protocol between forwarding devices, such as SDN switches, and their controllers. This includes defining the instruction sets of the forwarding devices. The SBI's main goal is to transmit the execution order from the management plane to the data plane using a standard protocol [37].

Eastbound-Westbound Interfaces

The Eastbound and Westbound interfaces are two scalability-related special case interfaces. Westbound Interface (SBI) connects the SDN to the traditional network. The flow table is converted into a classical routing path using WBI, and vice versa. The scalability of SDN involves the connection of many controllers, which requires an inter-controller communication protocol. Several SDN controllers are linked together via the Eastbound Interface (SBI) [41].

2.7 Network Function Virtualization (NFV)

Network Function Virtualization enables the virtualization of network services, such as routers, firewalls, and load balancers, that were traditionally operated on specialized hardware. These services are presented as Virtual Machines (VMs) on general-purpose hardware, enabling network providers to use standard servers instead of proprietary ones to run their network [42].

2.7.1 NFV Framework

European Telecommunications Standards Institute (ETSI) has created a framework for NFV architecture (shown in Figure 2.5), which allows to Virtualized Network Functions (VNF) to be deployed and executed on a Network Functions Virtualisation Infrastructure (NFVI). This infrastructure is comprised of standard servers, accompanied by a software layer that provides logical abstraction and partitioning for them. In the NFVI, a VNF is typically assigned to one virtual machine (VM) above the hypervisor layer. The deployment, execution, and operation of VNFs on the NFVI are managed by a Management and Orchestration (M&O) system. This system is guided by metadata describing the network services and their VNFs. The M&O system comprises an NFV Orchestrator that manages the lifecycle of network services, a set of VNF managers that handle the lifecycle of VNFs, and a virtualized infrastructure manager that functions as an extended cloud management system, responsible for controlling and managing NFVI resources [43].

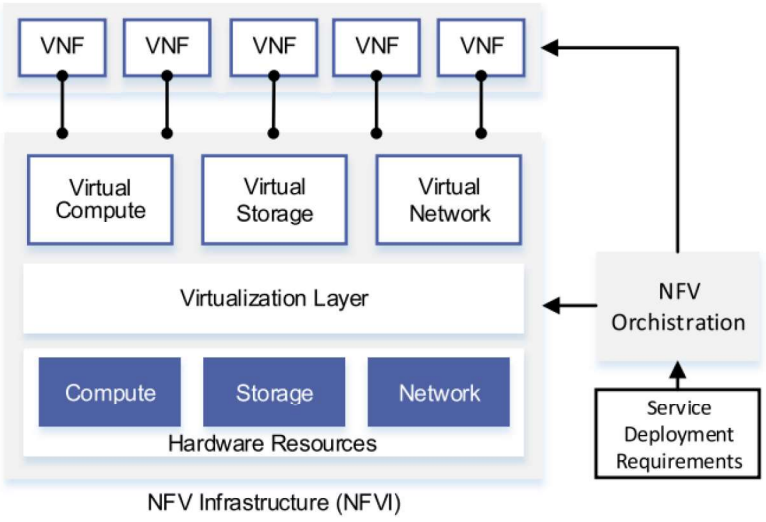


Figure 2.5: NFV framework Illustration.

2.7.2 NFV vs. SDN

NFV and SDN are related and can operate extremely well together (Figure 2.6). NFV can benefit SDN by virtualizing the SDN controller, which can then run on the cloud. This enables controllers to be dynamically relocated to their optimal positions. SDN, on the other hand, may help NFV by enabling customizable network connections between VNFs. This allows for better traffic engineering and traffic steering between virtualized network operations [44]. However, NFV and SDN are distinct from each other with regard to their concepts, system architecture, and functions. These differences can be summarized as follows [43]:

- NFV involves implementing network functions using software, while SDN involves creating a programmable network architecture that is centrally controlled to improve connectivity.
- NFV's goals include reducing expenses related to capital (CapEx), operating costs (OpEx), space, and power consumption. Meanwhile, SDN focuses on creating network abstractions that allow for flexible network control, configuration, and rapid innovation.
- NFV achieves agile provisioning and deployment by separating network functions from proprietary hardware. On the other hand, SDN separates the network control plane from the data plane forwarding, creating a centralized controller that enables programmability.

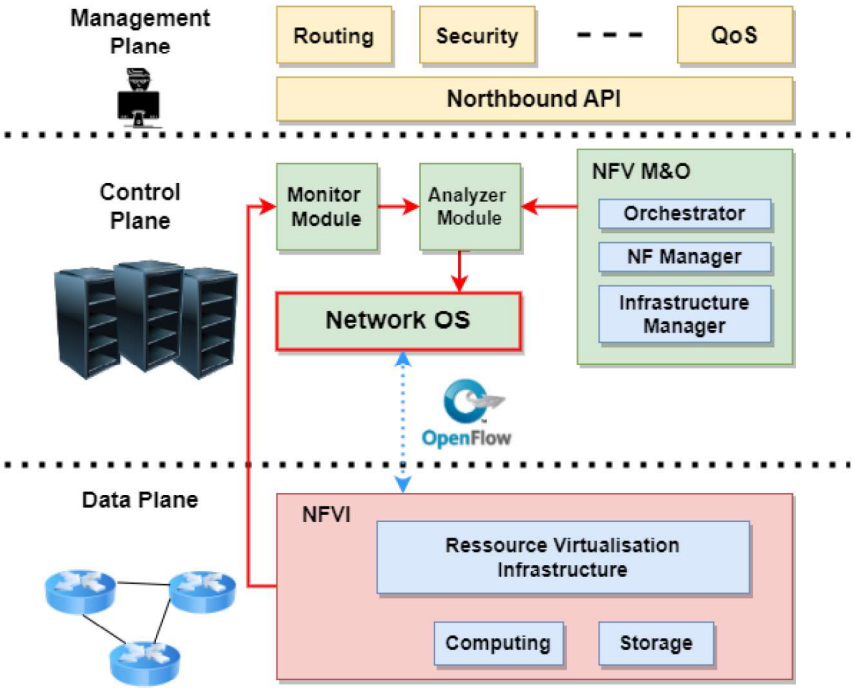


Figure 2.6: SDN/NFV Architecture [3].

2.7.3 Network Operating System (NOS)

A Network Operating System (NOS) is a core element of SDN architecture, it is used to manage the SDN controller and the network devices, such as switches and routers. The NOS is responsible for managing the flow of traffic, configuring network devices, collecting and analyzing network data for use in decision-making. It also enables network administrators to program and automate network tasks and services through the use of APIs and other interfaces (Figure 2.7).

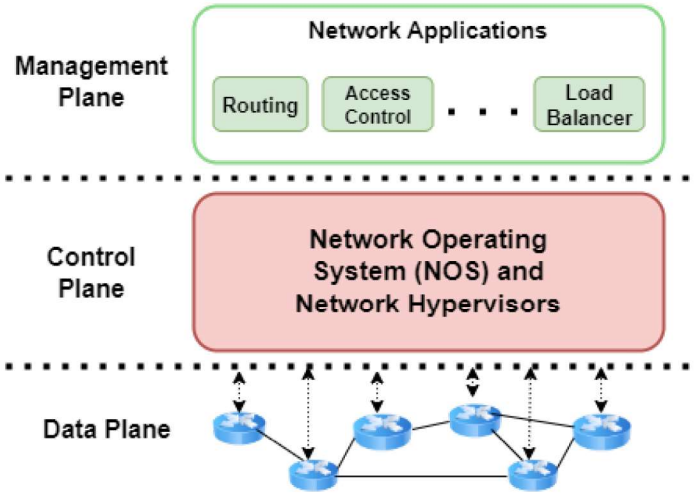


Figure 2.7: Network Operating System Overview [4].

Some of the most common NOS are mentioned in table 2.2 and are implemented in C, C++, Python, Java, and Ruby [41, 13].

Table 2.2: Some of the widely used NOS in SDN [13].

NOS	Architecture	Programming Language
Beacon	Centralized	Java
HyperFlow	Distributed	C++
Kandoo	Distributed	C, C++, Python
NOX	Centralized	C++
OpenDaylight	Distributed	Java
Floodlight	Centralized	Java
ONOS	Distributed	Java
POX	Centralized	Python
Onix	Distributed	Python, C
Ryu	Centralized	Python

2.8 Software Defined Anything (SDX)

SDN has been widely applied in several applications due to its ability to simplify network management, increase network agility, and reduce network complexity and costs. Due to the tremendous influence that software has on numerous domains such as hardware production, IT infrastructure, and industrial applications, the notion of "Software-Defined Anything" (SDX) has emerged. Its effect is vast and significant, revealing software's ability to shape and define several areas of modern technology [45]. The table below 2.3 illustrates some of these applications:

Table 2.3: Software Defied Anything (SDX).

SDN Type	Definition
SDDC [46]	SDDC stands for Software-Defined Data Center, which is a data center architecture that virtualizes and abstracts computation, networking, and storage resources to provide more agility, flexibility, and scalability.
SDS [47]	SDS stands for Software-Defined Storage, which is a storage architecture that separates the storage hardware from the software that manages it to create a more flexible and scalable storage infrastructure
SDI [48]	SDI stands for Software-Defined Infrastructure, which is an infrastructure architecture that virtualizes and abstracts hardware resources to enable more flexible and efficient management of compute, storage, and networking resources.
SDWAN [49]	SD-WAN stands for Software-Defined Wide Area Network, which is a networking technology that uses software-defined networking to provide more flexible, scalable, and cost-effective connectivity for enterprise networks over a wide geographic area.

In our thesis, we study one of the most widely used applications of SDN in an interesting field which is the Wide Area Network (WAN).

2.9 Software Define Wide Area Network (SD-WAN)

SD-WAN is a relatively new approach to network connectivity that is rapidly gaining popularity in the enterprise world. With the growth of cloud-based applications and the need for more flexible and scalable network solutions, traditional WANs have become increasingly outdated and inflexible. SD-WAN technology is designed to provide a more intelligent and dynamic way of connecting remote locations and users to a central network (see Figure 2.8), while also reducing costs and improving security.

The market currently has approximately 30 vendors with differing levels of product maturity. Notably, VeloCloud (owned by VMware), Viptela (owned by Cisco), Nuage (owned by Nokia), and Silver Peak are among the vendors with more mature product offerings. The SD-WAN market is expected to exceed 17 billion dollars by 2025. An example of an SD-WAN solution is provided by VeloCloud, which offers a cloud-based solution that provides a wide

range of integrated capabilities as a virtual network function, rather than relying on a traditional hardware-centric model. These capabilities include network overlay control, dynamic path selection, application performance monitoring, and other services [22].

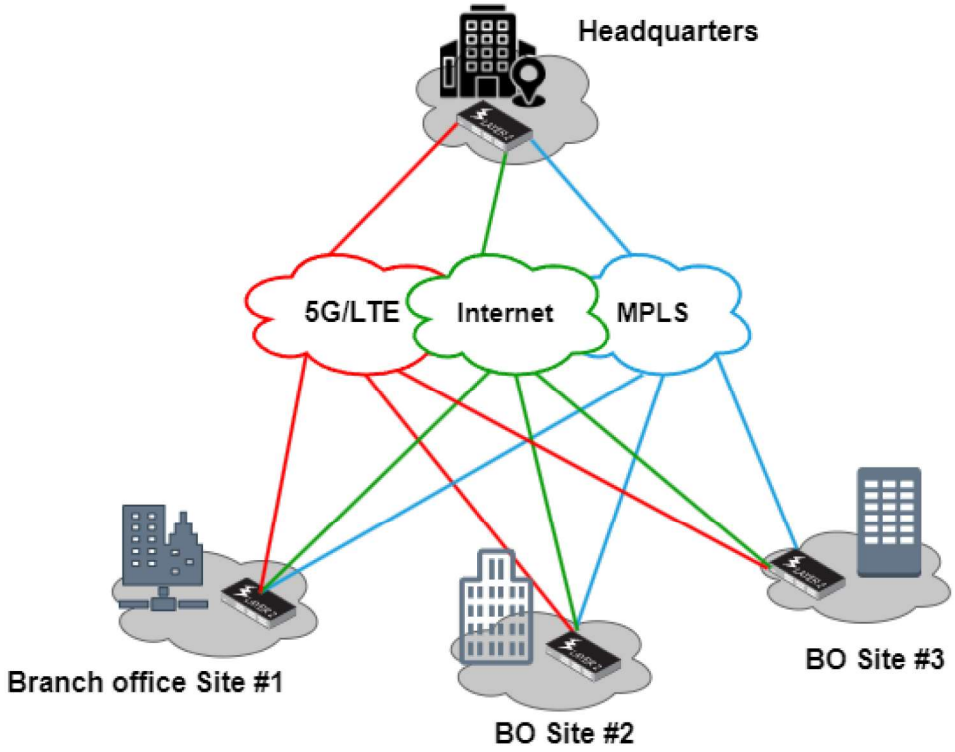


Figure 2.8: Typical SD-WAN deployment architecture [5].

2.9.1 Definition

SD-WAN is a networking technology that applies the principles of software-defined networking (SDN) to manage and enhance the performance of wide-Area networks (WANs). By utilizing SDN principles, it enables secure connectivity of users, applications, and data across multiple locations, resulting in improved performance, reliability, and scalability. Moreover, it simplifies WAN management by centralizing control and providing network-wide visibility [50].

2.9.2 MPLS vs VPN vs SD-WAN

A Wide Area Network (WAN) is a communication network that links together numerous Local Area Networks (LANs) spread across various geographical regions. Companies employ WANs to connect their various branch locations to their main headquarters, as well as to access cloud services provided by a cloud computing provider. As Enterprise Networking (EN) and Information Technologies (IT) continue to advance at a rapid pace, there is an ongoing increase in demand for WANs with higher capacities and better quality of Service (QoS) [51].

In the early 1980s, the initial public WANs were established and were subject to ongoing improvements. Initially, these wide-area connections were created to connect two distant locations by utilizing expensive leased lines that had restricted speeds. As a result, several alternative technologies were subsequently suggested to allow companies to rent capacity from public network infrastructure providers in order to establish their own inter-site connections [49].

The primary network technologies that have been implemented include Frame Relay (FR), Multi-Protocol Label Switching (MPLS), and Virtual Private Networks (VPN) [52]. Out of these options, MPLS was the most commonly used technology, and it was considered the milestone in EN as it offers complete mesh connectivity, excellent dependability, and isolation of confidential data transmissions from other IP traffic. MPLS achieves QoS by utilizing label-based forwarding instead of the traditional hop-by-hop IP routing, which transforms connectionless IP networks to connection-oriented ones. Even though MPLS ensures QoS, it does bring certain issues, such as [53]:

- The cost of MPLS for enterprise users is relatively expensive, meaning that it represents the most expensive option in terms of balancing cost and QoS;
- It is not feasible to have zero-touch deployments due to the need for the individual configuration of each device, which creates a significant amount of configuration overhead;
- The duration of the transition or upgrade process is directly proportional to the number of branch offices and edge devices. The exorbitant cost of MPLS is driving businesses to opt for internet or broadband services.

Conversely, at the opposite end of the cost vs QoS trade-off spectrum (Figure 2.9), we have VPN technology which is the most affordable in terms of cost, but the latter does not propose QoS guarantee.

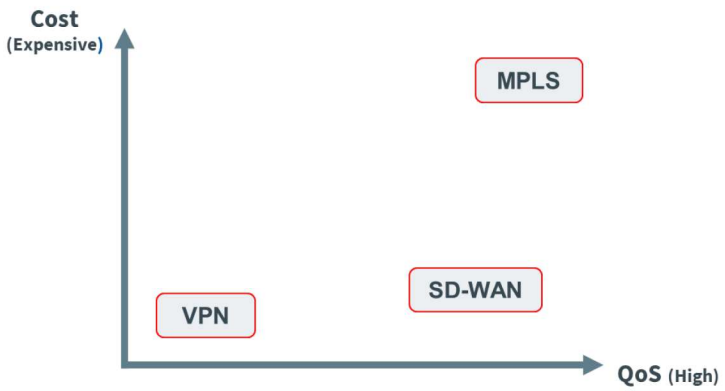


Figure 2.9: Trade-off between Cost and Quality of Service for famous network technologies.

2.9.3 Difference between Traditional WAN and SD-WAN

Traditional WANs use hardware-based routers and switches to manage and direct network traffic(see Figure 2.10), while SD-WAN uses software to centrally manage network traffic. This means that traditional WANs can be complex to manage and configure, while SD-WAN offers simplified management and automation, which makes it easier to deploy and manage.

In terms of cost, traditional WANs can be expensive due to the cost of hardware, maintenance, and dedicated leased lines, while SD-WAN can reduce costs by leveraging low-cost broadband Internet links as well as providing automated network management and reducing network downtime. When it comes to performance, traditional WANs can suffer from performance issues due to network congestion, while SD-WAN offers more efficient and flexible path selection and can dynamically adjust network paths to optimize performance. Security is another area where traditional WANs and SD-WAN differs. Traditional WANs may have limited built-in security features, while SD-WAN provides advanced security features such as encrypted traffic and built-in firewalls to protect against cyber threats [32].

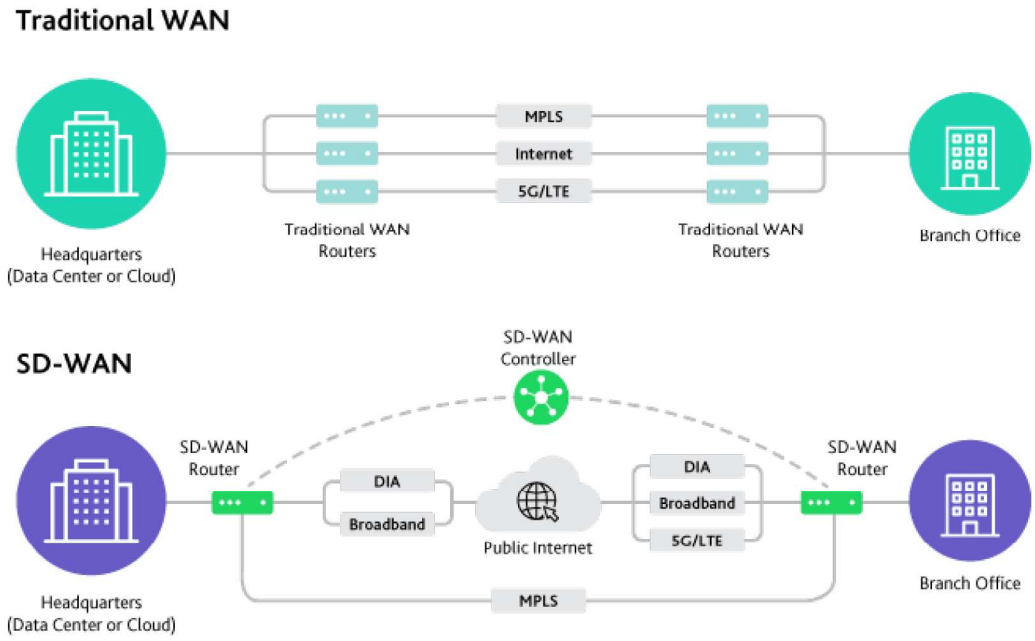


Figure 2.10: Difference between Traditional WAN and SD-WAN [6].

2.9.4 Architecture of SD-WAN

The rigidity of conventional WAN architecture fails to fulfill the Quality of Experience (QoE) requirements of modern businesses. SD-WAN aims to enhance WAN networking operations by introducing innovation and adaptability compared to older technologies. In this section, we present an SD-WAN structure illustrated in Figure 2.11, where the SD-WAN controller is

deployed at the headquarters (HQ). The architecture consists of three layers: Orchestration (or Application), Control, and Data layer [52].

Orchestration Plane

The Orchestration Plane offers application-level awareness, enabling SD-WAN to optimize network traffic based on the specific requirements of each application. This layer also includes mechanisms for visibility and control, empowering administrators to prioritize critical applications and ensure they receive the necessary bandwidth and resources.

Control Plane

The Control Plane handles policy definition and enforcement, network orchestration, and automation. It facilitates dynamic path selection, allowing SD-WAN to dynamically choose the optimal network path for each application based on network conditions and performance requirements. Moreover, the Control Plane provides security and encryption mechanisms to safeguard against cyber threats.

Data Plane

The Data Plane manages the forwarding of network traffic, encompassing packet forwarding, encryption and decryption, and Quality of Service (QoS) enforcement. This layer incorporates SD-WAN edge devices responsible for traffic forwarding and enforcing policies defined by the Control Plane.

The layered architecture of SD-WAN presents a flexible and scalable framework for managing and optimizing wide area networks. By segregating the application, control, and data planes, SD-WAN enables centralized administration of network traffic and policies, along with improved security, performance, and reliability [54].

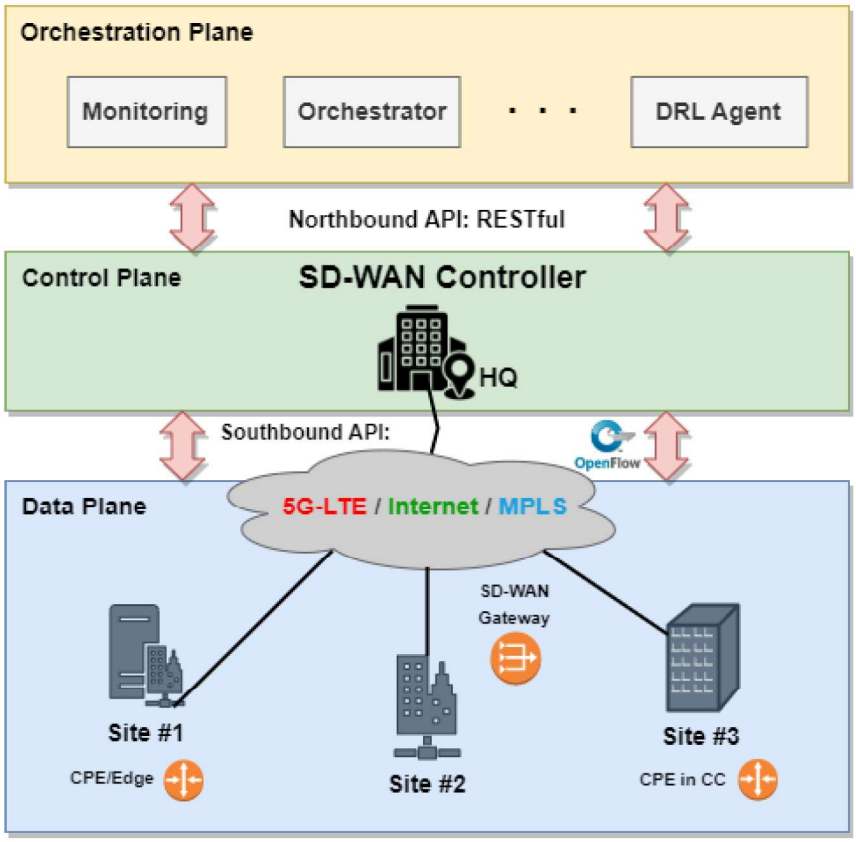


Figure 2.11: SD-WAN Architecture.

2.9.5 Functionality

SD-WAN works by utilizing software-defined networking (SDN) principles to centralize and automate the management of wide area networks (WANs). Here is a step-by-step breakdown of how SD-WAN works [52]:

1. Traffic enters the SD-WAN edge device, which acts as a gateway between the local area network (LAN) and the wide area network (WAN).
2. The traffic is inspected and classified at the application layer to determine its priority and required Quality of Service (QoS) level.
3. The SD-WAN edge device applies policy-based routing to determine the best path for the traffic based on network conditions, such as link quality, latency, and congestion.
4. The traffic is encrypted and sent across the selected path to its destination.
5. The SD-WAN edge device at the destination reassembles and decrypts the traffic and forwards it to the LAN.

6. The SD-WAN continuously monitors the network conditions and adjusts the traffic flow dynamically, re-routing traffic if necessary to ensure optimal performance.

7. The SD-WAN provides centralized management and control over the entire network, enabling administrators to set policies, monitor performance, and make changes in real-time.

2.9.6 SD-WAN Benefits

SD-WAN offers several benefits over traditional WAN solutions such as:

1. SD-WAN technology prioritizes strong connections with comprehensive security, enabling users to establish a unified network infrastructure that supports various connections such as MPLS, wireless LTE, or broadband, and allocate critical applications to strong connections and less critical data to public connections.

2. SD-WAN is flexible enough to utilize various local internet providers for bandwidth access and quickly adds new sites, eliminating provisioning that may be as a threat to the enterprises.

3. Both links and speed are equally important for efficient data transfer, to ensure maximum security, and cloud-based application traffic uses direct internet connectivity to achieve a streamlined and consistent performance in the cloud.

4. SD-WAN offers a policy-based, centralized management interface for traffic management and monitoring, simplifying complexity and enabling easy management from a single location, while also tracking application performance [50].

2.9.7 Challenges and open issues in SD-WAN

While SD-WAN undoubtedly offers numerous advantages, it is essential to acknowledge the various challenges that must be addressed, including the following [55]:

1. **Multi-cloud connectivity:** As more organizations move to the cloud, connecting to multiple cloud providers and maintaining consistent performance across different cloud environments can be a challenge for SD-WAN.

2. **Network visibility:** SD-WAN relies on real-time network data to optimize network traffic and ensure consistent performance. However, network visibility can be limited, making it difficult to diagnose and troubleshoot issues in real time.

3. **Integration with existing infrastructure:** Integrating SD-WAN with the existing networking infrastructure, such as routers and firewalls, can be challenging. SD-WAN

must be compatible with existing infrastructure and able to work seamlessly with other network devices.

4. **Network security:** Network security is a major concern for SD-WAN, as traffic is routed over the internet and may be vulnerable to cyber threats. SD-WAN must have strong security measures in place, including encryption, firewalls, and intrusion detection, to protect against these threats.

In this work, we tackle one of the major problems in SD-WAN which is the Path Optimisation Problem.

QoS Path Optimization in SD-WAN

QoS path selection is a critical component of SD-WAN that allows organizations to prioritize network traffic based on application requirements. However, there are still challenges with configuring QoS routing to meet the specific needs of an organization, such as determining which applications require the highest priority and how much bandwidth they need. In addition, QoS path selection policies must be flexible enough to adapt to changing network conditions and application requirements. Ensuring consistent QoS routing performance across different network environments and under varying network conditions is also an ongoing challenge for SD-WAN. To address these challenges, SD-WAN vendors are developing more advanced algorithms that can adapt to real-time network conditions and more customizable QoS policies to meet the specific needs of different organizations [56].

2.10 Conclusion

In this chapter, we have presented the theoretical aspect of SDN, the paradigm shift and its huge impact and reflections on the industry. After that, we studied the architecture of SDN, where we introduced its key technologies. Lastly, by illustrating the wide adaptation and applicability of SDN in different domains, we presented one of the most used SDN applications which is SD-WAN.

In the next chapter, we will present Deep Reinforcement Learning (DRL), then we will study the Quality of Service (QoS) Path Optimisation problem.

3

DRL for QoS Path Optimisation in SD-WAN: Background and Related works

Contents

3.1 Introduction

In recent years, DRL has emerged as a promising paradigm tackling different routing optimization problems. DRL is a machine learning technique that enables agents to learn optimal policies by trial-and-error interactions with an environment. By integrating deep neural networks and reinforcement learning, DRL can handle high-dimensional state and action spaces and learn from experience without explicit instructions.

In this chapter, we explore the applications of DRL for routing optimization. We start by discussing the challenges of applying DRL to routing optimization, such as the curse of

dimensionality. Then, we introduce the basic concepts of reinforcement learning, including the Markov Decision Process (MDP), Partially Observable Markov Decision Process (POMDP), and the Actor-critic methods. Finally, we review the existing literature on DRL-based routing optimization in SDN and SD-WAN, including the approaches for traffic engineering, and network optimization.

3.2 Deep Reinforcement Learning (DRL)

The primary objective of the Artificial Intelligence (AI) field is to create autonomous agents that can learn optimal behaviors through trial and error by interacting with their environments, and continually improve over time. This has been a significant challenge in the development of AI systems, whether it be physical robots that can perceive and react to their surroundings, or purely software-based agents that can understand natural language and multimedia.

Reinforcement Learning (RL) provides a mathematical framework for autonomous learning based on experience. Although RL has seen some successes in the past, it has been limited by scalability issues and restricted to low-dimensional problems. These limitations are similar to those faced by other algorithms such as memory, computational, and sample complexity [57]. However, the emergence of deep learning in recent years has provided new tools to overcome these challenges, relying on the representation learning and powerful function approximation properties of deep neural networks [58].

The emergence of deep learning has had a significant impact on various areas of machine learning, including object detection, speech recognition, and language translation [59], by considerably advancing the state of the art in these tasks. The most important feature of deep learning is its ability to automatically discover compact, low-dimensional representations or features of high-dimensional data, such as images, text, and audio. By incorporating inductive biases into neural network architectures, especially those of hierarchical representations, machine learning practitioners have made significant progress combating the curse of dimensionality [60]. Similarly, deep learning has also accelerated the progress of RL, with the integration of deep learning algorithms into RL resulting in the creation of the subfield of DRL (see Figure 3.1) [58].

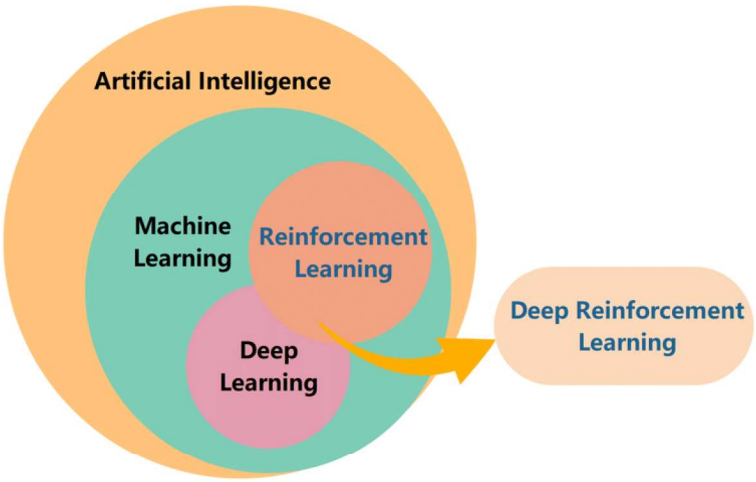


Figure 3.1: The interconnection between AI, ML, RL, DL and DRL [7].

3.2.1 Reinforcement Learning (RL)

Reinforcement learning is a subset of machine learning in which the learning process takes place through interaction with an environment. It involves goal-oriented learning, where the agent is not instructed on what actions to take, but instead learns from the outcomes of its actions. RL is a dynamic and rapidly growing field of research within artificial intelligence, with a wide range of algorithms that have been developed to tackle various problems. Consequently, it is currently one of the most active and evolving areas of AI research [9].

3.2.2 Elements of RL

Reinforcement learning (RL) involves several key elements, including [61]:

- **Agent:** The learner or decision-maker is responsible for making intelligent decisions, and they operate as learners in the field of RL. These agents interact with their environment by taking action, and they receive rewards based on the actions they take.
- **Environment:** The external system or process that the agent interacts with, and which is impacted by the agent's actions.
- **State:** The current condition of the environment, which is observed by the agent before it takes an action.
- **Action:** The decision made by the agent at a given state, which impacts the environment.
- **Reward:** A scalar signal given to the agent based on its action, indicating how well or poorly it performed at a given state. Typically, it is represented by v(s) and is equal

to the total expected reward that the agent will receive starting from the initial state. There can be multiple value functions, but the optimal value function is the one with the highest value for all states compared to others. Correspondingly, an optimal policy is the one that has the optimal value function.

- **Policy:** A mapping between states and actions, which determines what action the agent takes at each state. The behavior of an agent in an environment is determined by a policy. How the agent chooses which action to take is determined by the policy, which is usually represented by the symbol π. The policy can take different forms, such as a lookup table, a complex search process, etc.
- **Value function:** A measure of how good a state or an action is, which can be used to evaluate and improve the policy.
- **Model:** A representation of the environment that the agent can use to simulate and plan its actions.

In a Reinforcement Learning process, an agent can discover its best decision-making strategy, represented by the optimal policy π^*, by interacting with its environment. At each time step t, the agent observes the state of the environment s_t, chooses an action a_t, and receives an immediate reward r_{t+1} and the next state s_{t+1}, as shown in Figure 3.2. Based on this information, the agent adjusts its decision-making policy π. This iterative process continues until the agent's policy gradually gets closer to the optimal policy π^*, meaning $\pi \rightarrow \pi^*$[62].

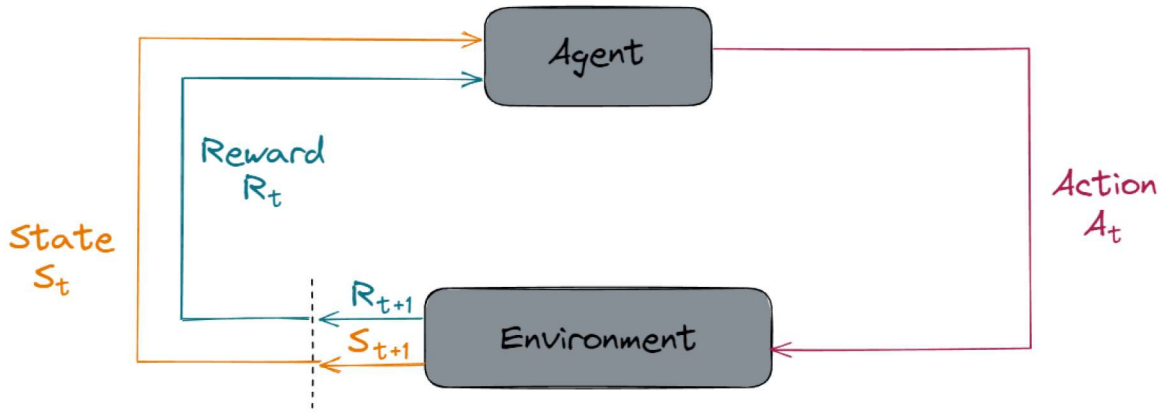


Figure 3.2: Overview of RL process [8].

3.2.3 Deep Learning (DL)

Deep Learning (DL) is highly regarded in various fields, and its achievements largely stem from the utilization of Artificial Neural Networks (ANNs) [14]. ANNs have become a conventional technique for representing data, and they consist of a group of interconnected nodes that

imitate the operations of the human brain. Each node has a weighted connection to numerous nodes in adjacent layers. The nodes take input from connected nodes and compute output values using weights and a simple function. ANNs, particularly Deep Neural Networks (DNNs), have become appealing inductive methods due to their exceptional flexibility, non-linearity, and data-based model construction [63].

Table 3.1 shows that there are three main types of neural networks: Feedforward Neural Networks (FNNs), Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs). FNNs follow a unidirectional flow of information from input to output, making them suitable for tasks like classification and regression. While CNNs are particularly well-suited for visual tasks like recognizing patterns in images and videos by taking advantage of spatial properties. On the other hand, RNNs are highly effective in classifying temporal relationships in data, making them superior for time-series tasks.

Table 3.1: Comparison between Neural networks [14].

Neural Network Type	Primary Use	Key Feature
FNNs	Classification and Regression tasks.	Ability to learn complex patterns and make predictions by processing information in a forward direction.
CNNs	Visual tasks such as image and video recognition.	Exploitation of fundamental spatial properties of images and videos.
RNNs	Time-series tasks such as speech recognition and natural language processing.	Ability to recognize temporal correlations in data.

3.2.4 Markov Decision Processes (MDP)

MDP is an expansion of the Markov chain, which provides a mathematical structure for representing decision-making scenarios. Practically all problems in the field of Reinforcement Learning can be modeled using MDP.

The concept of the Markov property implies that the future of a process is solely dependent on the present observation, and the agent has no need to consider the complete history. An MDP, which was introduced by Bellman in 1957 [64] is a type of stochastic control process that operates in discrete time. RL is defined as an MDP which consists of [9]:

- S is the state space, in which the agent can actually be in;
- A is the action space, that can be performed by an agent, for moving from one state to another;
- $T : S \times A \times S \rightarrow [0, 1]$ is the transition function (set of transition probabilities of moving from one state to another state by performing some action);

- $R : S \times A \times S \rightarrow R$ is the reward function, where R is a continuous set of possible rewards in a range $R_{max} \in \mathbb{R}^+$;
- $\gamma \in [0, 1)$ is the discount factor.

In an MDP, the system is considered to be fully observable, which indicates that the observation is equivalent to the environment's state (i.e., $\omega_t = s_t$). At each time step (t), the likelihood of moving to s_{t+1} is determined by the state transition function $T(s_t, a_t, s_{t+1})$, while the reward is determined by a bounded reward function $R(s_t, a_t, s_{t+1}) \in \mathbb{R}^+$. These concepts are depicted in Figure 3.3.

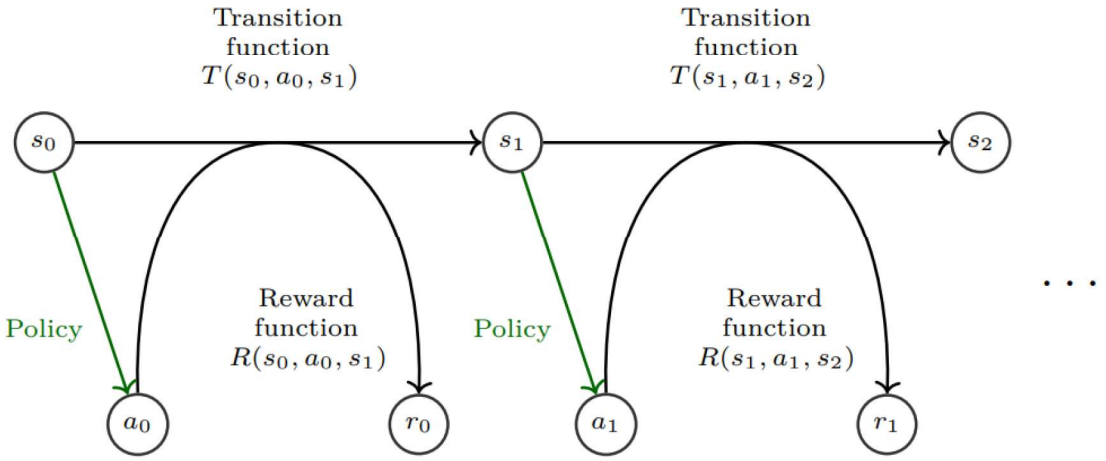


Figure 3.3: Illustration of Markov Decision Process [9].

Overall, policy π refers to a function that assigns probabilities to actions given a state, $\pi : S \rightarrow p(A = a \mid S)$. When dealing with an episodic Markov Decision Process, where the state is reset after each episode of length T, the series of states, actions, and rewards that occur in an episode make up the policy's trajectory or roll-out. Each policy's trajectory accumulates rewards from the environment, resulting in the return $R = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$. The objective of RL is to discover an optimal policy π^* that maximizes the expected return from all states:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[R|\pi\right] \tag{3.1}$$

MDPs that are non-episodic and have an infinite time horizon ($T = \infty$) can also be taken into account. When the discount factor $\gamma < 1$, it ensures that there is no accumulation of an infinite sum of rewards. However, the methods that rely on complete trajectories are no longer applicable, and those that depend on a limited number of transitions are valid. The discount factor γ plays a crucial role in determining the significance of future rewards as compared to immediate rewards. If $\gamma = 0$, the agent will only focus on maximizing its immediate reward. On the other hand, if $\gamma \rightarrow 1$, the agent will strive to achieve a higher long-term reward.

The fundamental concept of RL is the Markov property, which ensures that the present state has a direct influence on the next state, and the future is dependent solely on the current state, making it independent of the past given the current state. Although most RL algorithms rely on this assumption, it is not very practical since it requires the states to be entirely observable [63].

3.2.5 Partially Observable Markov Decision Process (POMDP)

Partially Observable Markov Decision Process (POMDP) is an extension of the MDP [65], which is used in RL. Unlike MDPs, POMDPs take into account that the agent does not have complete information about the environment. In POMDPs, the agent observes a partial view of the environment $o_t \in \omega$ rather than the complete state. The agent then uses this partial information along with its policy to select an action that maximizes the expected reward. The state of the environment is not directly observable in POMDPs, and the agent must infer the state based on its observations, which adds complexity to the decision-making process.

A POMDP is a 7-tuple (S, A, T, R, Ω, O, γ) where [9]:

- S is a finite set of states $\{1, ..., N_S\}$;
- A is a finite set of actions $\{1, ..., N_A\}$;
- T : S × A × S → [0, 1] is the transition function (set of conditional transition probabilities between states);
- R : S × A × S → R is the reward function, where R is a continuous set of possible rewards in a range $R_{max} \in \mathbb{R}^+$;
- Ω is a finite set of observations $\{1, ..., N_\Omega\}$;
- O : S × Ω → [0, 1] is a set of conditional observation probabilities;
- $\gamma \in [0, 1)$ is the discount factor.

In a POMDP, the agent does not have direct access to the current state s, but rather observes an observation o that is probabilistically generated by the environment based on the true state. The observation o is used by the agent to update its belief about the state of the environment. The belief is a probability distribution over the set of states [58].

3.2.6 Taxonomy of RL methods

The classification of Reinforcement Learning is depicted in Figure 3.4, based on two approaches: Model-based methods and Model-Free methods, also Value-based methods and Policy-based methods.

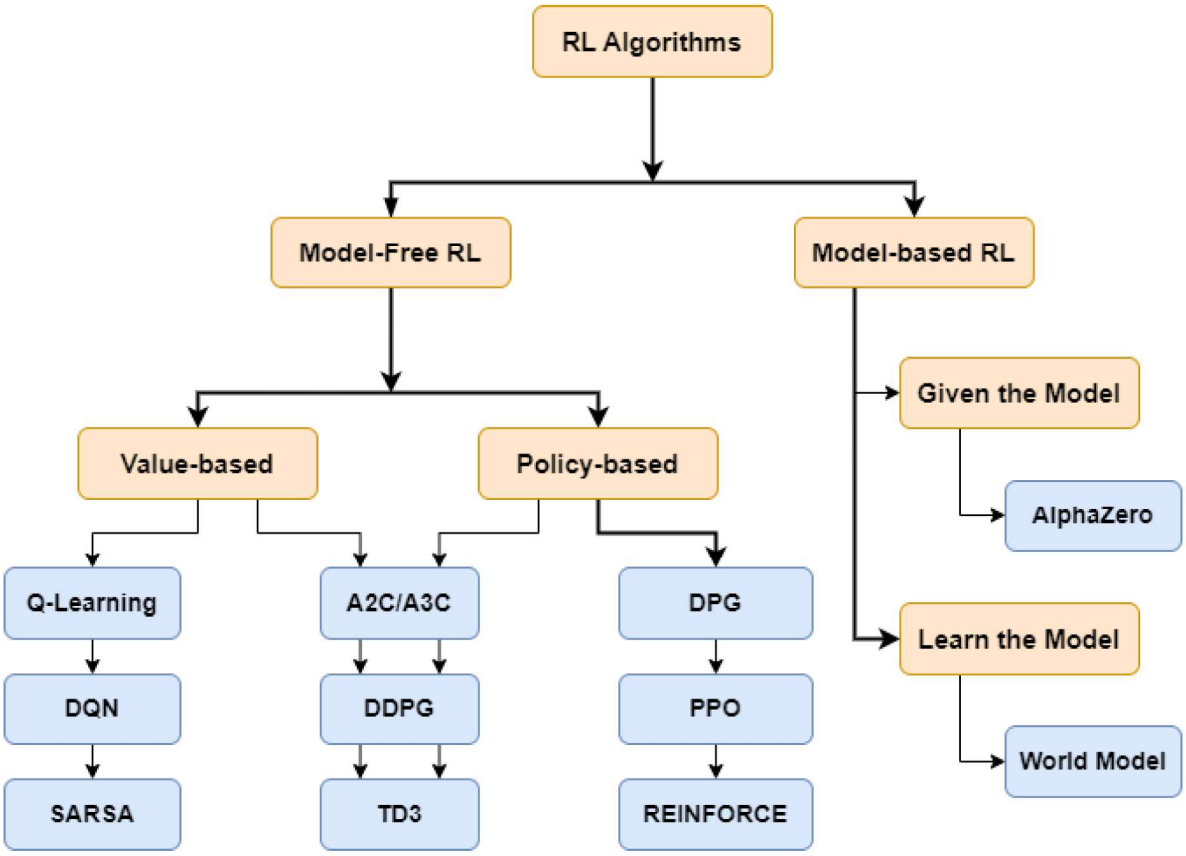


Figure 3.4: Map of Reinforcement Learning algorithms: Brown Boxes denotes different categories, Blue Boxes denotes different algorithms [10].

3.2.7 Actor-Critic Algorithms

Actor-critic algorithms are a type of reinforcement learning method that combines the benefits of both policy-based and value-based methods (see Figure 3.5 (a)). It consists of two components: the actor and the critic (see Figure 3.5 (b)). The actor learns a policy $\pi(a|s)$ that maps state s to actions a, while the critic estimates the value function V(s) that approximates the expected cumulative reward from being in state s and following the policy π. With DRL, both the actor and the critic can be modeled using non-linear neural network function approximators [66]. The actor uses policy gradients, which are derived from the policy gradient theorem, to adjust the policy parameters ω. Meanwhile, the critic, which is parameterized by θ, estimates the approximate value function for the current policy π, where Q(s, a; θ) is an approximation of $Q_\pi(s, a)$.

The actor-critic architecture has several advantages over other reinforcement learning methods. First, it enables the agent to learn both a policy (i.e., the actor) and a value function (i.e., the critic) simultaneously, which can lead to faster learning and better performance. Second, the critic provides feedback to the actor on the quality of the actions taken, allowing the actor to learn from his mistakes and improve the policy over time. Finally, the actor and critic

can be implemented using different neural network architectures, allowing for greater flexibility in the design of the overall system [9].

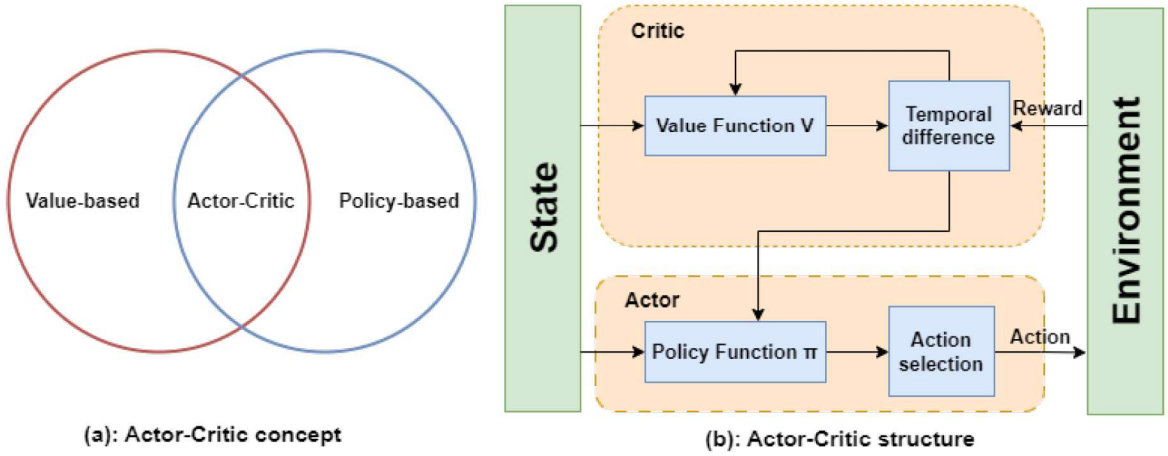


Figure 3.5: Overview of Actor-Critic method [11].

Table 3.2 provides a comparison between six popular Actor-Critic algorithms: A2C, A3C, TRPO, ACKTR, DDPG, and PPO. Each algorithm is compared on several factors, including exploration, gradient estimation, learning rate, sample efficiency, and scalability:

Advantage Actor-Critic (A2C)

A2C is an on-policy algorithm that does not require additional exploration mechanisms and uses Monte Carlo gradient estimation. It has a high learning rate and low sample efficiency, meaning it requires more experience samples to learn the policy and value functions. However, A2C is relatively simple and scalable to large and complex environments [66].

Asynchronous Advantage Actor-Critic (A3C)

A3C is also an on-policy algorithm that uses Monte Carlo gradient estimation but has a higher sample efficiency than A2C due to its asynchronous learning scheme. It can also incorporate additional exploration mechanisms if necessary. A3C has a high learning rate and is highly scalable [67].

Trust Region Policy Optimization (TRPO)

TRPO is an on-policy algorithm that uses KL-divergence exploration and natural gradient estimation. It has a low learning rate but high sample efficiency, making it a good choice for high-dimensional and continuous action spaces. However, TRPO can be computationally expensive and less scalable than other algorithms [68].

Actor-Critic using Kronecker-Factored Trust Region (ACKTR)

ACKTR is an on-policy algorithm that uses KL-divergence exploration and Kronecker-factored natural gradient estimation. It has a low learning rate but high sample efficiency and is highly scalable due to its efficient use of experience samples. However, ACKTR is also computationally expensive and may not be practical for all environments [69].

Deep Deterministic Policy Gradient (DDPG)

An off-policy actor-critic algorithm that learns a deterministic policy (i.e., one that maps states directly to actions) rather than a stochastic policy. DDPG is typically used in continuous action spaces and is known for its stability and robustness [70].

Proximal Policy Optimization (PPO)

A family of policy gradient algorithms that use a clipped surrogate objective function to prevent large policy updates. PPO has become popular in recent years due to its simplicity and strong empirical performance [71].

Table 3.2: Comparison between famous Actor-Critic algorithms.

Algorithm	Exploration	Gradient Estimation	Learning Rate	Sample Efficiency	Scalability
A2C [66]	None or Epsilon-Greedy	Monte Carlo	High	Low	Low
A3C [67]	None or Epsilon-Greedy	Monte Carlo	High	High	High
TRPO [68]	KL-Divergence	Natural Gradient	Low	High	Low
ACKTR [69]	KL-Divergence	Kronecker-Factored Natural Gradient	Low	High	High
DDPG [70]	Noise injection (Non-greedy)	Monte Carlo	Low	Low	High
PPO [71]	Through stochasticity (Non-greedy)	Natural Gradient	High	High	High

3.3 ML/DRL for QoS Path Optimisation

Machine learning's ability to solve intricate decision-making problems has inspired the exploration of RL-based techniques as a viable solution for addressing QoS routing problems in SDN. These issues are characterized by fluctuations over time and randomness in resource and service availability, including factors such as bandwidth and link, as well as service parameters such as loss and delay. As a result, this problem has been extensively researched by the scientific community, especially after the emergence of SDN and the new routing capabilities it comes along (data and control separation).

Problematic

QoS Routing Optimization is the process of selecting the most efficient path for data to travel through a network, while also ensuring that the data meets certain quality of service requirements. It takes into account various factors such as network congestion, bandwidth availability, and packet loss rates to determine the best path for data to travel through the network.

The goal of QoS routing optimization is to ensure that network resources are used efficiently while also providing a high QoS to users. This is especially important for real-time applications such as voice and video, which require low latency and high bandwidth. it is achieved through the use of specialized algorithms that take into account the various QoS requirements and constraints of the network. These algorithms calculate the optimal path for data to travel through the network while also ensuring that the QoS requirements are established [56].

3.4 Related Works

In this section, we review the literature related to the QoS-aware routing optimization in SDN generally. Hence, the discussion of related work is categorized into two broad groups: QoS Routing optimization for SDN specifically and for SD-WAN.

3.4.1 Optimizing QoS Routing in SDN

The use of Deep Reinforcement Learning (DRL) for QoS routing optimization in SDN has garnered significant attention in the scientific community. Several recent studies have explored the application of DRL in this context, focusing on different aspects and proposing various approaches.

Two studies, [72] and [73], propose centralized routing schemes based on Deep Deterministic Policy Gradient (DDPG). While [72] presents a comprehensive reward function customizable through parameters, it focuses solely on delay optimization. On the other hand, [73] adjusts the DDPG method to support differential QoS provisioning but does not address strategy generation. Similarly, [74] proposes a centralized implementation using Advantage Actor-Critic (A2C) with delay and packet loss optimization. Another study by [75] deploys a centralized architecture based on Asynchronous Advantage Actor-Critic (A3C) algorithm to balance network traffic loads, considering parameters like latency, throughput, and packet loss ratio.

Pham et al. [56] propose a centralized Convolutional Neural Network (CNN)-based solution for QoS routing using Knowledge Defined Networking (KDN). Their aim is to improve routing configuration by learning the mutual impact between network flows. However, their proposed mechanism may not perform well in more complex network problems. Another study by Zhang et al. [76] presents DSOQR, an online QoS-routing framework based on the SA3CR algorithm.

This framework dynamically selects routing paths based on current network conditions and service requirements but does not consider the traffic overhead problem. Naeem et al. [77] propose a parallel SDN solution for online routing problems in IIoT smart healthcare systems. Similarly, Zhang et al. [78] introduce CFR-RL, a parallel scheme named Critical Flow Rerouting - Reinforcement Learning, which autonomously acquires a policy to identify critical flows and redistribute network link utilization using a Linear Programming (LP) problem.

In reference [79], the authors utilize the Double Deep Q Network (DDQN) algorithm and Multiple-Agent Deep Reinforcement Learning (MADRL) techniques to improve the intelligence and efficiency of SDN systems. Their approach outperforms other methods, including Dijkstra and OSPF. Some solutions also incorporate distributed deployment for their SDN frameworks. For example, Wang et al. [80] propose an SDN-based Internet of Vehicles (IoV) framework that employs the Q-learning technique to select routing algorithms.

3.4.2 Optimizing QoS Routing in SD-WAN

In contrast to SDN, the optimization of QoS routing in an SD-WAN context has received limited attention, with only a few researchers addressing this problem. One study by Majdoub et al. [81] proposes an efficient deep Q-based routing approach based on Multi-Agent Reinforcement Learning (MARL) in a multi-controller SD-WAN. The method combines the SPFLR and DQN algorithms to determine optimal routes in both intra-domain and inter-domain scenarios. However, this approach may encounter concurrency issues between agents.

Botta et al. [82] introduce a Reinforcement Learning framework aimed at optimizing the utilization of multiple channels that connect distributed sites within a company's SD-WAN. Their proposed methodology leads to a significant 33

Another study by Emmanuel et al. [83] presents an effective monitoring tool designed for capturing and organizing network packets or flows within a simulated SD-WAN environment using Graphical Network Simulator-3 (GNS3). They suggest using a Decision Tree-based Machine Learning algorithm to categorize the packets accurately and organize them in a database structure. This structure enables further analysis, such as examining congestion window or throughput, with the aim of enhancing network performance and optimizing resource utilization.

3.4.3 Discussion

In this comparative study, we have examined several related works in terms of control deployment, techniques used, and performance metrics (summarized in Table 3.3), as well as implementation details such as the simulator used and hardware type (summarized in Table 3.4).

From these tables, it is evident that a considerable number of studies have addressed

the routing optimization problem in SDN, proposing various methods using different DRL techniques such as A2C, A3C, DDPG, and others. Additionally, simulations are often conducted using Mininet, OMNet++, or GNS3.

However, only a few studies have been conducted on the path selection problem in SD-WAN. Deep Q-Network, Smart Policy Routing, and Local Search algorithms are examples of the proposed methods in this context. To fill this research gap, we propose a robust multi-layered hierarchical control architecture specifically designed for SD-WANs. Additionally, we present a novel Hierarchical Teacher-Student framework for QoS path selection deployed in branch office controllers.

Table 3.3: Related Works 1.

Paper	Objective	Deployment	Technique	SDN	SD-WAN	Performance metrics
[72]	DROM uses DDPG to optimize the routing and improve network performance.	C	DDPG	✓		Delay
[56]	A CNN-based agent is exploited in Knowledge-Defined Network for QoS routing.	C	CNN	✓		Latency, Packet loss, and qualified flow(QoS metric).
[74]	Authors uses A2C algorithm to achieve situation-awareness routing for SDN.	C	A2C	✓		Delay and Packet loss.
[73]	IQoR is a routing scheme based on DDPG to support differential QoS provisioning.	C	DDPG	✓		Delay and jitter.
[77]	Parallel Routing technique has been proposed for IIoT-based smart healthcare systems.	D	k-Shortest	✓		Loss, delay, and jitter.
[79]	Introduces a novel approach that involves leveraging MADRL techniques across different domains to enhance intelligence and efficiency.	D	DDQN	✓		Throughput, delay, and packet loss rate.
[80]	SDCoR is a Q-learning based method that responds to changes in the IoV environment by selecting a conventional routing algorithm.	D	Q-learning	✓		Delivery delay and delivery ratio.

[78]	CFR-RL: a parallel RL-based scheme that aims to reroute and evenly distribute network critical flow automatically with the help of Linear Programming (LP).	C	REINFORCE	✓	Average load balancing, end-to-end delay.
[75]	AQROM: an A3C-based algorithm to determine routing strategies that balance the traffic loads in the network.	C	A3C	✓	Latency, throughput, and packet loss ratio.
[76]	DSOQR: an Online QoS Routing framework based on A3C algorithm.	C	A3C	✓	Delay, throughput, and packet loss.
[81]	DQR: a DQN and MARL-based approach for flows forwarding through end-to-end QoS-aware paths.	D	DQN	✓	Resource utilization, throughput, and Packet Loss Rate.
[83]	Employ a ML technique to capture and examine and store SD-WAN packets in a database for immediate network analysis.	C	Decision Tree	✓	Congestion window, and throughput.
[82]	Investigate the use of Reinforcement Learning to Traffic engineering problems in SD-WAN.	C	Q-Learning	✓	Latency, throughput, and Service up-time.

C: Centralized | **D**: Distributed

Table 3.4: Related Works 2: Implementation details.

Paper	Simulator/Emulator	Package	Used Metrics	Hardware
[72]	OMNet++	TensorFlow and Keras	Delay and Throughput	NVIDIA Tesla P100 GPU, 24GB DDR4 RAM.
[56]	OMNeT++	Not mentioned	Latency and packet loss rate.	Not mentioned
[74]	Mininet + POX controller	TensorFlow	Delay and packet loss.	CloudLab cluster
[73]	OMNeT++	TensorFlow and Keras	Average delay and jitter.	Not mentioned
[77]	Mininet + POX controller	PyCUDA	Average delay and jitter.	Nvidia GPU
[79]	Mininet+ Ryu controller	Not mentioned	Throughput, delay, and packet loss rate.	4-core CPU and 2 GB of memory.
[80]	OMNet++, Veins, SUMO	Python, C++	Average delivery ratio.	Not mentioned
[78]	Not mentioned	TensorFlow	Average load balancing, end-to-end delay.	21 CPU cores with 2.6GHz each.
[75]	OMNet++	TensorFlow	Delay, Throughput, and Loss.	Intel Xeon CPU (32 cores), 64GB RAM.
[76]	Mininet + Ryu controller	TensorFlow 2.0	Delay, loss rate, and load of the system.	Intel i7-5500U CPU, 8GB RAM
[81]	Mininet + Ryu controller	TensorFlow	Throughput, RMSD, and packet loss.	Not mentioned
[83]	Graphical Network Simulator-3	Python script	Congestion window, and throughput.	Not mentioned
[82]	Mininet + Ryu controller	Not mentioned	Latency, throughput, and Service up-time.	Not mentioned

3.5 Conclusion

In this chapter, we explored Deep Reinforcement Learning techniques focusing on the Actor-critic and its algorithms. After that, we presented the existing DRL solutions for Routing Optimisation in both SDN as well as SD-WAN. Then a comparative study of the state-of-the-art literature was reviewed. Based on the outputs of this study, we conclude that DRL for Routing optimization in SDN is widely studied among the research community. However, SD-WAN is still not covered completely.

In the next chapter, We will introduce our Hierarchical Teacher-Student framework (HTS) for QoS Path optimization in SD-WAN in order to bridge this research gap.

4

Design of a Hierarchical Teacher-Students (HTS) framework for QoS Path Selection in SD-WAN: Our contribution

Contents

4.1 Introduction

In this chapter, we describe our Hierarchical Multi-Layered control architecture for SD-WANs. Then, we present the problem formulation of QoS Path selection as Multi-Agent Markov Decision Process (MAMDP). After that, we introduce a Hierarchical Teacher-Students framework (HTS) to resolve our MAMDP problem. Last but not least, we establish an efficient algorithm for the QoS path selection problem in SD-WAN. Before we proceed, we first introduce the Teacher-Student Framework.

4.2 Teacher-Student framework

The teacher-student framework [84, 85] is a learning paradigm in which a teacher model is trained to teach a student model. The teacher model is usually a complex and accurate model that has already been trained on a large dataset and can perform the task very well, while the student model is usually a simpler and faster model that can be deployed on devices with limited resources.

The goal of the teacher-student framework is to transfer the knowledge learned by the teacher model to the student model in a way that the student model can perform the task almost as well as the teacher model. This is achieved through a process called knowledge distillation [86], in which the teacher model's knowledge is transferred to the student model through a soft target that provides more information than just the correct answer.

During the knowledge distillation process, the student model is trained to match the output probabilities of the teacher model for a given input. This encourages the student model to learn from the teacher model's decision-making process, rather than just the final output. This process can improve the student model's accuracy and speed, and reduce its memory and computation requirements, making it more suitable for deployment in real-world scenarios.

The teacher-student framework is widely used in various applications, including natural language processing [87], plant disease classification [88], and speech recognition [89]. It can also be used to combine multiple models to achieve better performance and to develop more efficient models for mobile and edge devices.

4.3 System Architecture

4.3.1 Single Controller vs Multiple Controllers

Both single and multi-controller infrastructures can be considered as viable options for implementing a centralized control plane. In a single controller SDN, a single controller is responsible for managing the entire data plane by calculating the optimal flow routes, utilizing its high computing power and global visibility of network states and traffic. This simplifies the management of complex flows and allows network operators to have direct and effective control over the network by configuring a single controller. However, single-controller SDNs face scalability issues as the network size or the number of flows increases, due to the computation limitations of a single controller. Additionally, reliability is a critical concern due to the potential single point of failure with the controller.

In the other hand, When a single controller's capability is insufficient for managing an entire network or when multiple controllers are more cost-effective, multi-controller SDNs are preferred. Under the multi-controller model, multiple controllers collectively manage the data plane, and a centralized interface is necessary to provide virtualization for underlying switches

that still view a single control unit from their perspectives. As a result, due to the widespread adoption of multi-controller models in large-scale networks, a multi-layer hierarchical architecture is proposed to address the design complexity and information inconsistency [90].

4.3.2 Multi-Layer Hierarchical SD-WAN (MLHS) Control Plane Architecture: Our contribution

Inspired by [91, 92], we propose a geographically distributed hierarchical architecture specifically for SD-WAN. We distinguish 2-layered controllers: a **Root controller** and a **Domain Controller**. As shown in Figure 4.1, the proposed architecture simplifies the process of synchronizing global information and enhances the efficiency of communication in an SD-WAN scenario. Proper deployment of either a root controller or a domain controller can enhance their ability to manage network time, distribute the load evenly between controllers, and reduce data loss in the control plane in a useful manner.

In our proposed scheme, we deploy the Root Controller in the organisation's headquarters, and a Domain Controller in each Branch Office.

Root Controller

The global view is controlled by the root controller (headquarters), which plans the inter-domain path. The network topology of each Branch is abstracted into nodes within the root controller. The branch controller can only identify if an address belongs to its domain. If the destination address is not within the local area, it is reported to the root controller. The root controller first determines the location of the destination address and then performs inter-domain route planning based on the reported status and connection information from the area controller.

Domain Controller

The primary role of the domain (Branch) controller is to compute the intra-domain routes and provide the root controller with relevant resources. These resources are comprised of two main parts: the first includes the connection link between the domain and the neighboring regions, along with their QoS requirements. The second part pertains to the resource limitations of all paths between the boundary nodes (Edge Routers) in the domain. The root controller selects the appropriate domain for cross-domain routing planning based on the information reported by the branch controller. The goal is to calculate the path based on the service type. Each domain controller is responsible for the customer Premises Equipment(specifically the edge routers) of the branch office.

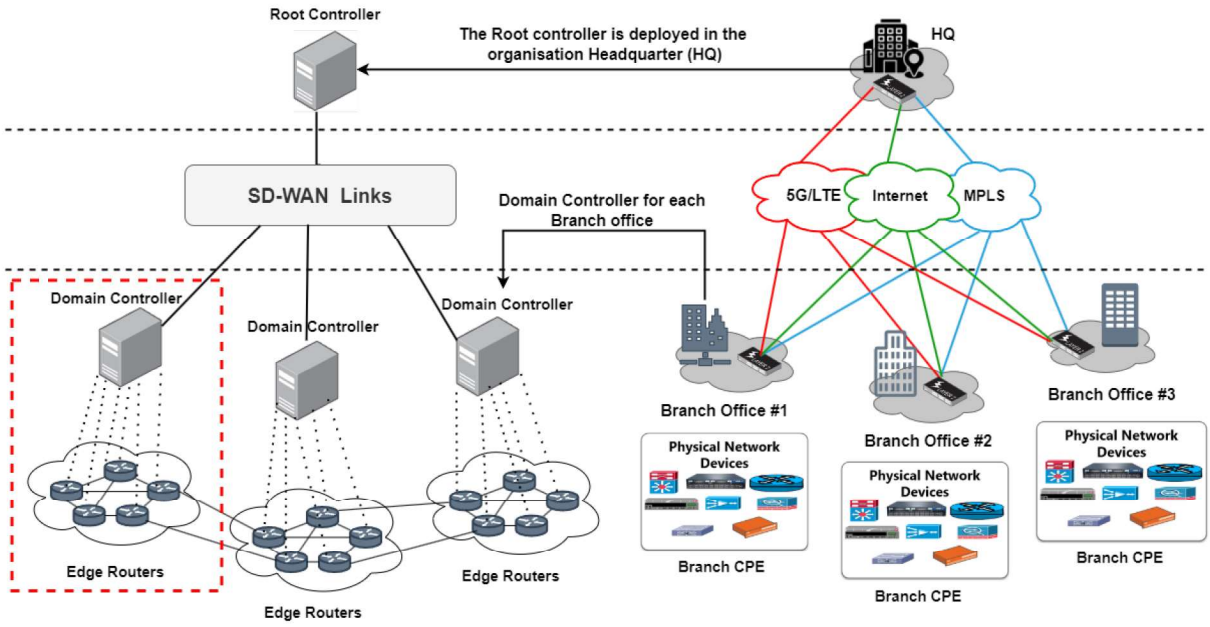


Figure 4.1: Our proposed Multi-layered Hierarchical SD-WAN enabled architecture.

4.3.3 Hierarchical Teacher-Students (HTS) framework for Path optimization in Domain Controllers: Our Contribution

As mentioned earlier, each domain controller is responsible for the branch office edge routers. However, if a single agent is deployed in a branch controller to make all the path selection decisions, it could severely impact the scalability of the approach. As the network size increases, the action space will expand significantly, making it too complicated to facilitate effective training and precise inference.

To tackle this problem, we break down the process of generating paths into a multi-agent DRL challenge where the students are DRL agents and the teacher is a domain knowledge.

As shown in Figure 4.2, each edge router is managed by a student agent, these students are supervised by a teacher that provides a guide-on-the-side backup plan when the students fail to generate a safe path. The teacher is a domain knowledge module and the students are DRL agents that choose the next hop for the edge router responsible for the creation of routes considering the QoS requirement and the flow destination IP address if we are using IP-based underlay or labels in the case of MPLS-based underlay or if we are combining both of them (see section 4.4.3). To achieve this with minimal message exchange, the student agents must collaborate with each other. Additionally, the agents should avoid unsafe routes, such as path loops and congestive paths caused by DRL's random exploration. The teacher (domain knowledge) interferes and provides a safe path based on a pre-define deterministic routing algorithm when the students generate an unsafe route. Furthermore, the student agent will receive an extra penalty through the loss function in order to minimize the difference between

its decision and the teacher's advice in the next training episodes.

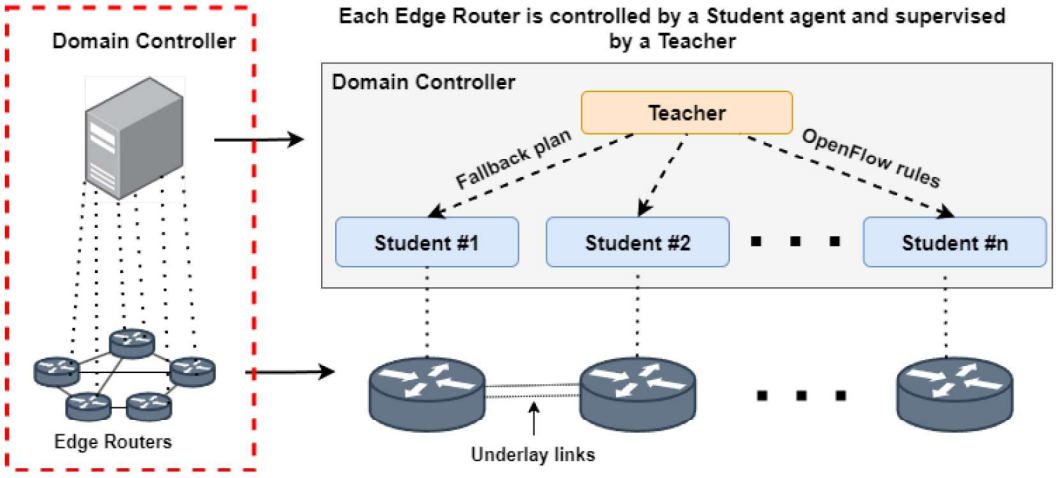


Figure 4.2: Envisioned Domain Controller Architecture with the HTS deployment.

Every network flow has specific performance metrics requirements, such as delay, throughput, and packet loss rate. These requirements can be grouped into a few categories, known as service types, and we have defined four service types:

- Throughput-sensitive
- Delay-sensitive
- Delay-throughput-sensitive
- Delay-loss-sensitive

For each service type, we have proposed an adaptive reward function that indicates the extent to which the flow's requirements are met. Our goal is to determine the next hop for each flow at every node it traverses to create a route, ensuring that each flow is forwarded successfully from the source node to the destination node without any path loops or black holes. Additionally, we aim to maximize the total reward of all traffic flows.

4.4 System Model

In this section, we describe the details of our system model. First, we formulate the QoS path selection problem and highlight the system objective. Then, we present the reward function to assess the quality of the selected path for different types of flows. Next, we model the path generation process as a multi-agent MDP within the Teacher-Student architecture. This combination led to the born of our framework named: Hierarchical Teacher-Students (HTS).

4.4.1 Problem formulation

The network is represented as a directed graph G(V, E) with switches and routers as nodes in set V and links in set E. Each link has a limited capacity. At a certain point in time, a flow request enters the network with a specified bandwidth requirement. Our model assumes a discrete-time system, where time is divided into consecutive timeslots labeled as t=1,2,... and a flow request for the t^{th} timeslot is received at the beginning of that timeslot. Let k be the number of the service types and ϕ is the service type of the t^{th} flow ($1 \leq \phi \leq k$). Let $F_\phi(t)$ be the reward function of the t^{th} flow. The objective is:

$$Max \lim_{T\to\infty} \frac{1}{T} \sum_{\iota=1}^{T} F_\phi(t) \tag{4.1}$$

4.4.2 QoS Reward design

Our aim is to assess selected paths for flows of varying service types. To achieve this, we suggest a comprehensive reward function that combines several performance metrics while taking into account fairness among flows. Specifically, we consider three performance metrics: throughput, delay, and packet loss ratio, to diverse Quality of Service (QoS) requirements. The reward function is denoted below:

$$F_t = \alpha \log(Throughput) - \beta Delay - \gamma Loss, \tag{4.2}$$

Where $\alpha, \beta, \gamma \in [0, 1)$ are the tunable weights that determine the importance of QoS metrics. We set α, β, and γ are 0,1,0, respectively for delay-sensitive flows. α, β, γ are 1,0,0 for throughput-sensitive flows. α, β, γ are 0.5, 0.5, 0 for delay-throughput-sensitive flows. Finally, for delay-loss-sensitive flows we set $\alpha = 0$, $\beta = 0.5$, and $\gamma = 0.5$.

4.4.3 Our Teacher-Student framework (HTS)

Inspired by [93], our approach involves a new Teacher-Students learning framework where deep reinforcement learning agents (students) strive to both maximize their expected cumulative return and minimize the distance to domain knowledge teaching data. This approach has the benefit of being universal and not dependent on specific deep reinforcement learning models or networking applications. A neural network is used to determine the probability of a certain action based on a given system state. This probability is determined by the network's weights, denoted as θ. During DRL training, these weights are updated iteratively to determine the optimal actions for each training instance.

As mentioned before, the system is modeled as Multi-Agent MDP (see section 4.4.4), which means that the student agents collaboratively maximize the expected cumulative reward. At the same time, they try to avoid selecting unsafe paths (e.g., path loops) and get a penalty

from the teacher that minimizes the cumulative reward, which eventually leads to terrible performance. As a result, we assume in our framework that the teacher does not only check the student's decisions and generate penalties but also interferes with the path selection decision by executing a customized deterministic path selection algorithm through its domain knowledge module. This helps to improve the reliability of our scheme by providing a backup plan for the student agents when they fail to generate a safe path.

As Figure 4.3 depicts, we consider the student as a DRL agent (actor-critic network, PPO network, etc.) that interacts with other students to collaboratively learn a path selection policy in SD-WAN by maximizing the cumulative reward. The teacher is a domain knowledge entity that consists of two integral modules, the first one is the Path Decision Check module which observes the path selected by its students. The second one is the Domain knowledge module which is a pre-defined deterministic path selection algorithm such as Stateless Hierarchical Routing (SHR) that provides the backup plan for the students when the latter generate unsafe paths. From a structural point of view, the proposed HTS framework can be described as a Hierarchical Teacher-Students, where the domain expert acting as a Teacher occupies a superior level in the hierarchy, while the student agents are situated at a lower. level.

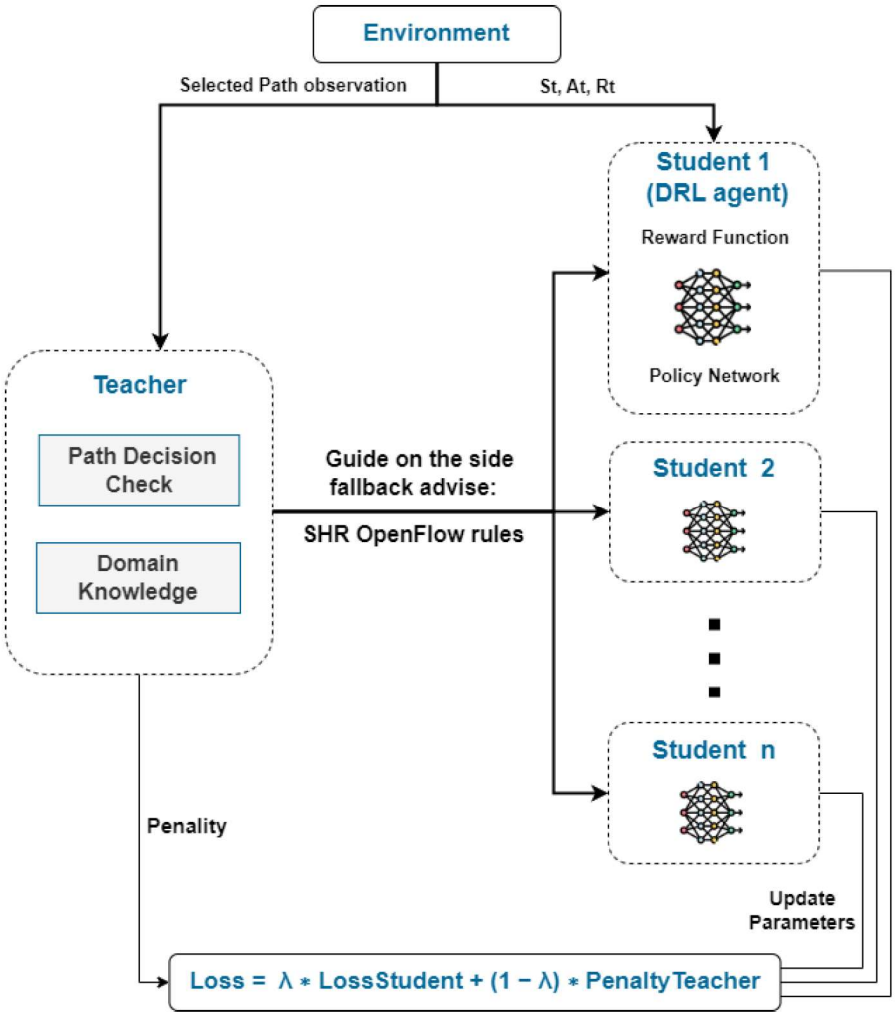


Figure 4.3: Envisioned Teacher-Student Framework.

As shown in Figure 4.4, If the students generate a safe path (λ=1), their reward and loss are then calculated to evaluate their actions and to eventually activate the edge router data plane using OpenFlow rules. conversely, if one of the students fails to select a safe path (λ=0), in this case, the teacher interferes by providing a backup plan advice to this specific agent through its domain knowledge module (eg. SHR algorithm). Additionally, the student agent receives a penalty in order to lower the erroneous path selection probability in the next episodes.

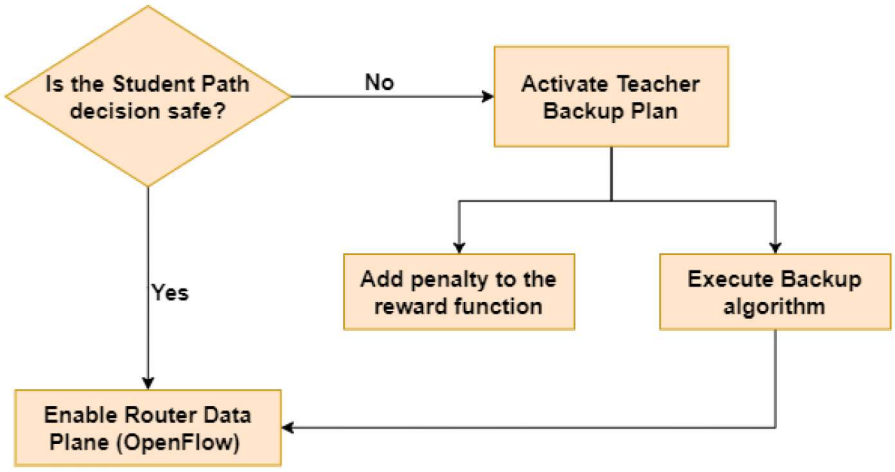


Figure 4.4: Teacher-Student data plane flowchart.

Algorithm 1 presents the pseudo-code for the teacher-student learning process:

Algorithm 1: The proposed Hierarchical Teacher-Student framework

Data: Flow features, topology and link information. Active nodes vector for each flow request. Teacher's Domain Knowledge.

Result: Policy π_θ.

Initialize student neural network $\Delta\theta \longleftarrow 0$;

Initialize replay buffer;

Set hyperparameters including, α, γ, ϵ, batch size, and training episodes ;

Initialize the teacher module with a deterministic path selection algorithm (eg. SHR);

for *each episode* **do**

  Reset environment;

  for *each step in episode* **do**

    Obtain state s_t, action a_t(path selection), and reward r_t;

    if *Path is Safe* **then**

      Enable router data plane via openFlow rule;

      Update student weights $\theta \longleftarrow \theta + \Delta\theta$;

      Update exploration rate ϵ;

    else

      Execute Teacher's backup algorithm;

      Give advice to student by adding a penalty to the reward function;

      Install decision to the router data plane;

    end

    Store experience tuple in replay buffer;

    Check if episode is finished;

  end

  Repeat for new service requests as they arrive in real-time;

end

4.4.4 Multi-Agent (Student) modeling

As previously explained, we determine the path for individual flows step by step, in order to break down the path selection issue into smaller sub-problems that have less complex state and action spaces. When a flow request arrives at a node, the node, acting as the student agent, chooses the best next step for the flow. This process is repeated at each subsequent node until the flow request reaches its final destination. Ultimately, we obtain the route for the flow, which is a sequence of nodes in a specific order.

We model the process of creating routes as a standard Multi-Agent Markov Decision Process (MAMDP), but under the guidance of a teacher, where each node acts as a student agent. This MAMDP is defined by a tuple $\left(\mathcal{S}, \left\{\mathrm{A}^{\mathrm{i}}\right\}_{i\in\mathbb{N}}, \mathcal{P}, \mathcal{R}, \mathcal{D}\right)$. The global state space that is shared by all agents $i \in \mathbb{N}$ is represented by $\mathcal{S}$, while A^{i} refers to the action space of agent i.

The joint action space of all agents is represented by $\mathcal{A} = \prod_{i \in \mathbb{N}} \mathrm{A^i}$. The state transition probability of the MAMDP is denoted as $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the function for computing global rewards. D represents the states in which the teacher intervenes. We consider Eq.(4.2) as our reward function. It is important to note that in this type of MAMDP, every agent has access to the global state space and reward values.

After each agent takes a local action based on the current state $s \in \mathcal{S}$, the route of a flow can be determined by action $a \in \mathcal{A}$. The decision-making process only involves agents on the determined path, and each agent's decision depends on the prior agents' decisions on the same path. If we assume that $\{src, a_1, ..., a_{n-1}, dst\}$ is a route determined by action a, then the process of determining action a based on state s can be described as follows:

$$Pr(a|s) = Pr(a^1|s)Pr(a^2|s, a^1)Pr(a^3|s, a^1, a^2) \cdots Pr(a^n|s, a^1, a^2, ..., a^{n-1}), \tag{4.3}$$

The action of the i^{th} student agent, denoted by a^i, determines the next node in the routing path, which is the $(i + 1)^{th}$ node.

Our design uses a zero-one vector of length |V| to represent the conditional state, where each item in the vector corresponds to a node in the network. If a node has been selected to construct the routing path, its corresponding item in the vector is 1. Otherwise, it is 0. When an agent makes an action for a flow request, it uses the conditional state and global state together as the input. Simply putting:

State: Conditional state vector. Flow features such as source node, destination node, and service type. Topology information like shortest distance to neighbors. Link information (eg. link failure ratio).

Action: The output action for an upcoming flow request is a vector that represents the probability distribution for choosing neighboring nodes as the next hop.

Given that the state and action spaces in real networks are typically extensive, we use neural networks to approximate agents' policies, which map the state input to the action output. Specifically, the local policy network of agent i is denoted as $\pi^i_{\theta_i}(a^i|s, c^i)$, where θ^i represents the neural network's parameters, and c^i is the conditional state. The purpose of $\pi^i_{\theta_i}(a^i|s, c^i)$ is to approximate the target conditional probability distribution, $Pr(a^i|s, a^1, a^2, ..., a^{i-1})$.

Within our MAMDP model, the agents are required to cooperate to discover a policy $\pi^i_{\theta_i}$ that maximizes the average long-term reward for network path selection problem. The objective to be maximized is the global objective $J(\theta)$, where θ represents the policy parameters of all agents θ_i, and can be modified by calculating the gradient of $J(\theta)$. According to [94], the standard format for a policy gradient estimator in reinforcement learning has the form:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[A(s, a) \nabla_\theta \log \pi_\theta(a|s) \right], \tag{4.4}$$

After generalizing the advantage function and adding the teacher loss function we get:

$$\nabla_\theta J(\theta) = \lambda \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[G_t \nabla_\theta \log \pi_\theta(a|s) \right] + (1-\lambda) l(s), \tag{4.5}$$

In the equation provided, τ represents a state-action pair (s, a) that is selected through sampling from $p_\theta(\tau)$ using policy π_θ, and G_t= A(s, a) is an estimator of the advantage value that is usually used in actor-critic methods such as Advantage Actor-critic (A2C). A(s, a), determines the improvement of selecting action a over randomly selecting actions based on policy π_θ, given the state s. Specifically, a positive value of A(s, a) indicates that selecting action a is better than choosing actions randomly based on the current policy π_θ, or vice versa. Moreover, G_t in Proximal Policy Optimization (PPO) takes the form: $G_t = \frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)} A(s,a)$. The policy gradient estimator for each agent's policy parameters can be obtained by using equations (4.3) and (4.5) as follows:

$$\nabla_\theta J(\theta) = m^i \left[\lambda \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[G_t \nabla_\theta \log \pi_\theta(a|s) \right] + (1-\lambda) l(s) \right], \tag{4.6}$$

$$m^i = \begin{cases} 1 & \text{if node } i \text{ on the path,} \\ 0 & \text{if node } i \text{ not on the path.} \end{cases}$$

In Equation (4.6), a mask variable m^i is employed to determine whether a particular node i is included in the path, in contrast to the policy gradient estimator used in Equation (4.5). If m^i = 0, the policy parameters θ^i of node i will not be updated in the current step, as $\nabla_{\theta^i} J(\theta) = 0$. Consequently, student agents may utilize the gradients to adjust their policy parameters.

4.5 Conclusion

Through this chapter, we covered the global design of our system, starting by presenting the multi-layered hierarchical SD-WAN architecture. Following that, we formulated the path selection problem as a MAMDP. Then, we introduced our Hierarchical Teacher-Students (HTS) framework and detailed its architecture. In the next chapter, we will present the experimental study of our system, where we will test the performance of our framework and discuss the results.

5 Implementation and Experimental Results

Contents

5.1 Introduction

To implement our framework, we utilized various development tools. This chapter is specifically dedicated to presenting the working environment, programming language, and tools used to construct this system. Following that, we will describe and discuss the suggested method's outcomes.

5.2 Development tools

Several tools have been put together to develop our system. For starters, Python was used to implement our model using some Deep Learning and Reinforcement Learning libraries, where we used Pycharm as a Integrated Development Environment (IDE). Moreover, we constructed the experimental network using Ryu and Mininet to test the effectiveness of our framework. Figure 5.1 illustrates the different development tools used in this work.

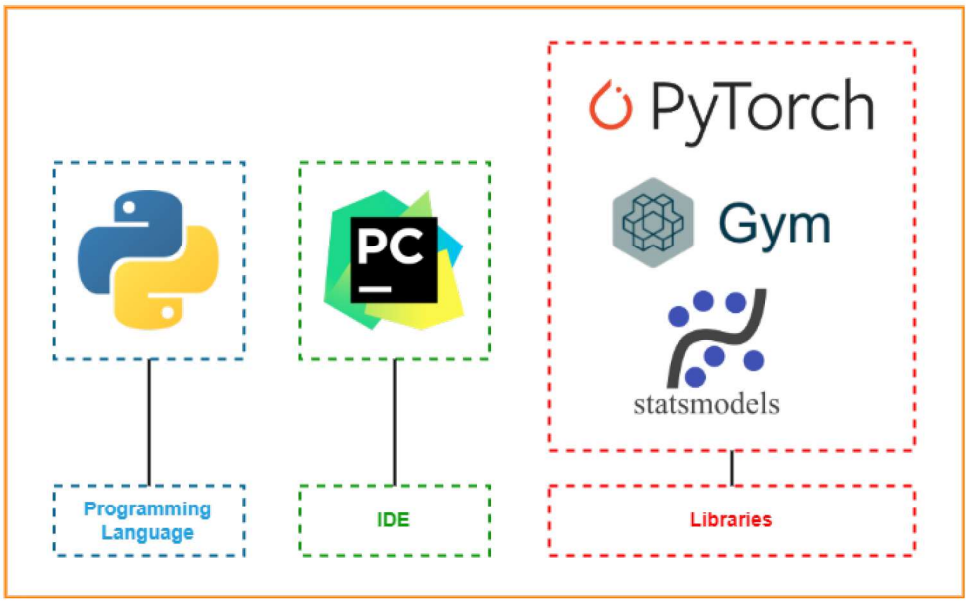


Figure 5.1: Implementation development tools.

5.2.1 Python

Python is a high-level programming language that has gained popularity over the years for its simplicity, readability, and versatility. Python is an interpreted language, meaning it is executed line by line rather than being compiled. It has a large standard library and many third-party libraries that make it a great tool for various purposes, including web development, data analysis, artificial intelligence, and scientific computing [95].

5.2.2 Pycharm

PyCharm is a popular IDE for the Python programming language. It was developed by JetBrains and offers a comprehensive set of tools and features to help developers write, debug, and test Python code more efficiently. PyCharm's code editor has advanced features such as syntax highlighting, code completion, and error highlighting, making it easier for developers to write code quickly and accurately [96].

5.2.3 Libraries

Python libraries are pre-written code modules that can be imported to perform specific tasks and reduce development time. In our case we have used:

PyTorch

PyTorch is a machine learning library that is widely used for developing neural networks, and has become increasingly popular due to its dynamic computational graph, making it easy

to build and train complex models [97].

Gym

Gym is an open-source toolkit for developing and comparing reinforcement learning algorithms. It provides a set of environments in which agents can be trained and a standardized interface for communicating between the agents and environments. This makes it easier for researchers and developers to experiment with different reinforcement learning techniques and test the performance of their agents in a variety of environments [98].

Numpy

NumPy is a powerful scientific computing library that enables fast and efficient operations on large arrays and matrices. It also provides many built-in functions for linear algebra and Fourier transforms [99].

Statsmodels

Statsmodels is a Python library used for statistical modeling and data analysis, providing a wide range of statistical models, tests, and tools for working with data [100].

5.3 Model Implementation

As described previously in section 4.4.3, the teacher is domain knowledge which consists of two modules. The first one is the Path decision Check module that is responsible for inspecting the students' actions, while the second module is the domain knowledge which is basically a deterministic routing algorithm Inspired by SHR [101]. SHR (Algorithm 2) is called when one of the students (agents) falls into a bad situation such as path loops in order to help them recover from this black hole, and then guide them to reduce the probabilities of falling back again using the extra penalty in the reward function design. Implementation details are shown in Figure A.2. On the other hand, the students are a DRL agents modeled as a MAMDP. Table 5.1 summarizes the model hyperparameters. We implement the model based on the reward function based on Eq. 4.2 and the policy loss function based on Eq. 4.6. Part of the student model implementation is depicted in Figure A.1.

Algorithm 2: Stateless Hierarchical Routing Algorithm

```
Algorithm I:
void main() {
    flag_prt ← DOWN // status of parent: UP, DOWN.
    num_crep ← 0 // number of CREP packets sent.
    flag_choose_prt ← NO // status of calling choose_parent().
    flag_crq_sent ← NO // status of CREQ packet sent.
    BS broadcasts CREQ packet
    sensor node broadcasts PREQ packet
    while rcv_pkt do // rcv_pkt is a received packet.
        if rcv_pkt is data packet then
            if destination is base station then
                if flag_prt = UP then
                    forward rcv_pkt to parent
                else // flag_prt = DOWN
                    buffer rcv_pkt in a queue
                end if
            end if
        else // rcv_pkt is routing packet.
            if rcv_pkt is CREQ packet then
                rcv_creq() // Algorithm II
            else if rcv_pkt is CREP packet then
                if (flag_prt = UP) || (my_addr = BS) then
                    send CACP packet
                end if
            else if rcv_pkt is CACP packet then
                rcv_cacp() // Algorithm III
            else if rcv_pkt is PREQ packet then
                if (my_addr = BS) || ((flag_prt = UP) &
                    (ID_src ≠ parent)) then
                    unicast CREQ packet to a communicating
                        party
                end if
            else if rcv_pkt is PRQY packet then
                send PREP packet containing necessary
                    information
            else if rcv_pkt is PREP packet then
                randomly choose a new parent from its
                    children according to received information
                send CREP packet to inform a new relation
            end if
        end if
    end while
}
```

Table 5.1: Model hyperparameters.

Hyperparameter	Value
Number of Students(Agents)	Depends on topology: 11 or 23
Learning rate	2.5×10^{-5}
Discount factor	0.99
RMSprop optimizer epsilon	1×10^{-5}
Entropy term coefficient	0.01
Value loss coefficient	0.5
Number of epochs	4
Number of pre-training steps	128
Number of training steps	1000000
Hidden layers	2 (hidden actor, hidden critic)

5.4 Experimental Results

We verify the effectiveness of our proposed HTS route selection method for different scenarios and parameters using simulations in this section.

5.4.1 Simulation Setup

Before we dive deep into the simulation setup, we briefly introduce Ryu controller and Mininet emulator:

Ryu Controller

Ryu is a popular open-source SDN controller that is written in Python. It provides a framework for developing SDN applications and enables network administrators to manage and control network traffic more efficiently. Ryu provides a set of libraries and APIs that allow developers to create custom SDN applications. These applications can be used to control network traffic, manage network resources, and implement network policies. Ryu also supports various network topologies, including flat, hierarchical, and mesh topologies. It is highly extensible, allowing developers to add new features and modules as needed. It also supports multiple OpenFlow versions, including 1.0, 1.3, 1.4, and 1.5, which enables Ryu to work with a wide range of network hardware. Some of the key features of Ryu include [102]:

- Support for OpenFlow protocol versions 1.0, 1.3, 1.4, and 1.5;
- Easy-to-use APIs and libraries for building SDN applications ;
- Support for a variety of network topologies;
- Extensibility, allowing developers to add new features and modules as needed;

- Compatibility with a wide range of network hardware.

Mininet

Mininet is an open-source network emulator that allows developers and researchers to create virtual networks on a single computer. It enables users to simulate complex network topologies and experiment with various network configurations and protocols without the need for physical network hardware. Mininet provides a command-line interface that allows users to create, configure, and interact with virtual networks. It also supports a variety of network protocols and applications, including OpenFlow, BGP, OSPF, and IPsec, which enables developers to test their network applications in a realistic environment. Several noteworthy features of Mininet are [103]:

- Support for creating virtual networks with any topology;
- Ability to run real operating systems and software applications within the virtual network;
- Integration with the OpenFlow protocol, enabling users to test and develop SDN applications;
- Support for network protocols and applications commonly used in production networks;
- Simple command-line interface for creating, configuring, and managing virtual networks.

Ryu and Mininet are commonly used together as a simulation environment for Software-Defined Networking. In our case, we used them test and evaluate the effectiveness of our scheme.

We utilize SDN deployment in branch offices to make it easier to create an experimental network using Ryu controller and Mininet. This allows us to implement our Hierarchical Teacher-Students framework and test its effectiveness. Our goal is to achieve positive outcomes so that we can apply it across all branch offices in the SD-WAN. Our experiment was conducted on Ubuntu virtual machine with an 8-core 3.2 GHz CPU, 8 GB memory, and an NVIDIA GeForce RTX 3050 Ti graphics card. The HTS was implemented based on A2C algorithm and using Pytorch 1.4.0. For the environment, two real-world network topologies, Abilene [104] with heavy load and GEANT [105] were used for the experiment. Abilene is comprised of 11 nodes and 14 bidirectional links, while GEANT is a larger topology with 23 nodes and 37 bidirectional links. Most links had a data rate of 10 Mbps, except for one bottleneck link in Abilene and two in GEANT, which had a data rate of 2.5 Mbps. Each link had a latency of 5 ms. Table 5.2 showcases the configuration used for this simulation.

Table 5.2: Simulation Parameters.

Parameter	Value
Topology	'Abilene', 'GEANT'
Number of switches	'11 for Abi', '23 for Gea'
Controller port	3999
Mininet host port	5000
Traffic Protocol	TCP
Controller type	'Ryu'
Link bandwidth	10 Mbps
Link delay	5ms
Southbound API	OpenFlow v1.3

5.4.2 Performance Evaluation

The evaluation process of the developed approach takes into account several performance metrics including Delay, Throughput, and Packet loss.

Delay

Delay refers to the time it takes for data to travel from its source to its destination in a network.

Throughput

Throughput is the amount of data that can be transmitted over a network in a given time period.

Packet loss

Packet loss refers to the failure of data packets to reach their intended destination in a network.

As depicted in Table 5.3, we set several flow types for the evaluation which are generated randomly by taking into account actual traffic data gathered by [104] and [105]. We use two sets of probabilities to determine the type of service a flow requires. The first set is used to test the effectiveness of the HTS framework by testing the impact of the teacher in the learning process and justifying its important role when it comes to convergence time. To do so, we compare our HTS with DRL solution based on Advanced Actor-critic (A2C) algorithm. While the second set is used to compare our method with three other routing solutions. In the first set, a flow has a probability of 0.4 of being delay-sensitive (type 1), 0.3 of being throughput-sensitive (type 2), and 0.3 of being delay-throughput-sensitive (type 3). There are no flows that

are delay-loss-sensitive (type 4). In the second set, a flow has a probability of 0.2 for type 1, 0.3 for type 2, 0.3 for type 3, and 0.2 for type 4.

Table 5.3: Flow Types.

Flow	Meaning
Type 1	Delay-sensitive
Type 2	Throughput-sensitive
Type 3	Delay-Throughput-sensitive
Type 4	Delay-Loss-sensitive

Convergence and Robustness

We examine how well the HTS system works in various scenarios, including situations with regular traffic and when there is a problem with a link using Abilene topology. We conduct tests with and without a teacher's presence, as our Teacher-Student learning framework considers the students as DRL agents and the teacher as a source of domain knowledge that offers guidance to the students. We provide evidence that having a teacher present positively affects the system's ability to learn, especially in terms of how quickly it converges and how reliable it is.

Figures 5.2 and 5.3 show that under normal traffic conditions, HTS outperforms a typical DRL solution, with HTS achieving faster convergence in both delay and throughput ratio metrics. Our approach results in the lowest delay values and achieves near 99% throughput ratio after convergence. We also observed that involving the teacher significantly reduces delay, indicating that our framework effectively speeds up the learning process and prevents bad path selection decisions that could result in high delay.

To test the resilience of the HTS framework, we compared its performance to a DRL solution in the event of a link failure. We simulated a failure on the bottleneck link of Abilene at the 40000^{th} timeslot, and the results are presented in Figures 5.4 and 5.5. We observed that without the teacher, the DRL solution experienced a sharp increase in delay and a simultaneous decrease in throughput ratio after the failure occurred. However, when the teacher is used as in the HTS approach, the delay and throughput ratio does not change significantly. The reason is that HTS provides a backup plan mechanism to avoid poor routes generated by the student agents in such specific cases. Additionally, the teacher, through our reward function design, helps the students learn a new policy quickly.

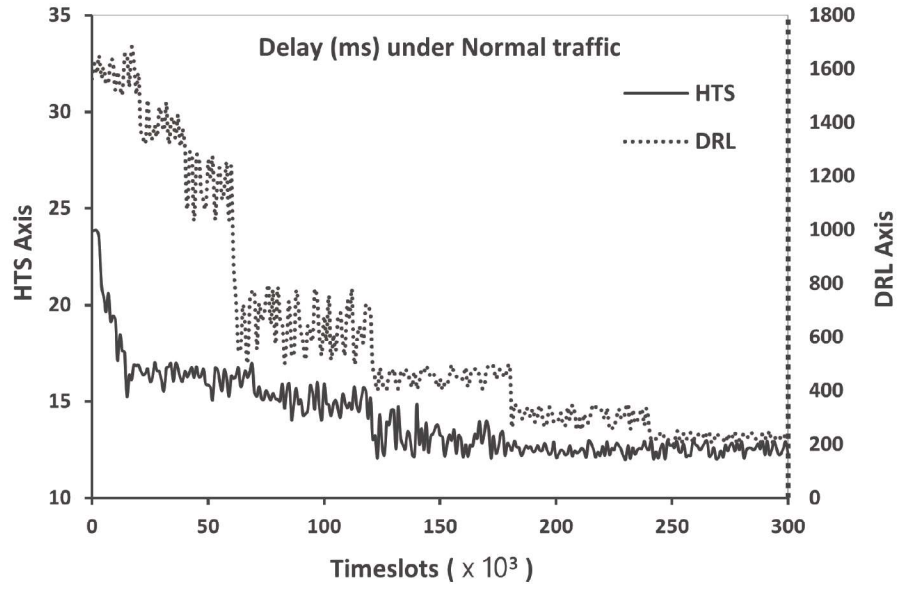


Figure 5.2: Delay under Normal traffic.

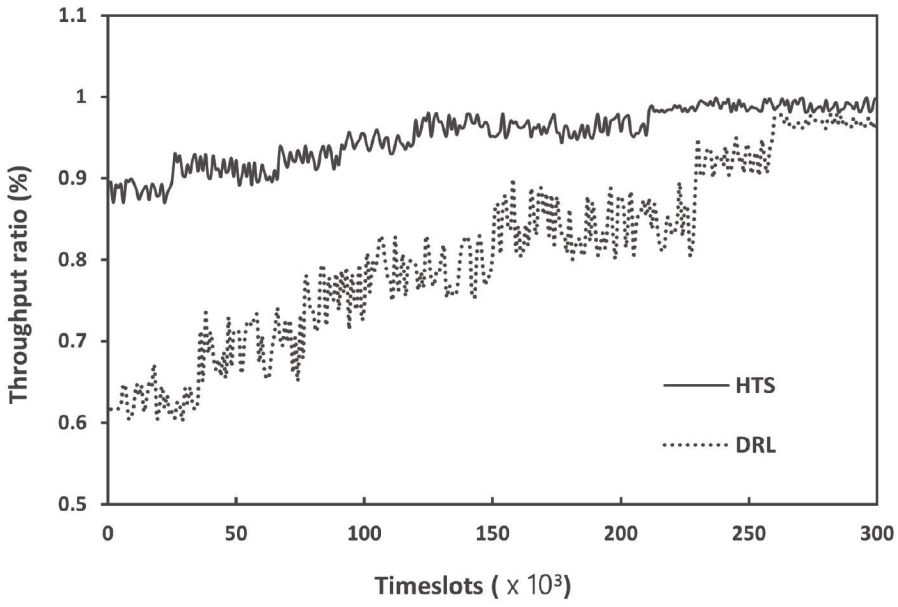


Figure 5.3: Throughput under normal traffic.

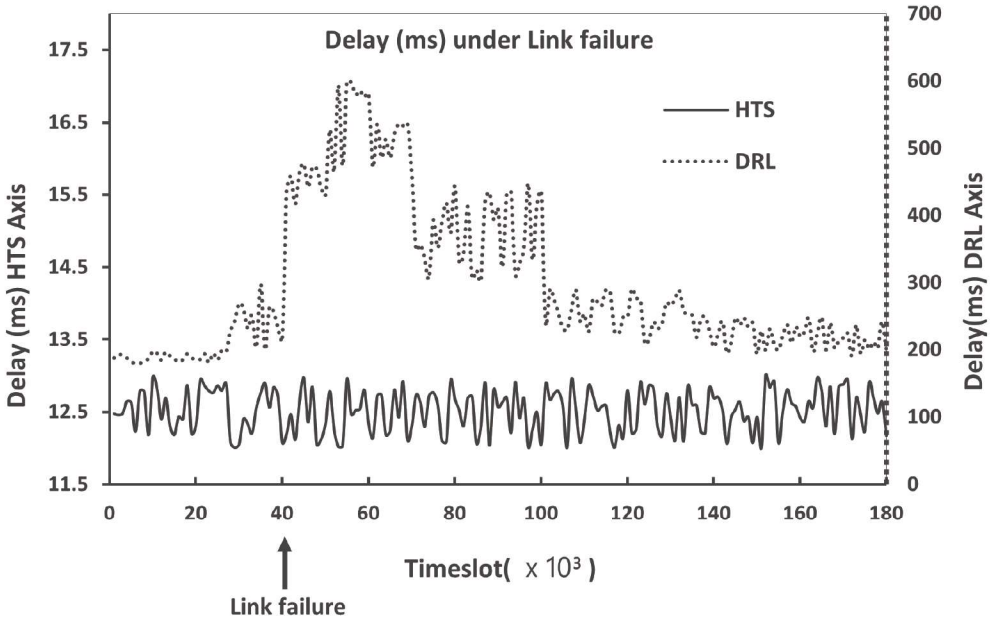


Figure 5.4: Delay under link failure.

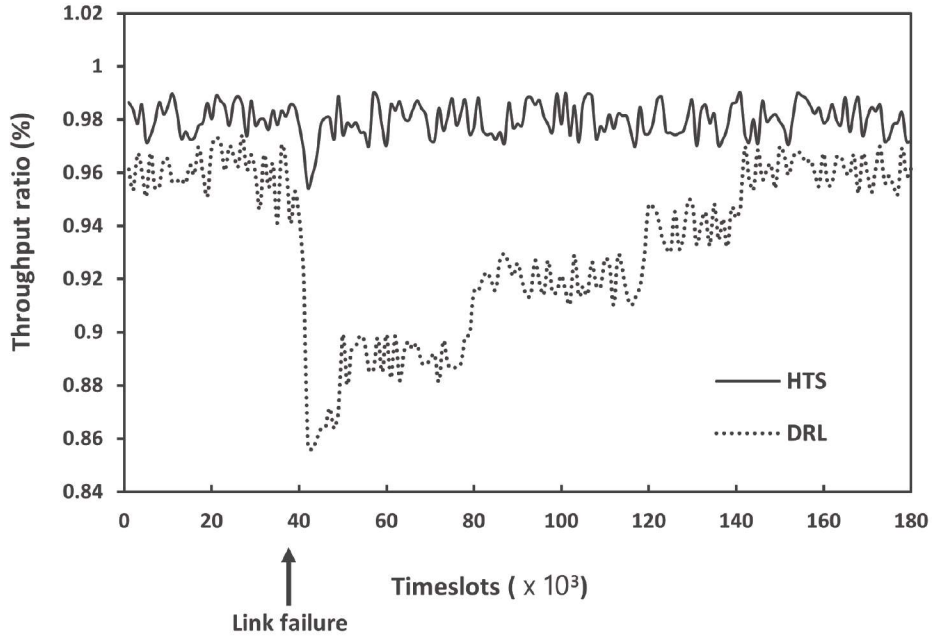


Figure 5.5: Throughput under link failure.

Performance validation

In order to validate our path selection scheme, we compare our HTS framework with two other solutions including SPR, which always uses the path with the least hop number for each flow request and the QoSR which employs various techniques depending on the type of flow it is dealing with. For a flow that is sensitive to latency, it uses SPR. On the other hand, for a flow that values throughput over latency, it employs Load Balancing Routing (LBR) to select a route that has the highest remaining capacity. For a flow that is sensitive to both latency and throughput, QoSR computes a capacity-constrained shortest path. Lastly, for a flow that is sensitive to latency and loss, SPR is used.

Figure 5.6 show the comparison results of the performance metrics under different scenarios (different topologies and flow type requirements). Where Figs. 5.6a, 5.6c, 5.6e shows average delay, average throughput, and average packet loss, respectively using a heavy-load traffic Abilene topology. While Figs. 5.6b, 5.6d, 5.6f illustrate the same metrics using GEANT, which is a bigger topology in terms of number of nodes and links. It is worth mentioning that the benchmark values are referenced from [106].

HTS routing has proven to be the superior choice in Abilene's heavy-load scenario, because it outperforms the other three baselines in all performance metrics. This indicates that HTS has the ability to adapt well to complex network environments with high traffic loads. Although QoSR is a better choice than SPR in heavy-load scenarios, HTS is still a better option overall because greedy-based methods like SPR and QoSR do not take future flows into account. To test the generalization of HTS, we conducted an experiment on GEANT, which is a larger network topology with more nodes and links. We found that it takes longer for student agents to converge in GEANT because more students are involved in routing decisions, and the input state space is much larger than in Abilene. However, HTS performed similarly to QoSR in terms of overall performance, with QoSR slightly outperforming HTS in flow delay and throughput ratio because such methods require less computational resources compared to agent-based solutions. However, HTS had a much lower packet loss ratio for Delay-loss-sensitive flows.

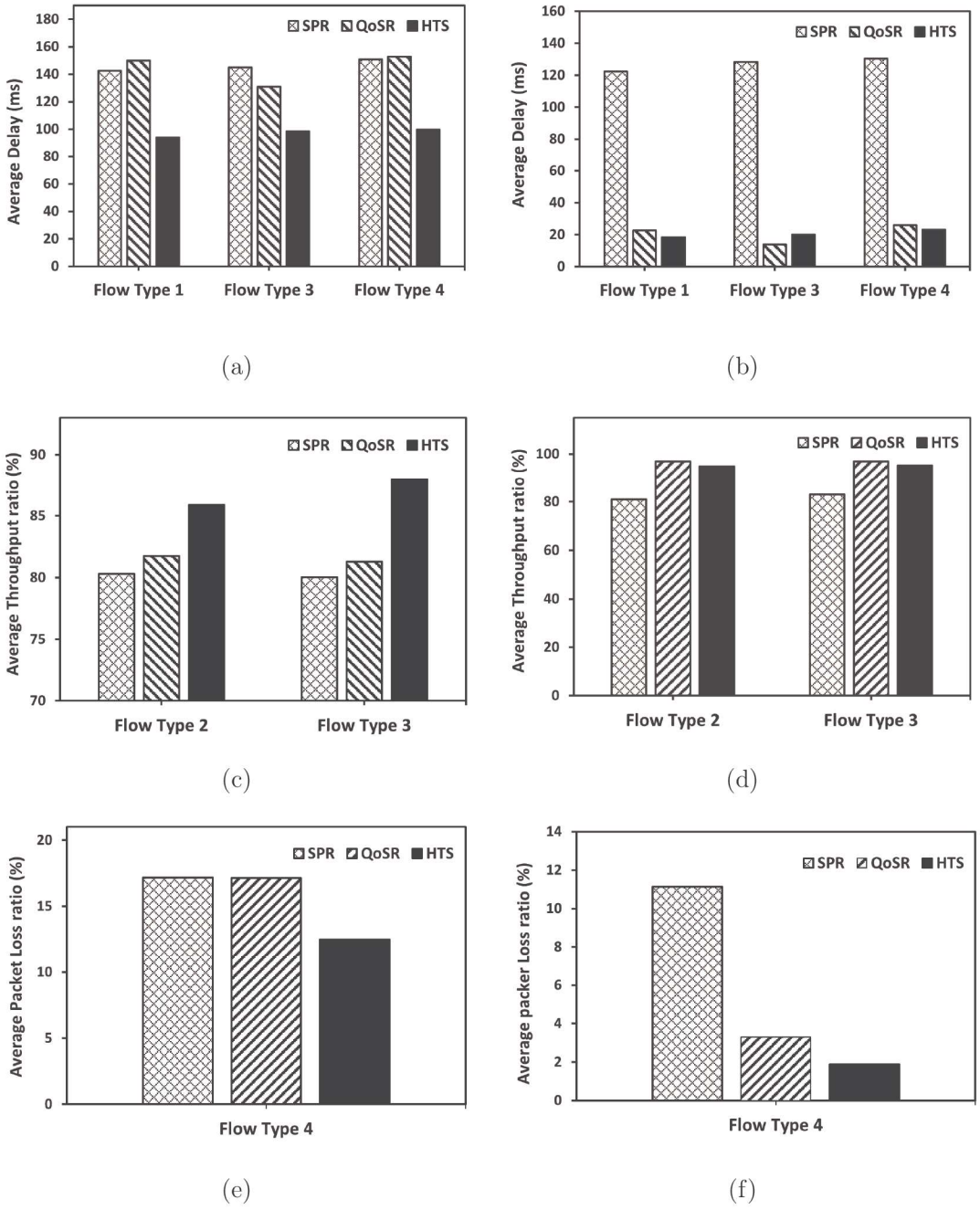


Figure 5.6: Performance metrics results with different scenarios

5.4.3 Discussion

In general, the evaluation of our HTS framework confirms beyond doubt that by incorporating a teacher component into the learning process, we can accelerate the learning process of DRL agents, while considering that the teacher in our scheme is a domain knowledge and not a prediction-based near-optimal policy algorithm. Moreover, HTS framework could be applied

to large-scale networks, where disruptions such as link failures or congestion are not uncommon. By using the HTS approach, the network could quickly adapt to changing conditions and maintain high levels of performance, even in the face of unexpected events. Additionally, the teacher component could provide valuable domain knowledge to the student agents, helping them to learn and make better decisions more quickly. This could result in more efficient use of network resources, reduced latency, and improved reliability for end-users.

By incorporating a teacher component into the learning process, we were able to speed up the learning process and prevent poor path selection decisions that could result in high delay and packet loss values.

5.5 Conclusion

In this chapter, we defined the tools and libraries used to conduct the simulation. After that, we presented the implementation details needed to develop our Hierarchical Teacher-Students (HTS) framework. Also, we put this framework under a series of networking test scenarios through simulation in order to investigate the effectiveness and robustness of our scheme. Finally, after analyzing and evaluating the HTS performance, we can say that the HTS framework could be an effective solution for real-world network routing problems, where disruptions and changing traffic patterns are common.

6 Conclusion and Future perspectives

Contents

6.1 Conclusion

SD-WAN (Software-Defined Wide Area Networking) is a powerful approach of network connectivity that uses software and virtualization technologies to simplify network operations and management. With the increasing adoption of cloud services and the need for remote access, SD-WAN has become crucial for organizations to ensure efficient, secure, and reliable communication between their branch offices and headquarters. To this end, ensuring high-quality network performance in SD-WAN requires efficient and reliable solutions that are able to tackle its numerous challenges such as the path selection problem. In this thesis, we have proposed a novel Hierarchical Teacher-Student (HTS) framework for path selection in SD-WAN, which leverages the strengths of both DRL agents and domain knowledge to achieve better performance and reliability. Our proposed HTS framework is deployed at the branch office level, where the teacher is a domain knowledge module that provides guidance to the DRL student agents. Through extensive experiments using Mininet and Ryu controller, we have demonstrated the effectiveness of our HTS framework in improving various quality-of-service metrics, including delay, throughput rate, and packet loss ratio. Specifically, our results show that our HTS framework outperforms greedy path selection algorithms as well as traditional DRL approaches under both normal traffic and link failure scenarios, with faster convergence times and greater resilience to disruptions.

In addition to our proposed HTS framework, we have also proposed a multi-layered hierarchical SD-WAN architecture that improves the scalability and efficiency of the network. This architecture consists of multiple layers. By using this architecture, we can distribute network

traffic more efficiently, reduce congestion, and ensure higher network performance by simply deploying the proposed HTS framework in each branch office.

6.2 Future Perspectives

Looking forward, there are several areas of future research that could build upon the work presented in this thesis. For example, future works could explore how to generalize HTS framework across all the branch offices, simulate the overall SD-WAN architecture and test its effectiveness. Additionally, we can incorporate security considerations into our HTS framework such as using Blockchain technology to secure communication between the organization's headquarters and branch offices.

Bibliography

[1] Amitava Mukherjee, Rashid A Saeed, Sudip Dutta, and Mrinal K Naskar. Fault tracking framework for software-defined networking (sdn). In *Resource allocation in next-generation broadband wireless access networks*, pages 247–272. IGI Global, 2017.

[2] Junfeng Xie, F Richard Yu, Tao Huang, Renchao Xie, Jiang Liu, Chenmeng Wang, and Yunjie Liu. A survey of machine learning techniques applied to software defined networking (sdn): Research issues and challenges. *IEEE Communications Surveys & Tutorials*, 21(1):393–430, 2018.

[3] Ángel Leonardo Valdivieso Caraguay, Lorena Isabel Barona López, and Luis Javier García Villalba. An overview of integration of mobile infrastructure with sdn/nfv networks. In *Proceedings of the 7th International Conference on Information Technology*, pages 250–265, 2016.

[4] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2014.

[5] Pham Tran Anh Quang, Sebastien Martin, Jérémie Leguay, Xu Gong, and Xu Huiying. Intent-based routing policy optimization in sd-wan. In *ICC 2022-IEEE International Conference on Communications*, pages 4914–4919. IEEE, 2022.

[6] starhub. Manage the next-gen virtual wan designed for the cloud-first enterprise. `https://www.starhub.com/business/products-and-services/data-connectivity/connecting-locally/managed-sd-wan.html`, 2023. Accessed on 24/03/2023.

[7] Hongjing Ji, Osama Alfarraj, and Amr Tolba. Artificial intelligence-empowered edge of vehicles: architecture, enabling technologies, and applications. *IEEE Access*, 8:61020–61034, 2020.

[8] starhub. Reinforcement learning - developing intelligent agents. `https://deeplizard.com/learn/video/my207WNoeyA`, 2020. Accessed on 27/03/2023.

[9] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, Joelle Pineau, et al. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018.

[10] Hongming Zhang and Tianyang Yu. Taxonomy of reinforcement learning algorithms. *Deep Reinforcement Learning: Fundamentals, Research and Applications*, pages 125–133, 2020.

[11] Md Billal Hossain. *QoS-Aware Intelligent Routing for Software Defined Networking*. PhD thesis, The University of Akron, 2020.

[12] Satyabrata Jena. Difference between software defined network and traditional network. `https://www.geeksforgeeks.org/difference-between-software-defined-network-and-traditional-network/`, 2020. Accessed on 20/03/2023.

[13] Maiass Zaher and Sándor Molnár. A comparative and analytical study for choosing the best suited sdn network operating system for cloud data center. *Annals of Emerging Technologies in Computing (AETiC)*, 6(1):43–59, 2022.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[15] Nguyen Cong Luong, Dinh Thai Hoang, Shimin Gong, Dusit Niyato, Ping Wang, Ying-Chang Liang, and Dong In Kim. Applications of deep reinforcement learning in communications and networking: A survey. *IEEE Communications Surveys & Tutorials*, 21(4):3133–3174, 2019.

[16] Thuy TT Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE communications surveys & tutorials*, 10(4):56–76, 2008.

[17] Ali Bou Nassif, Manar Abu Talib, Qassim Nasir, and Fatima Mohamad Dakalbab. Machine learning for anomaly detection: A systematic review. *Ieee Access*, 9:78658–78700, 2021.

[18] Rashid Amin, Elisa Rojas, Aqsa Aqdus, Sadia Ramzan, David Casillas-Perez, and Jose M Arco. A survey on machine learning techniques for routing optimization in sdn. *IEEE Access*, 9:104582–104611, 2021.

[19] Weichang Zheng, Mingcong Yang, Chenxiao Zhang, Yu Zheng, Yunyi Wu, Yongbing Zhang, and Jie Li. Application-aware qos routing in sdns using machine learning techniques. *Peer-to-Peer Networking and Applications*, 15(1):529–548, 2022.

[20] Hassan Fawaz, Julien Lesca, Pham Tran Anh Quang, Jérémie Leguay, Djamal Zeghlache, and Paolo Medagliani. Graph convolutional reinforcement learning for collaborative queuing agents. *IEEE Transactions on Network and Service Management*, 2022.

[21] Ning Yang, Haijun Zhang, and Randall Berry. Partially observable multi-agent deep reinforcement learning for cognitive resource management. In *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pages 1–6. IEEE, 2020.

[22] Sebastian Troia, Federico Sapienza, Leonardo Varé, and Guido Maier. On deep reinforcement learning for traffic engineering in sd-wan. *IEEE Journal on Selected Areas in Communications*, 39(7):2198–2212, 2020.

[23] Siamak Azodolmolky. *Software defined networking with OpenFlow*. Packt Publishing, 2013.

[24] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, and Haiyong Xie. A survey on software-defined networking. *IEEE Communications Surveys & Tutorials*, 17(1):27–51, 2014.

[25] Manar Jammal, Taranpreet Singh, Abdallah Shami, Rasool Asal, and Yiming Li. Software defined networking: State of the art and research challenges. *Computer Networks*, 72:74–98, 2014.

[26] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, et al. Onos: towards an open, distributed sdn os. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6, 2014.

[27] Global software-defined networking market–industry trends and forecast to 2028. `https://www.databridgemarketresearch.com/reports/global-sdn-market`. Last accessed 13/03/2023.

[28] Digital in 2018: World's internet users pass the 4 billion mark. `https://wearesocial.com/uk/blog/2018/01/global-digital-report-2018/`. Last accessed 14/03/2023.

[29] Arsham Farshad, Panagiotis Georgopoulos, Matthew Broadbent, Mu Mu, and Nicholas Race. Leveraging sdn to provide an in-network qoe measurement framework. In *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 239–244. IEEE, 2015.

[30] Marcel Caria, Admela Jukan, and Marco Hoffmann. Sdn partitioning: A centralized control plane for distributed routing protocols. *IEEE Transactions on Network and Service Management*, 13(3):381–393, 2016.

[31] The Learn Hub. Software-defined networking (sdn) vs traditional networking explained. `https://www.kyndryl.com/ca/en/learn/sdn-vs-traditional-networking`, 2023. Accessed on 20/03/2023.

[32] Joni Kytömäki. Artificial intelligence and machine learning with sd-wan. Master's thesis, 2021.

[33] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. *ACM SIGCOMM Computer Communication Review*, 43(4):3–14, 2013.

[34] Carlos Fernandez and Jose L Munoz. Software defined networking (sdn) with openflow 1.3 open vswitch and ryu. *UPC Telematics Department_ PhD Thesis*, 2015.

[35] T. Bouzid H. Ahmedi. *Software Defined Networking*. PhD thesis, UNIVERSITE AMAR TELIDJI LAGHOUAT, 2017.

[36] Alexander Shalimov, Dmitry Zuikov, Daria Zimarina, Vasily Pashkov, and Ruslan Smeliansky. Advanced study of sdn/openflow controllers. In *Proceedings of the 9th central & eastern european software engineering conference in russia*, pages 1–6, 2013.

[37] Lamiae Boukraa, Safaa Mahrach, Khalid El Makkaoui, and Redouane Esbai. Sdn southbound protocols: A comparative study. In *Emerging Trends in Intelligent Systems & Network Security*, pages 407–418. Springer, 2022.

[38] Wei Zhou, Li Li, Min Luo, and Wu Chou. Rest api design patterns for sdn northbound api. In *2014 28th international conference on advanced information networking and applications workshops*, pages 358–365. IEEE, 2014.

[39] Chenhui Wang, Hong Ni, and Lei Liu. An enhanced message distribution mechanism for northbound interfaces in the sdn environment. *Applied Sciences*, 11(10):4346, 2021.

[40] Alexandru L Stancu, Simona Halunga, Alexandru Vulpe, George Suciu, Octavian Fratu, and Eduard C Popovici. A comparison between several software defined networking controllers. In *2015 12th international conference on telecommunication in modern satellite, cable and broadcasting services (TELSIKS)*, pages 223–226. IEEE, 2015.

[41] Yongpeng Shi, Yurui Cao, Jiajia Liu, and Nei Kato. A cross-domain sdn architecture for multi-layered space-terrestrial integrated networks. *IEEE Network*, 33(1):29–35, 2019.

[42] Bo Han, Vijay Gopalakrishnan, Lusheng Ji, and Seungjoon Lee. Network function virtualization: Challenges and opportunities for innovations. *IEEE communications magazine*, 53(2):90–97, 2015.

[43] Yong Li and Min Chen. Software-defined network function virtualization: A survey. *IEEE Access*, 3:2542–2553, 2015.

[44] Hassan Hawilo, Abdallah Shami, Maysam Mirahmadi, and Rasool Asal. Nfv: state of the art, challenges, and implementation in next generation mobile networks (vepc). *IEEE network*, 28(6):18–26, 2014.

[45] Xiaoming Zhu, Bingying Song, Yingzi Ni, Yifan Ren, Rui Li, Xiaoming Zhu, Bingying Song, Yingzi Ni, Yifan Ren, and Rui Li. Software defined anything—from software-defined hardware to software defined anything. *Business Trends in the Digital Era: Evolution of Theories and Applications*, pages 83–103, 2016.

[46] Ala Darabseh, Mahmoud Al-Ayyoub, Yaser Jararweh, Elhadj Benkhelifa, Mladen Vouk, and Andy Rindos. Sddc: A software defined datacenter experimental framework. In *2015 3rd international conference on future internet of things and cloud*, pages 189–194. IEEE, 2015.

[47] Mark Carlson, Alan Yoder, Leah Schoeb, Don Deel, Carlos Pratt, Chris Lionetti, and Doug Voigt. Software defined storage. *Storage Networking Industry Assoc. working draft*, pages 20–24, 2014.

[48] Thomas Lin, Joon-Myung Kang, Hadi Bannazadeh, and Alberto Leon-Garcia. Enabling sdn applications on software-defined infrastructure. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–7. IEEE, 2014.

[49] Sanjay Uppal, Steve Woo, and Dan Pitt. Software defined wan for dummies, 2015.

[50] S Rajagopalan. An overview of sd-wan load balancing for wan connections. In *2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pages 1–4. IEEE, 2020.

[51] O. group. Five ways sd-wan is transforming cloud connectivity. `https://www.oracle.com/a/ocom/docs/five-ways-sd-wan-is-transforming-cloud-connectivity-wp.pdf`, 2019. Accessed on 18/03/2023.

[52] Sebastian Troia, Ligia Maria Moreira Zorello, and Guido Maier. Sd-wan: how the control of the network can be shifted from core to edge. In *2021 International Conference on Optical Network Design and Modeling (ONDM)*, pages 1–3. IEEE, 2021.

[53] Sebastian Troia, Ligia M Moreira Zorello, Alvin J Maralit, and Guido Maier. Sd-wan: an open-source implementation for enterprise networking services. In *2020 22nd International Conference on Transparent Optical Networks (ICTON)*, pages 1–4. IEEE, 2020.

[54] Fatma AL Deeb and Abdussalam Ali Ahmed. Software defined wide area network sd-wan: Principles and architecture. 2021.

[55] Zhenjie Yang, Yong Cui, Baochun Li, Yadong Liu, and Yi Xu. Software-defined wide area network (sd-wan): Architecture, advances and opportunities. In *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9. IEEE, 2019.

[56] Tran Anh Quang Pham, Yassine Hadjadj-Aoul, and Abdelkader Outtagarts. Deep reinforcement learning based qos-aware routing in knowledge-defined networking. In *Quality, Reliability, Security and Robustness in Heterogeneous Systems: 14th EAI International Conference, Qshine 2018, Ho Chi Minh City, Vietnam, December 3–4, 2018, Proceedings 14*, pages 14–26. Springer, 2019.

[57] Alexander L Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L Littman. Pac model-free reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 881–888, 2006.

[58] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[59] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[60] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

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

[62] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[63] Abdelkader Mekrache, Abbas Bradai, Emmanuel Moulay, and Samir Dawaliby. Deep reinforcement learning techniques for vehicular networks: Recent advances and future trends towards 6g. *Vehicular Communications*, 33:100398, 2022.

[64] Brendan O'Donoghue, Ian Osband, Remi Munos, and Volodymyr Mnih. The uncertainty bellman equation and exploration. In *International Conference on Machine Learning*, pages 3836–3845, 2018.

[65] Matthijs TJ Spaan. Partially observable markov decision processes. *Reinforcement learning: State-of-the-art*, pages 387–414, 2012.

[66] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[67] Ioannis Mitliagkas, Ce Zhang, Stefan Hadjis, and Christopher Ré. Asynchrony begets momentum, with an application to deep learning. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 997–1004. IEEE, 2016.

[68] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

[69] Yuhuai Wu, Elman Mansimov, Shun Liao, Alec Radford, and John Schulman. Openai baselines: Acktr & a2c. *url: https://openai. com/blog/baselines-acktra2c*, 2017.

[70] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[71] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[72] Changhe Yu, Julong Lan, Zehua Guo, and Yuxiang Hu. Drom: Optimizing the routing in software-defined networks with deep reinforcement learning. *IEEE Access*, 6:64533–64539, 2018.

[73] Yuanyuan Cao, Bin Dai, Yijun Mo, and Yang Xu. Iqor: An intelligent qos-aware routing mechanism with deep reinforcement learning. In *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, pages 329–332. IEEE, 2020.

[74] Md Billal Hossain and Jin Wei. Reinforcement learning-driven qos-aware intelligent routing for software-defined networks. In *2019 IEEE global conference on signal and information processing (GlobalSIP)*, pages 1–5. IEEE, 2019.

[75] Wei Zhou, Xing Jiang, Qingsong Luo, Bingli Guo, Xiang Sun, Fengyuan Sun, and Lingyu Meng. Aqrom: A quality of service aware routing optimization mechanism based on asynchronous advantage actor-critic in software-defined networks. *Digital Communications and Networks*, 2022.

[76] Lianming Zhang, Yong Lu, Dian Zhang, Haoran Cheng, Pingping Dong, et al. Dsoqr: Deep reinforcement learning for online qos routing in sdn-based networks. *Security and Communication Networks*, 2022, 2022.

[77] Faisal Naeem, Muhammad Tariq, and H Vincent Poor. Sdn-enabled energy-efficient routing optimization framework for industrial internet of things. *IEEE Transactions on Industrial Informatics*, 17(8):5660–5667, 2020.

[78] Junjie Zhang, Minghao Ye, Zehua Guo, Chen-Yu Yen, and H Jonathan Chao. Cfr-rl: Traffic engineering with reinforcement learning in sdn. *IEEE Journal on Selected Areas in Communications*, 38(10):2249–2259, 2020.

[79] Miao Ye, Linqiang Huang, Xiaofang Deng, Yong Wang, Qiuxiang Jiang, Hongbing Qiu, and Peng Wen. A new intelligent cross-domain routing method in sdn based on a proposed multiagent reinforcement learning algorithm. *arXiv preprint arXiv:2303.07572*, 2023.

[80] Cheng Wang, Luomeng Zhang, Zhong Li, and Changjun Jiang. Sdcor: software defined cognitive routing for internet of vehicles. *IEEE Internet of Things Journal*, 5(5):3513–3520, 2018.

[81] Manel Majdoub, Ali El Kamel, and Habib Youssef. Dqr: an efficient deep q-based routing approach in multi-controller software defined wan (sd-wan). *Journal of Interconnection Networks*, 20(04):2150002, 2020.

[82] Alessio Botta, Roberto Canonico, Annalisa Navarro, Saverio Ruggiero, and Giorgio Ventre. Ai-enabled sd-wan: the case of reinforcement learning. In *2022 IEEE Latin-American Conference on Communications (LATINCOM)*, pages 1–6. IEEE, 2022.

[83] Ikiomoye Douglas Emmanuel, Nigel Linge, and Steve Hill. Analysis of sd-wan packets using machine learning algorithm. In *2023 Conference on Information Communications Technology and Society (ICTAS)*, pages 1–6. IEEE, 2023.

[84] Jeffery A Clouse and Paul E Utgoff. A teaching method for reinforcement learning. In *Machine learning proceedings 1992*, pages 92–101. Elsevier, 1992.

[85] Lisa Torrey and Matthew Taylor. Teaching on a budget: Agents advising agents in reinforcement learning. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1053–1060, 2013.

[86] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[87] Heinrich Dinkel, Shuai Wang, Xuenan Xu, Mengyue Wu, and Kai Yu. Voice activity detection in the wild: A data-driven approach using teacher-student training. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:1542–1555, 2021.

[88] Mohammed Brahimi, Saïd Mahmoudi, Kamel Boukhalfa, and Abdelouhab Moussaoui. Deep interpretable architecture for plant diseases classification. In *2019 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*, pages 111–116. IEEE, 2019.

[89] Yan-Hui Tu, Jun Du, and Chin-Hui Lee. Speech enhancement based on teacher–student deep learning using improved speech presence probability for noise-robust speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27(12):2080–2091, 2019.

[90] Kshira Sagar Sahoo, Deepak Puthal, Mohammad S Obaidat, Anamay Sarkar, Sambit Kumar Mishra, and Bibhudatta Sahoo. On the placement of controllers in software-defined-wan using meta-heuristic approach. *Journal of Systems and Software*, 145:180–194, 2018.

[91] Shih-Chun Lin, Ian F Akyildiz, Pu Wang, and Min 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)*, pages 25–33. IEEE, 2016.

[92] Ruxia Liu, Sudan Li, Hong Wang, and Zhu Tang. A qos routing optimization algorithm based on hierarchical multi-controller coordination. In *2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, volume 1, pages 1820–1825. IEEE, 2019.

[93] Ying Zheng, Lixiang Lin, Tianqi Zhang, Haoyu Chen, Qingyang Duan, Yuedong Xu, and Xin Wang. Enabling robust drl-driven networking systems via teacher-student learning. *IEEE Journal on Selected Areas in Communications*, 40(1):376–392, 2021.

[94] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[95] Python Team. What is python? `https://www.python.org/doc/essays/blurb/`, 2023. Accessed on 25/04/2023.

[96] Pycharm: the python ide for professional developers. `https://www.jetbrains.com/pycharm/`, 2023. Accessed on 25/04/2023.

[97] Pytorch. `https://pytorch.org/`, 2023. Accessed on 25/04/2023.

[98] OpenAi Team. Gym documentation. `https://www.gymlibrary.dev/`, 2023. Accessed on 25/04/2023.

[99] NumPy Developers. What is numpy? `https://numpy.org/doc/stable/user/whatisnumpy.html`, 2023. Accessed on 25/04/2023.

[100] Jonathan Taylor statsmodels-developers. Josef Perktold, Skipper Seabold. Statsmodels. `https://www.statsmodels.org/stable/index.html`, 2023. Accessed on 25/04/2023.

[101] Niwat Thepvilojanapongy, Yoshito Tobez, and Kaoru Sezaki. Shr: Stateless hierarchical routing for dynamic sensor networks. In *The first International Workshop on Networked Sensing Systems (INSS'04)*, 2004.

[102] Andy Linton Dean Pemberton and Sam Russell. Ryu openflow controller. `https://nsrc.org/workshops/2014/nznog-sdn/raw-attachment/wiki/WikiStart/Ryu.pdf`, 2019. Accessed on 25/04/2023.

[103] Mininet Project Contributors. Mininet overview. `http://mininet.org/overview/`, 2022. Accessed on 25/04/2023.

[104] The University of Texas at Austin. Abilenetm. `https://www.cs.utexas.edu/~yzhang/research/AbileneTM/`, 2004. Accessed on 30/04/2023.

[105] Steve Uhlig, Bruno Quoitin, Jean Lepropre, and Simon Balon. Providing public intradomain traffic matrices to the research community. *ACM SIGCOMM Computer Communication Review*, 36(1):83–86, 2006.

[106] Chenyi Liu, Mingwei Xu, Yuan Yang, and Nan Geng. Drl-or: Deep reinforcement learning-based online routing for multi-type service requirements. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021.

A Appendix

A.1 Implementation code

```
def update(self, rollouts):
    obs_shape = rollouts.obs.size()[1:]
    action_shape = rollouts.actions.size()[-1]
    condition_state_size = rollouts.condition_states.size()[-1]
    num_steps, _ = rollouts.rewards.size()
    alpha=  1
    TeacherPenality=-5

    print(rollouts.actions.shape)
    print(rollouts.obs[:-1].shape)
    print(self.actor_critic)
    values, action_log_probs, dist_entropy, _ = self.actor_critic.evaluate_actions(
        rollouts.obs[:-1].view(-1, 1, *obs_shape),  # remove the last obs,[seq_len, batch=1, obs_shape]
        rollouts.recurrent_hidden_states[0].view(-1, self.actor_critic.recurrent_hidden_state_size),
        rollouts.condition_states[:-1].view(-1, 1, condition_state_size),
        rollouts.actions.view(-1, 1, action_shape))

    values = values.view(num_steps, 1)
    action_log_probs = action_log_probs.view(num_steps, 1)

    advantages = rollouts.returns[:-1] - values
    value_loss = advantages.pow(2).mean()
    masks = rollouts.masks

    action_loss = -(alpha*masks * advantages.detach() * action_log_probs + (1-alpha)*TeacherPenality).mean()
```

Figure A.1: Part of the Student model implementation.

```
Enabling Teacher ADVICE: Calculating the Shortest Path under the Bandwidth Constrains
'''
def TeacherDomainKnowledge_SHR(self, s, t, demand):
    fat = [-1] * self._node_num
    SHR_dist = [1e6] * self._node_num
    flag = [False] * self._node_num
    SHR_dist[t] = 0
    flag[t] = True
    cur_p = t

    while flag[s] == False:
        for i in self._link_lists[cur_p]:
            bandwidth = self._link_capa[i][cur_p] - self._link_usage[i][cur_p]
            if bandwidth >= demand and SHR_dist[i] > SHR_dist[cur_p] + 1:
                SHR_dist[i] = SHR_dist[cur_p] + 1
                fat[i] = cur_p
        cur_p = -1
        for i in range(self._node_num):
            if flag[i]:
                continue
            if cur_p == -1 or SHR_dist[i] < SHR_dist[cur_p]:
                cur_p = i
        if SHR_dist[cur_p] < 1e6:
            flag[cur_p] = True
        else:
            break

    if not flag[s]:
        return None
    path = [s]
    cur_p = 0
    while path[cur_p] != t:
        path.append(fat[path[cur_p]])
        cur_p += 1
    return path
```

Figure A.2: Teacher's domain knowledge SHR algorithm implementation.