DEVELOPMENT OF SCALABLE HYBRID SWITCHING-BASED SOFTWARE-DEFINED NETWORKING USING REINFORCEMENT LEARNING

BY

Max Blose

Submitted for the Degree of

Master Engineering in Telecommunications

in the Department of Electrical Engineering, Faculty of Engineering and The Built Environment, University of Cape Town

May 2024

Supervisor: Dr Lateef Adesola Akinyemi

Co-Supervisor: Prof Fred Nicolls

Declaration

I hereby declare that this dissertation is my original work and all the sources I used have been acknowledged and fully referenced. I further declare that I have not submitted this dissertation or any part of this research work to any university for other qualification or examination purposes.

I submit this work for examination for the Degree of Master of Engineering, specialising in Telecommunication. I grant the University of Cape Town full rights to reproduce this work for research related purposes.

I also declare that the submission of this work has been approved by my supervisor, Dr Lateef Akinyemi, and my co-supervisor, Prof Fred Nicolls.

Name: Maxwell Mazwi Blose Date: 29 May 2024 Signed by candidate

Signed by candidate

ABSTRACT

As the global internet traffic continues to grow exponentially, there is a growing need for cutting-edge switching technologies to manage this growth. One of the most recent innovations is Software Defined Networking (SDN), which refers to the disintegration of the infrastructure layer and the logically centralized control layer. SDN is a cutting-edge networking approach that provides network agility, programming flexibility, and enhanced network performance over traditional switching networks.

Even though SDN has some great benefits, there is a need to address and manage scalability challenges to guarantee optimal, scalable, and rapid data traffic switching within Service Provider network infrastructure, including Data Centres environments. These scalability issues are inherent to SDN's logically centralized control layer. Whenever a packet belonging to a new flow has to be transported, OpenFlow switch has to interact with the logically centralized SDN controller through southbound OpenFlow Application Programming Interface between OpenFlow switch and the logically centralized SDN controller. This results to an increase in communication overhead between the two instances. The control layer overhead traffic can impede scalability due to the controller's limited processing memory. There is therefore a strong incentive to enhance the scalability of SDN operations.

To address the SDN scalability issues identified by creating a scalable hybrid switching solution using machine learning algorithms. We propose an SDN OpenFlow model switch which collaborate with the traditional switch to represent a scalable framework of Hybrid Routing with Reinforcement Learning (sHRRL). We implement a reinforcement algorithm to randomly explore new routes and discover the most optimal path through the Q-learning algorithm. This primitive and model-free form of reinforcement learning utilizes the Markov Decision Process and the Bellman's equation to reiteratively update Q-values in Q-table for

every transition in the network environment state, until Q-function has converged to the best Q-Values. The greedy strategy is employed to guide the reinforcement learning agent in selecting the most suitable Q-values from the Q-Table. To ensure that the machine learning algorithm is able to discover a sufficient amount of possible routes and has a sufficient understanding of the network environment, sufficient training and evaluation episodes should be conducted.

The proposed hybrid switching methodology was benchmarked against the standard SDN OpenFlow switch in terms of network performance metrics, including average throughput and packet exchange transmission rates, CPU load, and delay, to compare the two switching approaches.

When statistically comparing the test results, it was observed that the number of packets exchanged by the hybrid switch was greater by more than sixty percent compared to the Open Flow switch which saturated first. The average throughput results demonstrate that the hybrid switching routing scheme achieves high throughput results. The first type of switch to reach saturation is the Open Flow switch, as it does not explore all available paths. Consequently, the hybrid switch is more efficient than the Open Flow Switch when it comes to CPU load. The average CPU load for the Open Flow switch is fifteen percent (15%) higher than for the hybrid Switch.

Our analysis of the simulation data suggests that the Q-learning-based reinforcement learning framework, sHRRL, enhances the performance of the hybrid switch when compared to the Open Flow switches. We are therefore of the opinion that the hybrid switching model proposed utilizing machine learning algorithms can address the scalability issues in the design of SDN controller networks, particularly in data centre environments where high switching speeds are of paramount importance.

Acknowledgments

I would like to express my sincere gratitude and appreciation to my supervisor, Dr Lateef Akinyemi and my co-supervisor Prof Fred Nicolls for their positive attitude and guidance toward the successful completion of this research work.

A lot of thanks to my family and particularly to my mother for her support, and encouragement.

Dedication

Finally, my thanks to the Almighty for everything He made possible…

Table of Contents

List of Figures

List of tables

List of Acronyms

API	Application Programming Interface
ARP	Address Resolution Protocol
AS	Autonomous System
BGP	Border Gateway Protocol
CLI	Command Line Interface
CPU	Central Processing Unit
DC	Data Centre
DDOS	Distributed Denial-of-Service
DIFANE	Distributed Flow Architecture for Network Enterprises
DiffServ	Differentiated Services
GUI	Graphic User Interface
HS	Hybrid Switching
HSDN	Hybrid Software Defined Network
IDE	Integrated Development Environment
IP	Internet Protocol
ISP	Internet Service Provider
JDK	Java Development Kit
MAC	Media Access Control
MDP	Markov Decision Process
MIPS	Million Instructions Per Second
NETCONF	Network Configuration Protocol
NIC	Network Interface Card
NOX	Network Operating System

OEM	Original Equipment Manufacturer
OVS	Open vSwitch
OSPF	Open Shortest Path First
OXM	OpenFlow Extensible Match
PES	Processor Elements
POX	Python-based Software
Q-Learning	Quality Learning
QoS	Quality of Service
RAM	Random Access Memory
REST	Representational State Transfer
RIP	Routed Internet Protocol
RL	Reinforcement Learning
SARSA	State Action Reward State Action
SBI	South Bound Interface
SDCs	Software-Defined Counters
SDN	Software Defined Network
SHRRL	Scalable Hybrid Routing with Reinforcement Learning
SNMP	Simple Network Management Protocol
SSL	Secure Socket Layer
SPT	SDN Path First
TD	Temporal-Difference
TPF	Traditional Path First
TCAM	Ternary content addressable memory
VLAN	Virtual Local Area Network
WAN	Wide Area Network

List of Symbols

N : Number of network devices

$\psi(N)$: Packet exchanged or throughput processed

$T(N)$: Latency or average response

$C(N)$: Control layer capacity to allocate flow rules

α : Alpha, for Learning Rate

ε : Epsilon for greedy strategy

γ : Gamma, for Discount Rate

π : Pi, for Optimal Policy

CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

The continuous evolution of the global internet is experiencing an alarming exponential growth daily. The observed high volume of internet traffic emanates from an increased number of bandwidth-hungry electronic gadgets and machines becoming vital tools of our lifetimes for conducting activities such as business and information search purposes. Subsequently, changing traffic patterns have moved Service Providers to match network traffic by increasing network bandwidth through the introduction of high-capacity transmission network hardware equipment such as switches, routers, and firewalls [1]. However, traffic patterns are becoming more complex. Rigid network topologies, incompatible routing policies, lack of scalability and the original equipment manufacturer (OEM) dependent hardware is seen as the limitations of traditional network devices.

Each network device has to be configured independently utilizing vendor or vendor specific configuration commands, requiring network operators to acquire highly skilled and usually expensive technical resources to configure original equipment manufacturer locked hardware. Most of the existing network hardware devices are vertically integrated. This implies that both the control layer and the data layer are embedded in every single network device. Therefore, impending flexibility, and limiting modernisation and progression of next-generation network infrastructure.

To provide an improved network performance, there is a compelling necessity to invent next-generation switching technology that matches today's revolving traffic patterns. Software-

defined networking (SDN) is one of the most recent innovations of the next-generation networks. It is characterized by three core components: the infrastructure or data layer, the application layer and the logically centralized control layer interconnected to the data layer through a southbound API [2]. Internet Service Providers (ISPs) have been able to take advantage of SDN solutions when it comes to high-speed networks, like data centres [3-4]. Software Defined Networking also enables infrastructure innovation in a form of a robust automated Quality of Service (QoS) model, traffic load-balancing, dynamic optimal routing, traffic segmentation and application aware routing in network infrastructure such as wide area network and Data Centre [5-6]. The software defined networking utilizes the OpenFlow protocol through the south bound interface to facilitate communication between the control plan and the network infrastructure layer (NIL), which is the plane where data is forwarded over the network devices such as router and switches.

However, SDN OpenFlow is not immune to challenges related to scalability due to its native detachment of the data forwarding plane and the control plane. Continuous communication between the controller and the forwarding plane through the south bound OpenFlow interface can lead to oversubscription and overloading of limited controller ternary content addressable memory [7-9]. Therefore, degrading controller performance in terms of flow table setup delay for the infrastructure layer devices.

Modern switches with high-performance capabilities can provide more than eight thousand (8,000) flows [7] with optimized memory capabilities that can handle more than a million flow table entries[8]. However, this can result in scalability issues due to the controller's limited computing capacity [9]. To address these issues, various frameworks are proposed as a way to offload overhead traffic towards the controller. For example, according to Ganjali in[10], multiple controllers can be used to distribute the flows.

Wang [11] suggests using DIFANE to create a scalable, flow-based network. He suggests taking some of the routing decisions away from the controller and putting them into predefined, special routing devices called authority switches. Further research efforts have been undertaken to identify a viable solution to alleviate traffic overhead burden off the SDN controller by delegating some of their functions to the data forwarding planes within the infrastructure layer. Further proposed research efforts include the design of a hybrid network control plane for a cloud Data Centre [12], as well as the design of an Application-aware customized forwarding planes within software defined networking [13].

In our research effort, we focus on the development and evaluation of hybrid switching with an intention of contributing by improving scalability challenges in Software Defined Networking. The proposed solution design introduces a machine learning algorithm to relieve a virtual centralized SDN controller from high control plane traffic volume sent by the data plane switches. By delegating some of the controller functionality to the infrastructure (data plane) switches and utilizing machine learning algorithms, we can improve scalability by reducing the delay in flow table setup by controller. By enabling backward compatibility and smooth collaboration between legacy switches and SDN controllers with the help of the reinforcement learning framework, we can improve network scalability. Our proposed solution can be deployed in a network environment, such as a data centre, where speed, high throughput capacity and low network latency are critical.

1.2 Objective of the Study

The purpose of this work is to achieve the following set of objectives:

- Review scalability challenges introduced by Software Data Networking's decoupling of control plane and data forwarding plane.

- To develop a Scalable framework of Hybrid Routing with Reinforcement Learning (sHRRL), collaboration mechanism between infrastructure layer (data plane switches) and SDN controller OpenFlow switch in a flow table setup.

- The overall performance of the proposed hybrid switching technique is technically and scientifically evaluated against the SDN OpenFlow switch and Hybrid switch based on network performance metrics including average throughput scalability, packet exchange transmission rate, and CPU load between the OpenFlow switch and hybrid switch.

- The results obtained from the numerical simulation are analyzed statistically and compared by plotting performance results in a graph.

1.3 Problem Statement

In traditional networks, network devices such as firewalls, routers, and switches have a data plane and control plane integrated into each network device. These devices can be interconnected to perform packet forwarding decisions based on routing protocols such as OSPF, RIP and BGP to mention a few. Traditional networks' limited scalability, limited automation, and rigid architecture are seen as a drawback to the future of internetworking growth [14-17].

It is out of these traditional network limitations that Software Defined Networking was introduced as a next-generation network to revolutionize internetworking. SDN brings about a number of key benefits over a traditional networking approach, due to its logical centralized control plane. SDN promises to improve network performance which includes security, programmability through API, and flexibility. Since SDN is developed with a virtually centralized controller, scalability is one of the primary challenges to be tackled [18].

SDN control plane scalability challenges are native to SDN due to the separation of the data plane from a logically centralized control plane structure. This means data plane devices like routers and switches no longer have the intelligence to make packet routing decisions. They have to depend on a logically centralized controller for traffic routing decisions. Constant communication between the control plane and the data plane can lead to additional communication overhead, which can lead to scalability bottleneck in the SDN controller. Thus, there is a huge need for improving SDN in terms of scalability. Hence, efforts are geared towards addressing this scalability problem by developing a hybrid switching for enhancement and improvement of scalability in the software defined-networking.

1.4 Research Methodology

In solving the SDN scalability challenge, different research methodologies have been considered. Similar research work has been explored with different methodologies such as the distributed controller approach, and the centralized controller approach which are considered less scalable [19]. The hierarchical Controller design approach subdivides the controller into different layers and different domains, where a controller has a full global view of the infrastructure layer with its domain. However, the hierarchical controller approach is likely to experience path stretch-related problems [20]. Research conducted by Original Equipment Manufacturers (OEMs) evaluated the backward compatibility of OpenFlow with network equipment, including Brocade switch [21-22].

In this study, a number of research methodologies such as supervised learning, unsupervised learning, deep learning, and reinforcement learning were considered. Supervised learning is best at labelled data with clear output to predict. Feedback is explicit and immediate as the model receives the correct output for each input. The objective function of this model is to minimize the loss or error between the predicted output and the true output.

Unsupervised learning methodology works with unlabelled data to discover patterns or structures. In The feedback is implicit and delayed as the model infers the output from the data distribution. Its objective function is to maximize the likelihood or similarity of the data.

We also considered deep learning method which enables computers to learn from big data. Deep learning is best at recognising patterns in data. It may be more difficult to configure and implement. However, we may consider deep learning technique in our future research work.

Reinforcement learning is useful when there is sparse or stochastic feedback and long-term goal to optimize the reward. This model is trying to maximize a reward signal instead of trying to find hidden structure. The objective function is to maximize the expected cumulative reward over time. reinforcement learning technique can learn from its own experience and adapt to changing environments. The model can handle complex and dynamic problems that require sequential decision making. While all the approaches have their strengths and weaknesses, reinforcement learning has advantages over other learning techniques in certain scenarios such as unlabelled data, sequential decision making, exploration vs exploitation, adaptability to dynamic environment, and reward to signal. In this research work, we adopt reinforcement learning as an algorithm to train new routes and establish the highest accumulative reward in a Hybrid-SDN environment.

The proposed methodology adopts a hybrid controller design approach due to its advantages of a collaboration mechanism, where infrastructure layer devices communicate with the SDN controllers in a flow table configuration. Computer simulations with the help of Cloudsim configuration library, maven dependency library and Java tools are utilized to evaluate scalability enhancement in SDN. Novel Hybrid switch-based reinforcement learning with computation techniques is utilized for optimal routing.

1.5 Scope of Study

Many studies have been conducted to explore SDN Controllers' performance and scalability [23-27]. In this study, we focus on the control plane than the data plane. Our study zooms into the following metrics:

- Control plane throughput, where we refer to the total number of requests processed per second by the controller.
- Control plane flow setup duration, referred to as latency or delay to respond to flow requests and,
- Control plane Central Processing Unit, related to the controller response time.

A Control plane scalability study can be classified into two distinct sub-academy approaches: Mechanisms-based and Topological-based. The Mechanisms-based approach focuses on the relationship between the various mechanisms employed to optimise controllers and the scalability issues associated with them. This approach is divided into two subcategories: Concurrency-based and Control Plane Routing Scheme-based.

Topology-based approaches take into account the association of topology architectures with scalability issues. This approach divides the design of controllers into two main categories: Centralized controller design and distributed controller design approaches. Subsequently, the design of distributed or flat controllers, the design of hierarchical controllers and the design of hybrid controllers are further subdivided. This study concentrates on two of these topology approaches: the Control Plane Routing Scheme-Based Optimization Mechanism Approach and the Hybrid Controller Design Topology Approach.

1.6 Research Questions

Our goal in this research project is to address the following research questions:

- How can a framework of Hybrid Routing with Reinforcement Learning (sHRRL) algorithm improve SDN control plane scalability.
- How efficient is the proposed algorithm and mechanism when compared to SDN OpenFlow Switch and other SDN scalability approaches proposed by other researchers' efforts?
- If applied through the SDN Hybrid controller design approach, how does the proposed scalable hybrid switching perform compared to the OpenFlow switch in terms of the following metrics:
    - Control plane throughput?
    - Control plane flow setup latency?
    - Control plane CPU response time?
    - Control plane packet exchange rate?

1.7 Dissertation Outline

This section provides a high-level summary of the research work by chapters, as illustrated in Figure 1.1. Chapter one is centred around the objectives of this research work. We discuss the background and motivation why this research effort is valuable. We also discuss the problem statement on scalability challenges introduced by the separation of the control place from the data plane. In Chapter two, we discuss SDN architecture in detail. Traditional network and SDN literature review, including related work.

Chapter three introduces the framework of reinforcement learning we propose to solve scalability challenges in SDN. In Chapter four, we present the proof-of-concept implementation where we apply the proposed framework of the Hybrid Routing with Reinforcement Learning (sHRRL) algorithm to improve SDN control plane scalability. The conclusions and Network simulation results are technically and scientifically analysed and

validated end-to-end through a comparison between the OpenFlow switch and our proposed Hybrid switching. Conclusions and future works are finally addressed in chapter five.

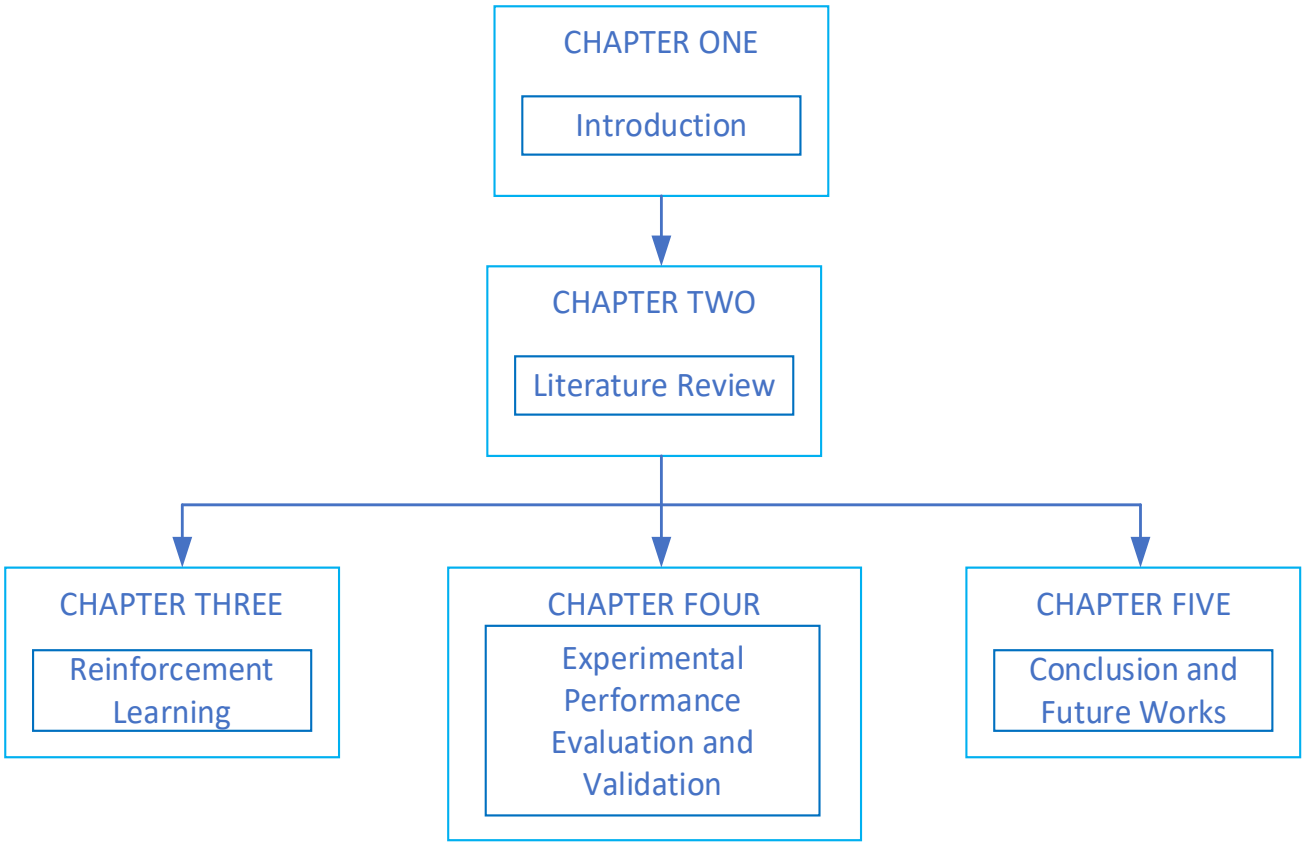

Figure 1. 1 Structure of the thesis

1.8 Research Publications from this Study

The following conference papers were produced from this research effort:

1. Max Blose, and Lateef Adesola Akinyemi, "Development of Scalable Hybrid Switching based Software Defined Networking." *In 2020 23rd International Symposium on Wireless Personal Multimedia Communications (WPMC2020), Japan, 19 - 26 October 2020*, pp. 1-5. IEEE, doi: 10.1109/WPMC50192.2020.9309465.

2. Max Blose, and L.A. Akinyemi, Fred Nicolls, and Neco Ventura, “Development of Scalable Hybrid Switching based SDN using Reinforcement Learning. “ *Southern Africa Telecommunication Networks and Applications Conference (SATNAC), KwaZulu-Natal, South Africa, August 27-29,* pp. 163-168, 2023

3. M. Blose, L. A. Akinyemi, S. Ojo, M. Faheem, A. L. Imoize and A. A. Khan, "Scalable Hybrid Switching-Driven Software Defined Networking Issue: From the Perspective of Reinforcement Learning Standpoint," in *IEEE Access*, vol. 12, pp. 63334-63350, 2024, doi: 10.1109/ACCESS.2024.3387273

4. Max Blose and Lateef Akinyemi “Scalable Hybrid Switching-Inspired Software Defined Networking Challenges: From the Perspective of Machine Learning Approach,” *2024 Conference on Information Communications Technology and Society (ICTAS),* Durban, South Africa, 2024, pp. 186-191, doi: 10.1109/ICTAS59620.2024.10507139.

1.9 Chapter Summary

The first chapter briefly introduces the technologies around this research work, the development of scalable hybrid switching based Software Defined-Networking using reinforcement learning. We also discuss the scalability challenges of SDN as a result of decoupling the control plane and the data plane in a network environment such as a Data Centre. Chapter one further outlines the objectives of the proposed hybrid switching approach to address SDN scalability challenges. The scope of work, research methodology and dissertation outline are presented in this chapter.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

In this chapter, we present an in-depth a literature review of the technologies underpinning this research work. These technologies include the Software Defined-Networking and the traditional switches to enhance scalability through a hybrid switching approach in a network infrastructure environment such as a Data Centre.

We delve into the various components of the Software Defined Networking, including the controllers, the interfaces, the switches, the OpenFlow protocols, and the traditional switches. We also review the potential scalability of Software Defined Network (SDN) based hybrid switching solutions that have been suggested in the past. The strengths and weaknesses of these solutions are examined. The chapter also leverages some of the concepts from the preceding schemes to illustrate the evolution of scalable SDN solution, based on the hybrid switching.

2.2 Traditional Networks

The traditional IP network is composed of two tightly connected planes, the control plane, and the data plane, which are integrated within a switch, or packet routing network device such as a router and switch. As illustrated in Figure 2.1, the control plane is responsible for forwarding data packets and makes forwarding decisions based on the switch table or the routing table. When a frame in layer 2 or a Packet in Layer 3 device is received from a switch port, the MAC address of the packet is stored in the Switch MAC address table, along with the

port number from which it was learned. The MAC address determines the manner in which packets are transmitted on the access media.

The MAC address is a unique physical address assigned to each switch or NIC [28]. The Ethernet frame header contains a source and a destination MAC address, which helps to distinguish between the sender and receiver of a data payload. Once a switch has become aware of other neighbouring switches' MAC addresses, it stores those learnt MAC addresses in a MAC address table, also known as a CAM table, along with the port number that was used to receive the MAC address. When an Ethernet frame is to be forwarded to a different network device, the first step is for the switch to check its CAM table for port and destination MAC addresses. If the target MAC address is not present, the switch will transmit the frame to all ports except the one to which it was initially received. Ethernet frame broadcasting is only performed once, after the switch has exchanged the first MAC address, it will be able to determine where to forward the packet the next time.

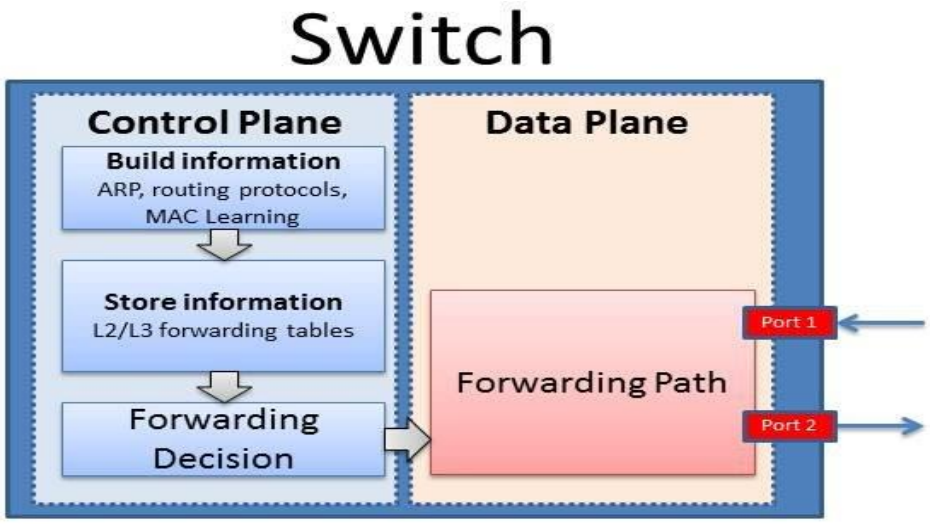

Figure 2. 1 Traditional switch architecture [29]

Fig.2.1 illustrates the distributed structure of a Traditional switch displaying the switch control and the data plane. Unlike Layer 3 routing devices which execute routing on a flow basis, traditional Layer 2 switches path selection is based on a target address.

In traditional networks with distributed control plane architecture, the routing protocols used can be either static or dynamic or both. Dynamic routing protocols include Routing information protocol (RIP) [30], Open Shortest path first (OSPF) [31], and Interdomain Protocol (BGP-4) [32]. Dynamic routing protocols employ route updates to find networks on layer 3 routing devices. Routing protocols, typically defined within the control plane through a Command Line Interface (CLI), specify a set of protocols that switches, and routers utilize to exchange routing information between neighbouring routing devices. As an example, a non-proprietary link-stake routing protocol such as OSPF establishes a global network by periodically forwarding multicast link state information to all OPSF-speaking routing devices that are part of the same Autonomous System (AS). Dynamic routing protocols are capable of rapidly detecting topological alterations and network state transitions, including router interface failures, as well as calculating new loop-free routes, reconverging, and propagating new link status information database across all OSPF configured routers in the same autonomous system.

2.2.1 Traditional Networks limitations

As computer networking technology advances and the majority of services transition to cloud-based environments, traditional network architecture is characterized by a static and intricate structure, manual device configuration through the use of the command line interface, heterogeneous network policies, lack of security, lack of scalability, and hardware that is

vendor-dependent. Network operators must configure individual network switches individually using vendor-specific commands.

This necessitates network operators to invest in highly qualified personnel or technical expertise, which proves to be costly in addition to the technical resources required to configure the vendor-specific hardware. In a similar way, traditional switches are vertically-oriented, meaning that the control plane and data plane are physically connected to the routing devices.

This vertical integration impairs flexibility and impedes the development of future computer networks, thus hindering innovation. Software-defined networking and network function virtualization (NFV) technologies offer the potential to overcome many of the limitations of traditional networks in terms of scalability, availability, flexibility, and programmability, as well as security.

2.3 Software Defined Networking

In the preceding chapter, we briefly discussed the concept of software defined networking. In this section, we will delve further into the details of SDN technology. Each component will be described in detail including will interaction with other layers via northbound or southbound interfaces. Additionally, we will address the scalability issues associated with SDN, as well as other proposed research initiatives.

2.3.1 SDN Architecture

Software-Defined Networking is a cutting-edge innovation technology developed by the Open Network Foundation(ONF) [25]. It advocates the decoupling of control plane and data plane, with a control plane logically centralized.

The primary objective of Software-Defined Networking is to facilitate the development and advancement of future computer networks through network architectures that are programmable, flexible, high availability, scalable, and secure across the entire architecture. SDN introduces forwarding decisions that are flow-based as opposed to destination-based forwarding decisions in traditional networks. The technology is primarily characterized by the following three main layers [33] illustrated in Figure 2.2.

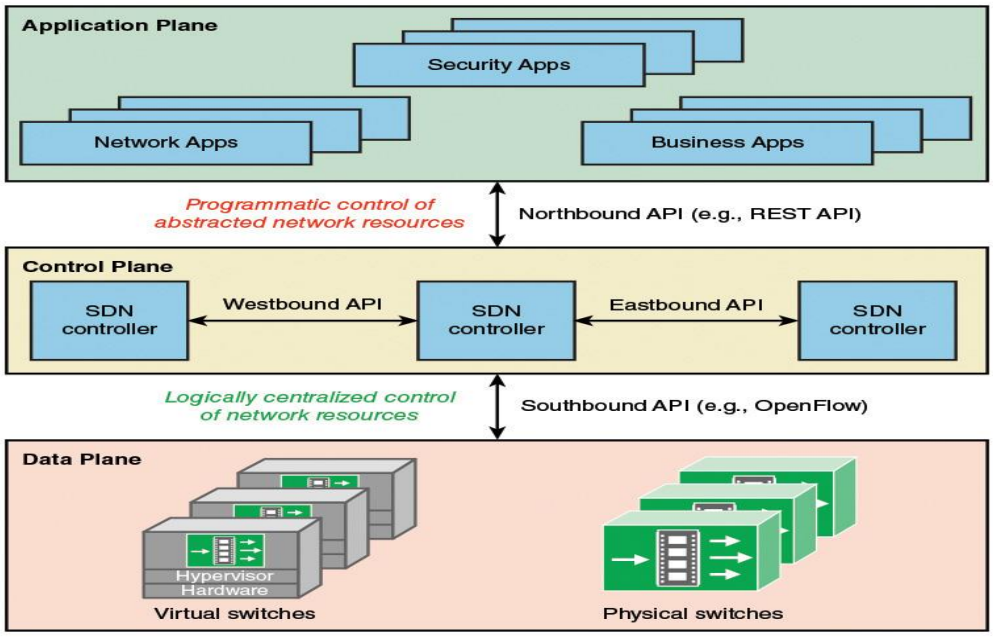

Figure 2. 2 SDN network architecture [29]

The Data Plane Layer represents the infrastructure layer, where virtual and physical switches are physically positioned in the network infrastructure. This infrastructure layer interacts with a logically centralized controller through the southbound API interface utilizing OpenFlow protocol.

The Control Layer is a logically centralized layer responsible for the control and distribution of routing policies to virtual and physical network resources such as virtual machines, switches,

routers, and firewalls. The control plane layer can host multiple SDN controllers interconnected through the westbound API and the eastbound API. This is the central layer which interlink the infrastructure layer and the application layer through the OpenFlow southbound API and the REST northbound API respectively.

The Application Plane is the top layer representing network applications, security applications and business applications. It forms part of the network response management and interconnect to the lower layers such as the control layer and the infrastructure layer through the northbound API protocols such as the Representational State Transfer API, SNMP, and NETCONF protocols [34], for the network resources programmability.

2.3.2 SDN Programming Interfaces

The northbound and the southbound interfaces of the Software Defined Networking architecture are commonly referred to as APIs, and are found between the infrastructure layer, the control layer, and the application layer [35]. Both eastbound API and the northbound API interfaces are utilized for adjacent SDN controllers to form the SDN controller cluster.

2.3.2.1 Northbound APIs

Northbound APIs are communication interfaces that facilitate the connection between the application layer and the control layer [36]. One of the primary roles of the northbound API is to facilitate applications in managing network resources and the infrastructure layer devices. Since there are no standardized specifications, vendors have different northbound API specification features.

2.3.2.2 Southbound APIs

The southbound API interface which utilizes the OpenFlow protocol is located between the Infrastructure Layer and the Control Layer [36]. This interface facilitates communication between the Software Defined Networking controllers and the data plane layer network devices for control policies, optimum routes exploration and maintenance of the data layer flow tables. The OpenFlow protocol is one of the most widely used southbound APIs.

2.3.3 Open Flow Overview

OpenFlow is a standard protocol for southbound interfaces defined by the Open Network Foundation Standard[37]. This standard protocol is intended for communication between a Software Defined Networking controller and a data plane switch. An OpenFlow-compatible switch is referred to as an OpenFlow switch and is composed of three main components: a flow table, a secured channel, and the OpenFlow protocol [38]. The most recent version of OpenFlow (Version 1.5), which is the successor to the first version, was published in 2009. This version introduced significant refinements to the flow table, the flow entry, the scalability of the system, and a variety of new functions, as illustrated in table 2.1[38].

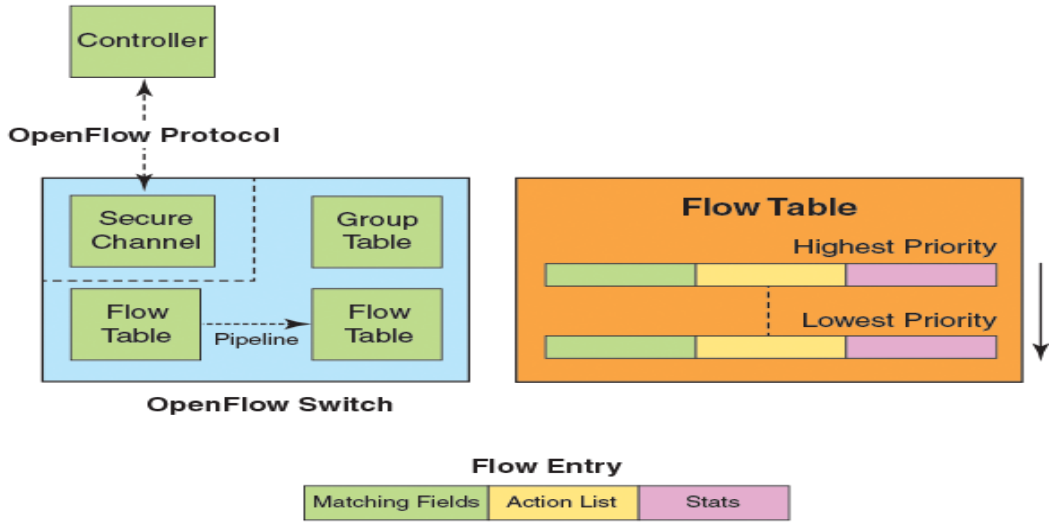

Figure 2. 3 OpenFlow switch architecture [37]

Fig. 2.3 illustrates how the OpenFlow switch communicates with an OpenFlow controller via a secure tunnel using SSL. Each flow table within the switch comprises a sequence of flow entries, each of which contains a match field, a counter, and a sequence of instructions to be executed on matching packets. The OpenFlow Switching Architecture is composed of OpenFlow Switches, OpenFlow Controllers, and Secure Channels between the Switching Switches and the Controller.

The OpenFlow switch is composed of three main segments [37]. The first segment is the flow table, which contains match and action fields associated with flow entries. The second segment is related to communication channel, which serves as a link between the controller and the switch to transmit commands and packets. The third segment is the OpenFlow protocol which is facilitates the communication channel between an OpenFlow controller and routing and switching devices within the data plane of the infrastructure layer.

Table 2. 1 OpenFlow major version Release [38]

OpenFlow Version	Major Feature	Reason for release	Use Case
Version 1.0 to 1.1	Moving from one table structure to many tables	To prevent the explosion of the flow entry	
	Group Tale	Enable action sets to be applied to a collection of flows.	Failover, Link Aggregation, and Load balancing
	MPLS and VLAN Support	Full support	
Version 1.1 to 1.2	OXM Match	Enhanced matching flexibility	
	Multiple Controller	High availability, load balancing and scalability	Controller Failover, and load balancing support
Version 1.2 to 1.3	Meter table	Add QoS and DiffServ capability	
	Table miss entry	Provide flexibility	
Version 1.3 to 1.4	Synchronized table	Enhanced table scalability	MAC Learning and Forwarding
	Bundle	Extended switch synchronization	Multiple switch configuration
Version 1.4 to 1.5	Egress Table	Processing can be performed in the output port.	
	Scheduled bundle	Further enhanced switch synchronization	

2.3.4 SDN Challenges

There is no doubt that software-defined networking is a new paradigm that offers numerous advantages to cloud and computer networking innovation and development. However, SDN also faces a number of challenges. Below, we will discuss some of the challenges associated with SDN.

2.3.4.1 Availability Challenges

Data plane devices, such as switches, rely on a logical centralised controller to modify traffic flow rules. In the case of an SDN centralised controller deployment approach, the control layer may be a single point of fault, leaving the infrastructure layer devices isolated if connection to control layer is lost. Therefore, it is recommended to adopt a distributed controller approach to guarantee high level of availability for the control layer.

2.3.4.2 Flexibility Challenges

Flexibility is the capability of a network to respond quickly to unexpected network changes, such as a route link failure or reconvergence, within a short period of time. One of the primary difficulties of Software Defined Networking is the ability to efficiently manage packet routing and streamline flow processing. With SDN, flexibility is achieved by prioritizing flow rules among the switches and the controllers through the OpenFlow interface, with action being taken based on the matched packet header fields. The packets can be either forwarded out of the switch port or amended or dropped, with the flow rules being stored in the hardware memory of the OpenFlow switch. Unfortunately, TCAM memory can be slow in updating the flow table, which can lead to bottlenecks in networks with extensive dynamic routing policies and compromised flexibility in the control plane. These bottlenecks could be eliminated with the next-generation of switches.

2.3.4.3 Security Challenges

Most widely recognized security threats associated with the Software-Defined Networking systems are rooted in the traditional network architectures. The design of Software-Defined Network systems necessitates a separation between the control layer and the infrastructure layer, which has led to a number of security issues.

An example of an attack that SDN is vulnerable to is Distributed Denial of Service in the control layer [39]. This attack can be initiated from different infrastructure layer devices and queries that flood the control layer. With a high volume of queries sent to the controller, a latency or packet drop may occur due to these inundated queries. This type of attack can be mitigated by implementing multi-SDN controllers, where a single controller can be assigned a master role and load balance queries from the switches to the additional SDN controllers.

Man-in-the-middle (MIME) attacks can exploit communication messages between the controllers and the infrastructure layer devices. The attacker can alter the flow rules between the controllers and the data layer switches, and then take control of the data layer and the controllers. To protect against this type of attack, it is recommended to implement enhanced integrity, robust authentication protocols, and digital signature encryption of the controller-to-OpenFlow switches communication.
The application layer is also vulnerable to cyberattacks due to its vulnerability in terms of programmability. An attacker can hijack network traffic by accessing SDN secure applications and compromise the network infrastructure in the process. Best security practices must be followed.

2.3.4.4 Scalability Challenges

Although Software Defined Networking was created to address the complexity and limitations of traditional network architecture in order to meet the unprecedented growth of cloud services and virtualization, SDN is not without its own scalability issues. This is largely due to the architecture of SDN, which decouples the control layer and the infrastructure layer.

One of the primary drawbacks of Software Defined Networking is a centralized controller, which can lead to significant scalability issues. For instance, research conducted by Erickson et al [41] implies that a network of one hundred (100) switches can result in a ten million flow rate in a worst-case scenario. Additionally, according to the study by Kanduala el al [40], a cluster of 1,500 servers can receive an average of 100,000 flows per second. These findings suggest that the control plane of the SDN is likely to experience scalability issues due to its logically centralized controller nature.

The separation of the infrastructure layer and the control layer is a foremost source of scalability issues in SDN. As the infrastructure layer must rely on a centralized controller to make decisions regarding the setting of flow rules, continuous communication between the two separate layers is necessary. Depending on the architecture of the SDN network, whether it is centralized, distributed or hierarchical, such continuous signalling messages between the two separate layers add significant overheads, which can become a bottleneck and a scalability issue for the control layer. A centralized controller design that processes a large number of requests can cause scalability issues due to the growth of the infrastructure layer devices.

This type of network expansion can result in an increase in the number of flow requests that must be handled by the controller, and a potential bottleneck due to the

controller's central processing unit's limited computing capacity and memory capacity to accommodate the additional flow requests. An example of a controller that may not be suitable for a medium-to-large network set-ups, such as a Data Centre network infrastructure, is the NOX controller, which can only manage a maximum flow request rate of 30 thousand requests per second[42].

A number of studies have been conducted on the scalability challenges of data planes in Software Defined Networking with a particular focus on buffers and memory capacity, as well as processing power, to name a few [43-45]. However, in the scope of this research, SDN data plane is not included. Instead, the focus is on the scalability improvement of the SDN control plane in a Data Centre network environment. The main focus of the research is on the throughput of the control plane, which includes the amount of requests handled per second by the controller, the time taken to set up flow table and the delay associated with the response to the flow requests, as well as the central processing unit (CPU) associated with the controller response time.

2.4 Scalability Metrics

The concept of scalability is a concept that can be interpreted in different ways depending on the target audience. There is no precise definition of what scalability is. However, in the context of computing, it can be interpreted as the capacity of a system, network device, or process to handle an increasing volume of work. It can also be interpreted as the capacity to expand to accommodate future growth [46].

A variety of research studies have been proposed to assess the scalability of systems through the use of various metrics. The majority of these studies [47-52], suggest Isospeed and Isoefficient Scalability metrics in homogeneous, heterogeneous, or hybrid settings. For

instance, a study conducted by K. Wang and Z. Xu[50] proposed that an algorithm-machine combination is expressed as scalable if the speed-efficiency achieved on a given system remains constant as the number of processors increases, regardless of the size of the system. The speed of the system remains linear if the size of the problem is incremented at the speed-efficiency function. In a subsequent proposal, X. H Sun et al [47] provide a metric to characterize the scalability function of a homogeneous system, defined as:

$$\psi(p, p') = \frac{p'W}{pW'} \tag{2.1}$$

In equation 2.1, p and p' refer to the number of processors initially and incrementally available on the system, respectively.

W and W' respectively, refer to the initial and incrementally available problem size.

Kumar et al [52] define Isoeffeciency as the capacity of the parallel machines to maintain a consistent parallel efficiency when increasing the system or network size and challenge size. They further define parallel efficiency as an increase in parallel speed over a given number of processors presented as:

$$E = \frac{S}{p} \tag{2.2}$$

The speedup is expressed as a ratio of the size of the problem (W) to the number of times it can be executed in parallel, expressed as (T_p).

$$S = \frac{W}{T_p} \tag{2.3}$$

where $T_p = \frac{W + T_0(W,p)}{p}$ with $T_0(W,p)$ being an extra communication overhead [48].

From the Software Defined Networking perspective, the separation of the infrastructure layer and the control layer is in fact a distributed structure. Consequently, a productivity-based metric of the distributed system scalability can be employed to measure the scalability of the software defined networking control layer or controller [53]. In order for a distributed or heterogeneous network to be considered scalable, it is necessary to maintain network infrastructure efficiency, as the scale of the network infrastructure is flexible. In a distributed network infrastructure, F(N) is defined as per equation (2.4) [54]:

$$F(k) = \varphi(N) \times \frac{T(N)}{C(N)} \tag{2.4}$$

The several numbers of the infrastructure layer devices are referred to as $\varphi(N)$, while the number of processing requests by the controller, for network flow set-ups is referred to as *T(N)*, and the cost or capacity of the controller to process the flow setup requests is referred to as *C(N)*. In a case where the SDN controller scalability metrics transition from N_2 to N_1 , it can be defined as:

$$\psi(N_1, N_2) = \frac{F(N_2)}{F(N_1)} \tag{2.5}$$

2.5 Related Work

This section reviews the relevant literature and prior suggested solutions to address the scalability challenges associated with software defined networking. We focus on the methods that relate to different design architectures for software defined networking controllers, and related work about scalability of the SDN, which continues to be one of major topics of research in the software defined networking field.

A number of research papers have been published proposing various approaches to address SDN scalability issues. These include DIFANE [55], DevoFlow [57], Devolved Flow [56], and Kandoo [56,58]. DIFANE is a distributed flow architecture that reduces network overhead by reducing control traffic sent across the OpenFlow protocol interface. In DIFANE, wildcard rules define protocols on how traffic is routed amongst the network switches.

DevoFlow suggests an approach of reducing the network overhead traffic towards the controller through a collaboration model where the network devices process a limited portion of traffic flow rules, whilst the controller takes care of giant packets flow rules. Authors of Software defined converters in [59], Hyper Flow [60], Maple [61], ONIXN [62], and Maestro [63] have proposed similar solution as DevoFlow to address SDN scalability challenges.

Different SDN controller designs such as the centralized, distributed, hierarchical, and the hybrid controller design approaches have been proposed in previous research efforts. In *centralized controller design strategy*, a single controller takes full responsibility of managing all network devices within its domain. The centralized controller model is not considered as scalable compared to other approaches. The *distributed controller method* allocates a sub-set or a small domain of the network devices to each controller, as described in [64-66]. The distributed model further splits the controllers into those which have local management responsibility and those with global domain management responsibility role. The controllers assigned local management function have limited knowledge or exchange of routing and control information about other local controllers. The controllers at global layer have a privilege of a comprehensive view of the entire network estate of the controllers and the network devices at the infrastructure layer. The local controllers have to exchange full routing tables and network status updates through the global controllers. This can lead to unnecessary additional control traffic overhead and high latency in processing of flow rules at the local controller level.

The next controller design method is based on the *hierarchical approach* where the controllers operate at different domain layers. The highest layer takes responsibility of being a root controller with a full view of all lower layers including the infrastructure layer. However, the hierarchical controller model has a disadvantage of length flow paths [59] and high latency in flow rules set-up. The last design strategy is related to the *hybrid controller approach*, where the legacy network devices collaborate with the SDN controller in flow rules set-up.

The idea and basic analysis of software-defined networking were investigated in the works [67-70] through the application of both traditional and table-based reinforcement learning. Remarkable outcomes were attained. Nevertheless, actual neural learning was not used. Additionally, a variety of fields were covered in [71–82] by the application of deep learning, machine learning, and metaheuristic algorithms like genetic algorithms (GA) and deep sleep optimizers (DSO) to optimisation problems in radio wave propagation, wireless communication, and healthcare systems. Nevertheless, the open and hybrid switch systems were never taken into account by the techniques used in the research. This study compares the open-based switch and hybrid-inspired form utilising a range of performance parameters, including throughput, packet exchange rate, CPU load, and latency.

Machine learning is a research proposal that shows how networks can learn from experience and get better at what they do compared to static and traditional routing algorithms. Zhang et al in [83] showed how a Reinforcement Learning technique could be used for routing, link cost calculation and path selection. But their proposal only focused on delay optimization. Research efforts [84] and [85] suggested improving traditional routing protocols by using the SDN innovation. But their work didn't really use network operation information to get smarter routing. Martin et al in [86] employed machine learning algorithms approach in software

defined networking, but the extra data labelling they added introduced extra overhead, routing complexity and reliance on traditional routing protocols.

Our research effort recommends the development of a Machine Learning algorithms to create a Scalable Framework for Hybrid Routing with reinforcement learning (sHRRL), where an interaction mechanism between a legacy switch and the OpenFlow switch introduces a hybrid switching in the flow table set-ups, in accordance with the findings of [67] following a similar research methodology.

2.6 Chapter Summary

This chapter provides an in-depth literature review of the technologies underpinning this research work. We provided a general overview of the traditional switching architecture and its limitations, such as the bundled control and the data planes, inconsistent policies, scalability issues, and hardware vendor dependency. Additionally, detailed Software Defined-Networking components were discussed.

We looked at the SDN controllers, interfaces, switches, and the OpenFlow protocol, as well as different architectures of SDN controllers that have been utilized to solve the SDN controller problems. We also looked at some related work that suggested different ways to improve SDN scalability. Next, we'll look at our idea for a Machine Learning algorithm and methodology to craft a Scalable Framework for a Hybrid Routing using Reinforcement Learning (sHRRL), an interaction mechanism between a legacy switch and the OpenFlow switch introducing a hybrid switching in a flow table setup.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

In this chapter, a reinforcement learning algorithm applied in the development of scalable hybrid switching based on software-defined networking, is presented. The test environment chosen to emulate the data centre network for this research is discussed. Chapter three also presents our methodology used in the proposed solution, showing how data will be gathered and the tools utilized.

3.2 Reinforcement Learning

Reinforcement learning is an area of machine learning that focuses on how something might act in an environment to maximize some given rewards. This machine learning approach relates to the capability of an agent or systems to learn from previous experience and subsequently enhance performance. Reinforcement algorithms study the behaviour of subjects in such an environment and learn to optimize that behaviour. For example, it is natural to train a dog by rewarding it when it responds appropriately to commands. In this animal example, the owner of a dog can throw away a ball in the garden, and a dog will run after the ball, pick it up and take it back to the owner. The result will be an appreciation in the form of dog food from the owner as a result of the action by the dog. However, if a dog happens not to go after the ball, an alternative action could be some form of punishment.

We can relate this animal reinforcement example to figure 3.1, where the dog can be referred to as an *Agent.* The throwing away of the ball is a *State* for the *Agent.* An *Action* will be when the dog runs after the ball. The dog owner and the garden represent the *Environment* where the

dog runs after the ball thrown by the owner, and food given to the dog after running for the ball is a *Reward*. It is the environment that gives the agent a reward as an output to the agent's current state and action as an input. An alternative action to a reward is a *Penalty*, or no food given to the dog, as a punishment for not running after the ball.

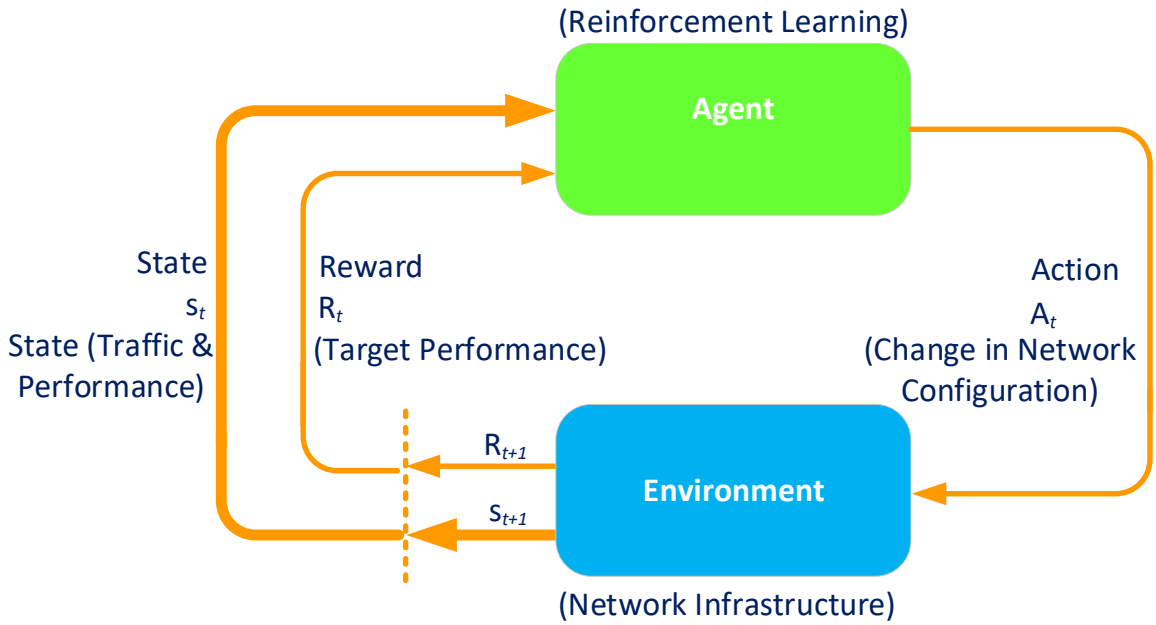

Figure 3. 1 Reinforcement Learning

In computer networking, machine learning is a viable alternative in resolving routing challenges associated with traditional networks' convolutional routing protocols and static routing. The ability for reinforcement learning models to learn the intent and reward based on data, is the main motivation for adoption in this work as an algorithm for training new routes and setting the highest cumulative reward in Hybrid-SDN environment. This learning technique can learn from its own experience and adapt to changing environments, as well as handle complex and dynamic problems that require sequential decision making.

The proposed scalable hybrid routing with reinforcement learning algorithm is designed with the capability to choose an alternate path through training the agent over time. The problem is characterized by all possible State *S*, an Action *A* likely to be selected based on current state, and the Reward *R* function. A reinforcement learning algorithm can learn from a delayed reward by executing a specific order of actions before being rewarded. As depicted in Fig 3, at time *t*, the environment is in State S_t, the agent observes the current state and selects action A_t. The environment transitions to the state S_{t+1} and grants the agent a reward $R_{t+1} = R(S_t, A_t)$, and the next State S_{t+1} with probability $T(S_t, A_t, S_{t+1})$ The process starts over again and repeats until the agent has reached the optimal goal of maximizing the reward. To learn such an order of actions, the agent has to determine and understand the action which was taken for a reward to be awarded. this problem is referred to as *temporal credit assignment* [87]. Therefore, this machine learning concept is concerned with the optimal order of actions through exploitation and exploration trade-offs to learn about an environment to enhance the rewards reiteratively.

Two different types of machine learning model approaches are considered, supervised and unsupervised learning. In the supervised learning model, a machine is trained using data that has well-labelled input and output datasets during the training phase. In this algorithm, a level of human interaction or supervision is required for data labelling purposes, before being used to train the model. Unsupervised learning model approach processes unlabelled or raw data. As its name suggests, it requires less or no human interaction, and is used during initial exploration phase for better interpretation of datasets.

Reinforcement learning algorithms are different to the supervised learning algorithm approach, as reinforcement learning algorithms are directed to the goal by maximizing the rewards received reiteratively. In the next sections, we go beyond the agent and the environment and discuss the main elements of reinforcement learning and its formulation.

3.3 Value Function

Reinforcement learning algorithms consider *value function* as a model to decide on the best routes for the agent to choose, and estimation on how good it is for an agent to be in a given state. We can mathematically express the reinforcement learning function the agent experiences as a Markov Decision Process (MDP) [88]. This process is conceptualized as a mathematical model of decision-making in a dynamic system, where the outcome can be random or governed by an agent which makes successive choices reiteratively [89]. A Finite Markov Decision Process, where there is a finite number of states and actions, has the following characteristics :

A finite set of Action A,

A finite set of States S,

A reward R characterized by $R : S \times A \rightarrow \Re$, and

A state s defined by function $T : S \times A \times S \rightarrow \Re$, where $T(s, a, s')$ is the probability of advancing from state s to s' when taking action a. Policy is a critical factor of reinforcement learning that identifies the behaviour of the agent in a specific environment. It is defined as a function that maps a given state s to probabilities of selecting each action a in the state s [89]. It is denoted by the symbol π. An agent follows a policy, meaning if an agent follows policy π at time t, then $\pi(a|s)$ is the probability that $A_t = a$ if $S_t = s$. That is to say, at time t, under policy π, the probability of taking action a in state s is $\pi(a|s)$. The agent's main objective is policy enhancement through increasing the expected return received over time. The expected return or reward can be calculated in diverse ways, based on identified agent's assignment. The agent can be reset to an initial state at the end of each episode. In the case of such episode tasks,

an expected return is obtained through equation (3.1), The equation depicts a sum of rewards received over finite horizon *h*.

$$R_t = \sum_{k=0}^{h} r_{t+k+1} \tag{3.1}$$

However, other tasks tend to be infinite, and may need to be resolved through future reward discount as per equation (3.2).

$$R_t = \sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} \tag{3.2}$$

where γ represents a discount rate between $0 \leq \gamma < 1$. The purpose of discount rate is to discount the future return an agent can expect by undertaking the current reward value. The state value function denoted by $V^{\pi}(s)$ under policy π, is the expected reward when starting state s is followed by policy π subsequently. In Markov Decision Process, we can mathematically define $V^{\pi}(s)$ as:

$$V^{\pi}(s) = E_{\pi}\{R_t \,|s_t = s\} \tag{3.3}$$

where $E_{\pi}\{\}$ denotes the expected reward when the agent follows policy π, during time step t. In the case of a discounted infinite horizon case, we have:

$$V^{\pi}(s) = \left\{ \sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} \mid s_t = s \right\} \tag{3.4}$$

We maximize V^{π} for the current and future states through the optimal value function V^* as per equation (3.5)

$$V^*(s) = \max_{\pi} V^{\pi}(\forall s) \tag{3.5}$$

The optimal policy can be defined as the best action to take at each state of the environment in order to maximize the returns over time [90], and can be express by equation (3.6) as:

$$\pi^* = arg \max_{\pi} V^{\pi}(\forall s) \tag{3.6}$$

The state transition probabilities T and the reward function R of the Markov Decision Process also serve as models for the dynamics of the environment [89]. To identify the ideal value function, we can use value iteration, a dynamic programming technique [89]. Once we know the ideal value function, we can achieve the optimal policy π^* by providing the maximum value function for all of the immediate successor states, which we describe in equation (3.7) as:

$$\pi^*(s) = arg \max_{a} V^*(s') \tag{3.7}$$

Where s'denotes the successor of the state s.

In reinforcement learning problems, an agent does not have access to the environment elements within the frame of the transition probabilities T. In this way, we cannot utilize dynamic programming strategies in our SDN scalability approach. Within the next sections, we look at reinforcement learning algorithms based on dynamic programming [91], where we do not have earlier information on the environment flows. Instep, an agent needs to learn from the environment through the rewards experienced by taking diverse activities.

3.4 Temporal Difference Learning

In this section, we focus on the issue of learning the ideal approach without culminating information about the environment. One of the alternatives we can utilize to learn about the environment is through Temporal-Difference learning [92]. Richard S et al [93], define Temporal-Difference (TD) learning as the model-free class of reinforcement learning

technique that utilizes bootstrapping for learning purposes, meaning that TD update estimates based on other estimates or the current estimate of the value function. It can be deciphered as an unsupervised learning procedure with the idea of predicting the total reward anticipated over time. The intention of Temporal-Difference learning is to utilize a distinction between an anticipated reward and the actual reward to overhaul the agent about an expected return. This learning technique is the product of Monte Carlo [94] and Dynamic Programming learning strategy. Monte Carlo strategy alter their estimates after the ultimate output is received, while Temporal-Difference learning alters expectations to coordinate afterward, more precisely, expectations of the future before the ultimate result is known.

One of the options of solving the Temporal-Difference learning challenge is through gradationally evaluation of value function V^{π} as the environment transitions to the next state. In this approach we take models from the network environment, based on Monte Carlo methods [94], and update the reinforcement learning agent on the present estimations related to Dynamic programming methods [91].

Temporal-Different training algorithms update present estimate $V(s_t)$ through value function estimates of temporally successive states [92]. If we observe the next transition state s_t of the value function V^{π}, we can express the estimated value function of the current state s_t through equation (3.8) as follows:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{3.8}$$

There are different learning algorithms that can be used for Temporal-Difference learning method. The most common algorithms are State Action Reward State Action (SARSA) algorithm [95], and the notable Q-Learning algorithm technique. SARSA utilizes an on-policy strategy where it learns from the agent's actions by keeping track of the anticipated

return in the form of a reward for any transition to a new state after an executed action by the agent, and updates the expected reward based on the actual reward received. SARSA updated are executed through Temporal Difference [92]. In Fig 3.1, we illustrate the environment transitions from one state to another and the rewards granted to the agent after taking an action. This process starts over again for the next time step reiteratively until the agent has reached its optimal goal.

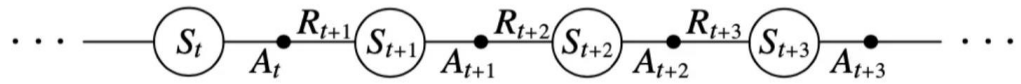

Figure 3. 2 SARSA State-Action Transition [96]

On the other hand, the Q-Learning algorithm utilizes an off-policy control technique to differentiate between a learning process and an acting policy. Both methods operate in a finite environment. SARSA is more conservative than the aggressive Q-learning. The Q-learning rely on the optimal policy for leaning than the near-optimal policy learning by SARSA. In this research effort, we focus on an off-policy TD control algorithm using a Q-learning approach. The common point between the Q-learning and the SARSA algorithms is a similar value function both algorithms reach as their exploration probability ϵ approaches "0". In the next section, we explore the Q-learning approach, which is in line with our research effort.

3.5 Q-Learning

In the previous section, we briefly presented the idea of Q-learning as another reinforcement learning method employed for an optimal policy in the Markov Decision Process. We are going to unpack this further, by looking at the control problem of finding the

optimal policy π through training the agent to learn the optimal Q-values for each action selected at each state observed by the agent.

According to Watkins [87], Q-learning is defined as a primitive and model-free form of reinforcement learning. It is called off-policy because of its Q-learning feature, which learns from actions outside the current policy and takes actions randomly that result in no policy being needed. This learning method utilizes the Bellman equation [97] presented in equation (3.9) to iteratively update the Q values of each state function pair until the Q function has converged to the optimal value of Q function Q*. We refer to this learning approach as value reiteration [98].

$$V(s) = \max_{a}\left(R(s,a) + \gamma \sum_{s'} s'(P(s,a,s')\,V(s')\right) \tag{3.9}$$

In Bellman equation (3.9), we represent:

$s =$ A particular state in our network environment,

$a =$ An action to be taken to route a packet,

$s' =$ A state to which a packet is routed from state s,

$\gamma =$ A discount rate,

$R(s,a) =$ A returned reward value after a transition to a state s as a result of an action a selected by the agent.

$V(s) =$ A value of being in a specific state s, could be the best taken chosen by our novel switch to route a packet.

If we assume that the agent is mindful of the transition probability T of the network environment and the assumption is true, then the agent ought to be able to select the action that leads to traffic being routed to the next node, through the optimum value function defined in equation (3.7), including the immediate return.

The fundamental challenge is that we normally do not know a model of the environment, subsequently the agent does not know in advance, which action a can lead to which states s. This can be resolved by characterizing the defining a value-function $Q^{\pi}(s,a)$, as the esteem of taking action a for observed state s resulting in optimal policy π. The newly defined value-function is named the action-value function, and $V^{\pi}(s)$ is the state-value function. In this manner, one can mathematically characterize $Q^{*}(s,a)$ in terms of $V^{*}(s)$ as depicted in equation (3.10), as the anticipated reward for choosing action a amid state s.

$$Q^{*}(s,a) = R(s,a) + \gamma \sum_{s'} (P(s,a,s')\, V^{*}(s')) \tag{3.10}$$

We can utilize the condition in equation (3.10) to measure the quality of a specific action selected by the agent. In equation (3.5) we specified that the state-value function $V^{*}(s)$ for policy π, is the expected reward when the initial state s is followed by policy π. Therefore, we rewrite the state-value function $V^{*}(s)$ as:

$$V^{*}(s) = \max_{a} Q^{*}(s,a) \tag{3.11}$$

As a result, we can recursively rewrite equation (3.10) as follows:

$$Q^{*}(s,a) = R(s,a) + \gamma \sum_{s'} \left(P(s,a,s') \max_{a'} Q^{*}(s',a')\right) \tag{3.12}$$

According to Watkins and Dayan [87], whenever Temporal-Difference (0) is utilized in predicting the expected reward of state s under policy π, Q-learning successively evaluates the best action-value function of $Q^{*}(s,a)$. In the research work, we utilize Temporal-Difference as the technique to back our proposed novel switch in computing the Q-values concerning the future states transitions in the hybrid network over time. If we contemplate our switches in a position of receiving a packet and wanting to forward that packet to a certain path, then a switch should already know from its routing table or flow table, the Q-value of choosing

the action of forwarding a packet in a chosen path based on a routing table. In that case, we recognize that our network environment has stochastic characteristics and the returns in the form of a reward received by the agent could differ when compared to the initially observed network state. To capture this change and the Temporal-Difference, we calculate a novel $Q^*(s,a)$, from a similar condition in equation (3.12) and subtract the previously $Q^*(s,a)$ from it.

$$TD_t(s,a) = R(s,a) + \gamma \sum_{s'} \left(P(s,a,s') \max_{a'} Q^*(s',a') \right) - Q^*(s,a) \tag{3.13}$$

This in turn gives us a new $Q^*(s,a)$. Equation (3.13) gives us a Temporal-Difference within the Q-values which further makes a difference to acquire random transitions within the environment which may be imposed. The Q-function $Q^*(s,a)$ can be transformed into equation (3.14) as follows:

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha TD_t(s,a) \tag{3.14}$$

Alpha α represents a learning rate that determines how swiftly the network environment can adapt to unplanned routing changes forced by the network environment. $Q_t(s,a)$ is the present state Q-value and the $Q_{t-1}(s,a)$ is in the past registered Q-value path in the routing table. Therefore, if we can mathematically characterize $TD_t(s,a)$ into equation (3.15) which defines the Q-learning algorithm.

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha \left(R(s,a) + \gamma \max_{a'} Q(s',a') - Q_{t-1}(s,a) \right) \tag{3.15}$$

In figure 3.2 we present the Q-Learning algorithm adopted for our research effort. The algorithm is based on the selection of an appropriate exploration method derived from quality Q. An approach that assures a state-action pair would be attempted over time will be sufficient. One of the methodologies utilized is ε -greedy. This methodology can be utilized to exploit

the network environment by selecting an action with the largest Q-value where the current state has a probability of $1-\varepsilon$ and a random explore the network environment by choosing an action based on the current state with insignificant probability ε. In a situation where an agent selects an action which rewards an optimal Q-value, it is then exploiting the already-known data about the network environment. Otherwise, it is explored in the case of a random action result. In the next section, we deep-dive into a trade-off, whether to explore or to exploit.

***Algorithm* 1**: $\boldsymbol{Q-Learning}$
1: Algorithm parameters: step size α $(0 \in [0,1]$, small $\varepsilon > 0)$
2: Initialize Q(s, a), for all s $\in \delta^+, a \in A$ (s), arbitrarily except that Q(terminal,·) $= 0$
3: Loop for each episode:
4: Initialize s
5: Loop for each step of the episode:
6: Choose a from s using policy derived from Q (e. g. , ε − greedy)
7: Take action a, observe R, s'
8: Q(s, a) ← Q(s, a) + α$[R + \gamma max_a$ Q$(s'$, a) − Q$(s, a)]$
9: s ← s'
10: Until s is terminal

Figure 3.2 Estimating Q^* with Q-learning

3.6 Exploitation and Exploration

In the previous section, we left our discussion of Q-learning with a question on how an agent chooses between exploiting or exploring the environment to choose its action.

Exploitation and exploration are the two different approaches utilized by the agent to choose the best action to maximize the rewards. The agent is likely to explore or find out about the environment when a random action is chosen. During exploration, the agent gives up some immediate rewards to maximize future rewards. The agent exploits the environment when it chooses the most likely action from the Q-table to maximize immediate rewards. To hit a balance between exploration and exploitation, an approach called *epsilon greedy strategy* [87] is used. Epsilon greedy strategy defines an exploration rate ϵ to be initiated to one. The exploration rate ε is the possibility that the agent will randomly discover the environment rather than exploiting the already known information about the network environment. When an exploration rate $\varepsilon = 1$, it is 100% guaranteed that the agent will begin by exploring the environment by selection an action at random. As the agent begins to know more about the environment, the exploration rate ε, will start to decline by some rate which will result in the probability of an agent selecting exploration becoming less favourable at the beginning of each episode. The agent will start being greedy in exploiting the environment after learning more about the environment through exploration.

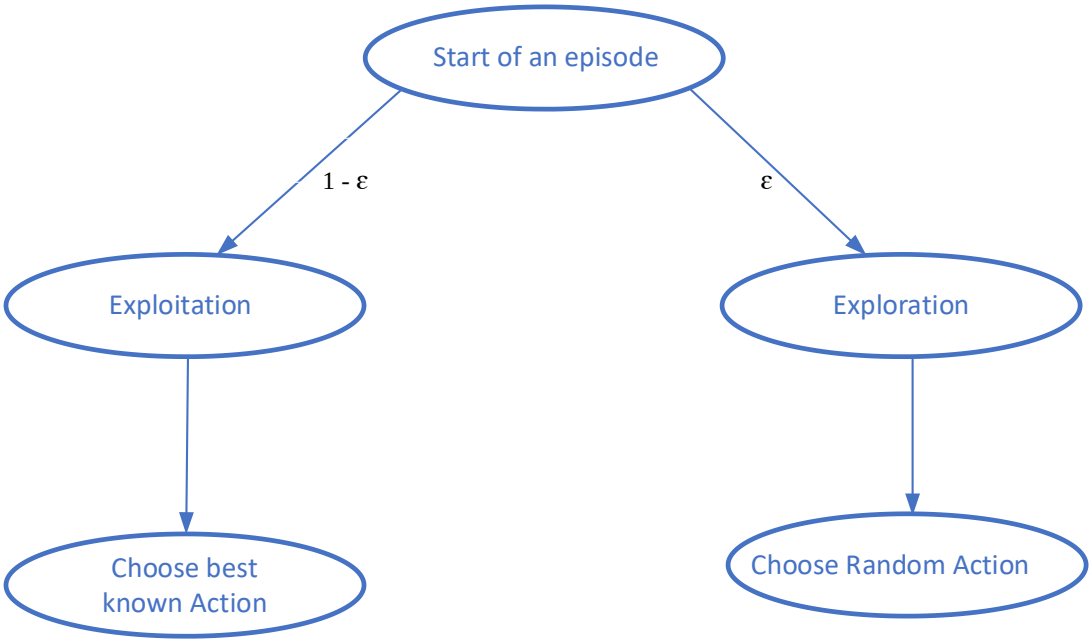

Figure 3. 3 Exploration and Exploitation

To decide if the agent has to select a random action through exploration or to take advantage of the previously successful approach and exploit the network environment, we ought to produce an arbitrary integer between zero and one. On the off probability that the created integer is larger than epsilon, at that point the probability of an agent opting to exploit the network environment is high. This implies that the agent is likely to select the next action based on the most rewarding return and explore the network environment by randomly selecting the next possible action and exploration of the current network environment state within the next time step.

3.7 Q-Learning Algorithm

At the start, when a packet or frame is forwarded between a source node and a destination node or server, the agent or switch has no idea where to route the packet. Thus, the Q-values from each action selected for the current observed environment state will all be reset

to zero, as the switch does not know the current environment or network routing status and expected reward. The Bellman equation discussed in equation (3.10) is utilized to update the Q-table with Q-values for a chosen action during the current observed network environment state as close as possible to the optimal $Q^*(s,a)$ of Bellman equation.

Algorithm 2*: *Epsilon − Greedy Q − Learning Algorithm

1: **Data**: α: *Learning rate*, γ: *Discount rate*, ϵ: *A small number*
2: **Results**: $A - Table\ containing\ Q(s,a)\ pairs\ defining\ estmated\ optimal\ policy\ \pi^*$
3: *Initialize* $Q(s,a)$ *arbitrarily, except* $Q(terminal)$;
4: $Q(terminal) \leftarrow 0$;
5: *for each episode do*
6: *Initialize state s*
7: *for each step in the epistode do*
8: *do*
9: $a \leftarrow select - action(Q,s,\epsilon)$;
10: *Take action a, then observe reward R and next state s′*
11: $Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma \max_a Q(s',a) - Q(s,a)\right]$;
12: $s \leftarrow s'$;
13: *while s is not terminal*;
14: *end*;
15: *end*;

Figure 3. 4 Q-Learning Algorithm

We utilize the Q-table as a storage for the Q-values generated during each action selected after the environment has transitioned to the next state. The Q-tables are updated iteratively utilizing value iteration. The Q-table is updated per episode, meaning whenever a new frame or packet is received. At a later stage, when a similar packet or frame is received towards a similar destination, the switch can examine the Q-table, also called the flow table, as

a reference for the next action to be selected to route the packet from the source node to the destination node.

```
29    class Qtable {
30                ArrayList<ArrayList<Double>> qValues = new ArrayList<ArrayList<Double>>();
31                ArrayList<ArrayList<Double>> Rewards = new ArrayList<ArrayList<Double>>();
32                int actions;
33                int states;
34
35                private final double learningRate = 0.1;
                  private final double rewardDecay = 0.9;
                  private final double greedy = 0.9;
38
39                public Qtable(int actions, int states) {
40                  this.actions = actions;
41                  this.states = states;
42
                    initializeQtable();
                    initializeRewardsMatrix();
45
46                }
47
```

```
48  public void updateQTable(int currentState, int nextState, int action, double gamma){
49      double reward = takeAction(currentState);
50      qValues.get(currentState).set(action, qValues.get(currentState).get(action)
51       + learningRate * (reward + (gamma * Collections.max(qValues.get(currentState)) - qValues.get(currentState).get(action))) * 1.0);
52  }
```

Figure 3. 5 Q-Table Algorithm

3.8 Q-Learning Routing Algorithm

We implement a reinforcement learning algorithm to randomly explore new routes and discover the optimal path through the Q-learning algorithm which assists in finding the maximum collective reward. In our network environment, traffic congestion cannot be predicted in advance. Therefore, we take advantage of the Q-learning algorithm to craft our sHRRL to solve the link congestion challenges through dynamic routing in our network environment. We propose a hybrid switching method which utilizes the sHRRL routing approach to identify and choose an available alternate routing path based on the learned optimal path.

```
class Qlearning_routing_algorithm {
  public static void main(String[] args) {

    Qtable one = new Qtable(9, 4);
    Random random  = new Random();

    for(int i=0; i < 100; i++){
      int state  = random.nextInt(4);
      int action = random.nextInt(9);
      int nextState = random.nextInt(4);
      one.updateQTable(state, nextState, action, 0.9);
      one.print();
    }
  }
}
```

Figure 3. 6 Q-learning routing algorithm

First, we initialize the Q-table with some arbitrary values $Q(s, a) = 0$ for all the state-action (s, a) pairs. Then we iterate for *one thousand* episodes. At each episode we start from the initial state s and perform k steps if we do not terminate the state. At any state s, we first query ε-greedy policy ε. Then we apply the action and observe the returned reward and the subsequent state s'. After an action has been executed, the Q-learning update rule is applied to update the old Q-value from the previous Q-table state. The Q-learning algorithm process is illustrated through a flow chart depicted in Fig 3.7.

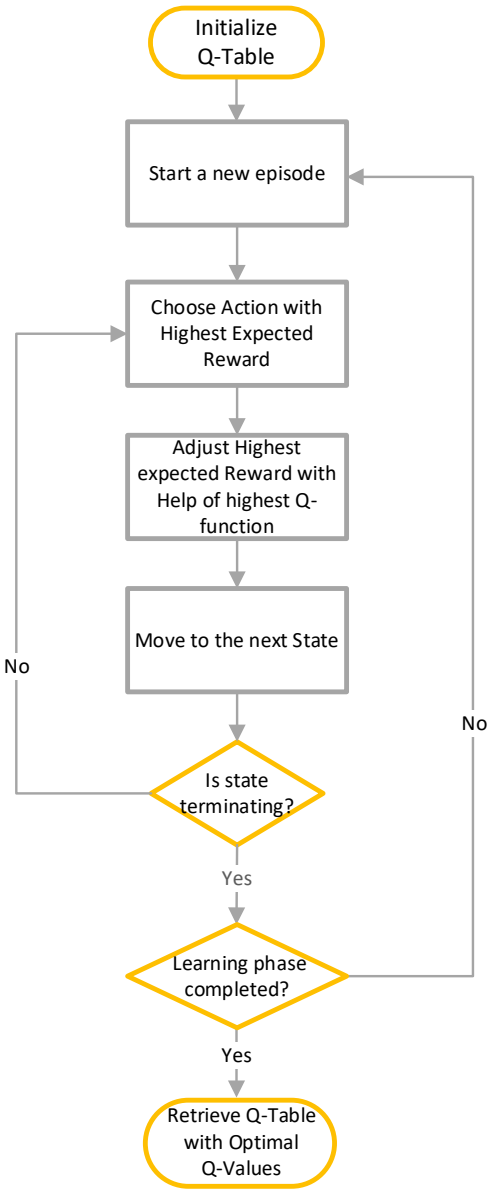

Figure 3. 7 Q-Learning Routing Algorithm Flowchart

3.9 Hybrid SDN Design Model

In this research work, we follow the same concept and similar findings presented in [83] for the Hybrid-SDN model topology design depicted in the communication diagram between the controller and switches as per Figure 3.8. We implement a system of collaboration

between the infrastructure layer of traditional switches and the software-defined networking OpenFlow switches.

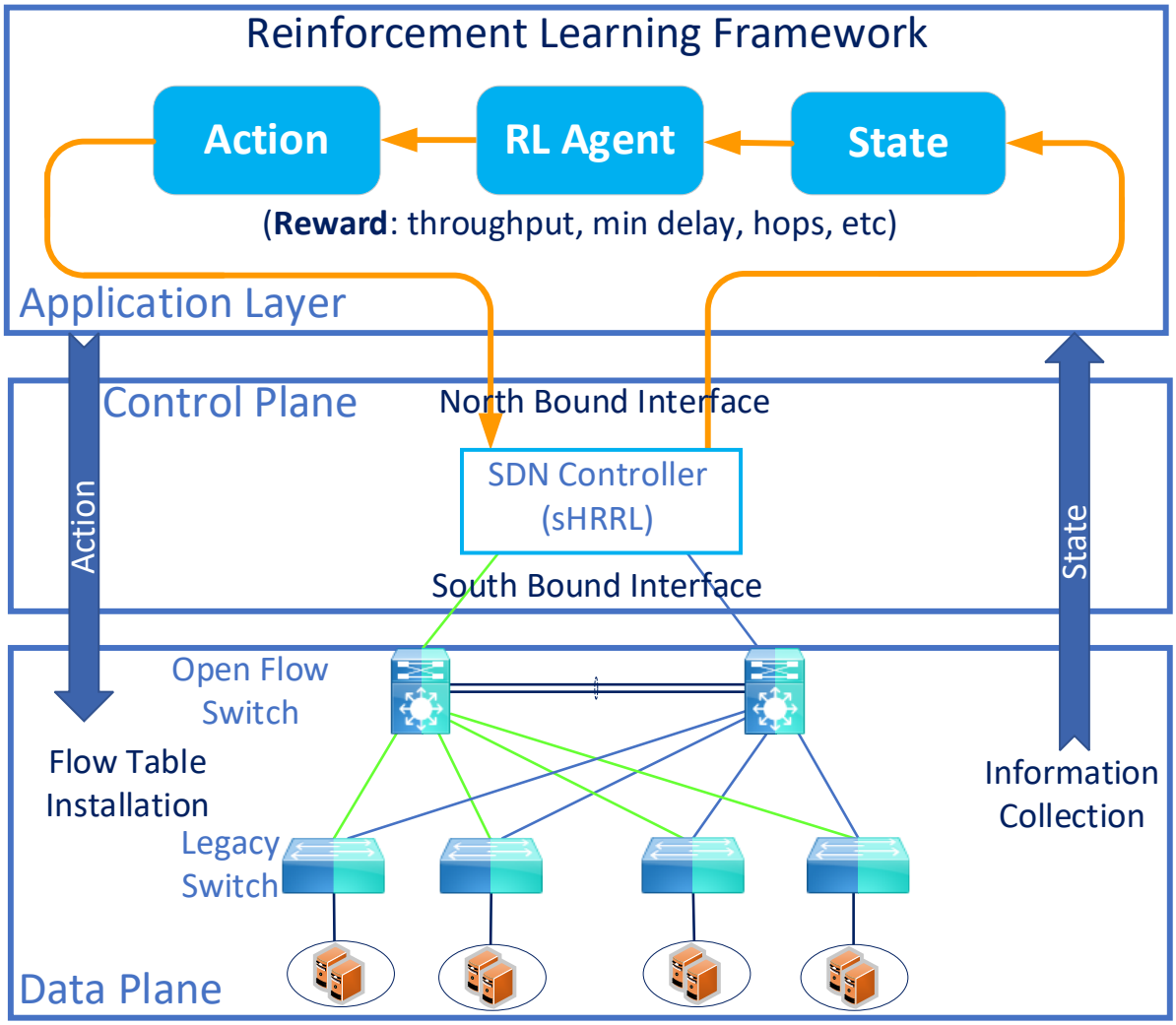

Figure 3. 8 Hybrid SDN topology with hosts, switches, controller, and RL framework [70]

To begin, the virtually centralized controller discovers network paths by discovering the network under its control domain. The discovered network paths between source and destination nodes are planned with the idea of selecting the shortest routes first. At the switches, routing paths are established as proactive rules in the flow table. The proposed machine learning algorithm's initial phase is proactive rule placement. A rule specifies proactive actions that should be taken after the proposed learning algorithm has observed the current network

environment state of an incoming packet at the switch to determine the shortest and least congested path between source and destination nodes. The algorithm can instruct the nodes to discard the data packet or forward it to a particular port towards the destination node. The controller utilizes threshold value to predict different types of network services during the second phase of packet forwarding. These threshold values for multiple network services are chosen based on network service offerings.

3.10 SDN Controller Design

In this design, the controller provides a set of pathways that comprises all feasible data packet forwarding routes between the source and the destination requesting switches. When the initial collection of paths between the source and destination nodes is empty, but an alternate path does exist, a different set of routes is constructed between the two transmitting nodes. All network routes congested beyond the allocated bandwidth threshold value are eliminated from the data traffic forwarding table after monitoring each link utilization status. Only network paths with the highest path Q-values are selected for traffic forwarding with the aid of the proposed Q-learning model which reinforces the best path routing process for forwarding network switches.

3.11 Chapter Summary

In summary, our research methodology primarily focuses on reinforcement learning, which is an area of machine learning that focuses on how something might react in an environment to maximize some given rewards. This machine learning approach relates to the ability of programs to learn from experience and subsequently improve performance. We adopt

a reinforcement learning algorithm to train new routes and establish the highest accumulative reward in a Hybrid-SDN environment.

We mathematically expressed the reinforcement learning function the agent experiences as a Markov Decision Process. The Temporal-Difference (TD) was discussed as a model-free reinforcement learning methodology which learns by bootstrapping. It is an unsupervised learning technique which predicts the total reward expected over time. The TD is a combination of the Monte Carlo and the Dynamic Programming learning techniques. The Monte Carlo technique adjusts their estimates after the final output is received, whereas the Temporal-Difference learning adjusts predictions to match later, more accurate, predictions of the future before the outcome is known.

We defined Q-learning as a primitive and model-free form of reinforcement learning, which uses the Bellman equation, to iteratively update the Q-values for each state-action pair, until the Q-function has converged to the optimal Q-function Q_*. We used a Q-learning algorithm to design our proposed sHRRL to find the highest collective reward in an environment. We also introduced a trade-off between exploration and exploitation. These two different approaches are employed by the agent to decide whether to explore or exploit at each time step. To make a choice, we generate a random number between zero and one. If the generated number is greater than epsilon, then the probability of an agent choosing its next action through exploitation is high. This means that the agent will select the action with the highest Q-value through exploration by randomly selecting its action and exploring what happens in the environment in the next time step. We also proposed a Hybrid-SDN model topology design, which depicts communication diagram between the controller and switches. In the next Chapter, we present performance evaluation and validation of our proposed framework of Scalable Hybrid Routing with Reinforcement Learning (sHRRL) algorithm to improve SDN control plane scalability.

CHAPTER 4

PERFORMANCE EVALUATION AND VALIDATION

4.1 Introduction

This section looks at how our idea of using a scalable hybrid routing algorithm with a reinforcement learning can help make SDN control planes more efficient. A series of experiments were conducted to assess the simulated scalability of the sHRRL network. In the framework of our solution, a reinforcement learning was employed to train new flow-forwarding routes and a Q-Learning algorithm was employed to identify the highest aggregate reward in the data centre network environment. In our comprehensive explorations, we concentrate mostly on the three primary parameters. Firstly, we look at the control plane throughput, which is how many requests the controller can handle per second. Secondly, we look at how long it takes the control plane to set up the flow tables, which is called latency or how long it takes for the controller to respond to the flow requests. And finally, we look at the control plane's CPU, which is related to the controller's response time.

4.2 SDN Controller Design

In this model, the controller specifies a set of paths that contains all possible packet-forwarding paths from the request switch to the target node. When the first set of paths is empty and there are no existing routing paths between the source and target nodes in the routing table of the switches, an agent must observe the current network status and select an action to construct routes between the sending switch and the target node that are less congested.

In this design, the controller provides a set of pathways which comprises all feasible packet-forwarding routes from the requesting switch to the destination node. When the first collection of paths is empty, where we don’t have an existing route between the source and the

destination nodes in the switches' routing table, an agent has to observe the current network state and select an action to construct less congested routes between the transmitting switch and the receiving node.

The SDN controller is designed to keep track of all the links and find the routes that are overloading beyond the bandwidth limit. The suggested algorithm agent keeps track of the current state of the network and finds an alternative routes with the highest Q-values from the Q-table and allocates new routes in the form of the flow table for the network switches. We utilized the Q-learning model to ensure that the agent can handle all the packet-forwarding requests and select the best routes through the reinforcement process, represented as sHRRL.

4.3 Model Components

The hardware and software components utilized in this experiment are presented in Table 4.1. To meet the objectives in this research effort, an open-source framework, CloudSim [98], was utilized to model and simulate a Cloud Computing infrastructure in the form of a Data Centre environment. CloudSim is written in Java and is developed by the Cloud Lab organization. In the CloudSim simulation, a scalable switching-based SDN was developed using hybrid routing and reinforcement learning techniques through the CloudSim plugin. The IDE Java tool was utilized as a development environment, with the aid of Maven dependency.

Table 4. 1 Testbed hardware and software components

Hardware	**Software**
Lenovo Laptop X13 Yoga	Windows 11 Enterprise
16 GB RAM	64-bit operating system
500GB Hard Drive Disk	CloudSim
11th Gen Intel(R) Core(TM) i5-1145G7	IDE Tool – NetBeans 12.5, Java JDK 17.0
2.60GHz 2.61 GHz, x64-based processor	IDE Tool – Eclipse

	Maven Dependencies: CloudSim 4.0 jar, json.jar, junit-14.13.2.jar, Hamcrest-all-1.3.jar

4.4 Test Bed to Enhance Scalability

In order to improve scalability, we employed the host machines and the hybrid switches managed in the Data Centre with the assistance of the CloudSim configuration library and the maven library. Additionally, the novel hybrid Switch-based reinforcement learning combined with Q-learning computation approach was employed for optimal routing.

Figure 4. 1 Hybrid SDN Topology Construction

Hybrid SDN Topology, as shown in Figure 4.1, has been designed and implemented in CloudSim.

As depicted in Fig 4.1, Hybrid SDN Topology has been constructed and configured in CloudSim. The Topology has been implemented with appropriate simulation parameter hostname, bandwidth, storage, MIPS memory, processing elements and RAM.

```
cd D:\sundar NS3 backup\28083\SDN_Cloud\SDN_Cloud; "JAVA_HOME=C:\\Program Files\\Java\\jdk-17.0.1" cmd /c "\"C:\\Program
Running NetBeans Compile On Save execution. Phase execution is skipped and output directories of dependency projects (wi
Scanning for projects...

----------------------< org.cloudbus:cloudsimsdn >----------------------
Building cloudsimsdn 1.0
--------------------------------[ jar ]---------------------------------
The POM for org.cloudbus.cloudsim:cloudsim:jar:4.0 is missing, no dependency information available

--- exec-maven-plugin:3.0.0:exec (default-cli) @ cloudsimsdn ---
{"nodes":[{"size":10,"bw":100,"name":"vm1-0","mips":100,"type":"vm","pes":1,"ram":256},{"size":10,"bw":100,"name":"vm3-1
==============WORKLOAD=============
start, source, z, w1, link, dest, psize, w2
-1.0,vm0-0,0,1,default,vm0-1,1000000000000000,1
-1.0,vm0-0,0,1,vm0-0vm0-1,vm0-1,1000000000000000,1
-1.0,vm0-1,0,1,default,vm0-0,1000000000000000,1
-1.0,vm0-1,0,1,vm0-1vm0-0,vm0-0,1000000000000000,1
-1.0,vm1-0,0,1,default,vm1-1,1000000000000000,1
-1.0,vm1-0,0,1,vm1-0vm1-1,vm1-1,1000000000000000,1
-1.0,vm1-1,0,1,default,vm1-0,1000000000000000,1
-1.0,vm1-1,0,1,vm1-1vm1-0,vm1-0,1000000000000000,1
-1.0,vm2-0,0,1,default,vm2-1,1000000000000000,1
-1.0,vm2-0,0,1,vm2-0vm2-1,vm2-1,1000000000000000,1
-1.0,vm2-1,0,1,default,vm2-0,1000000000000000,1
-1.0,vm2-1,0,1,vm2-1vm2-0,vm2-0,1000000000000000,1
-1.0,vm3-0,0,1,default,vm3-1,1000000000000000,1
-1.0,vm3-0,0,1,vm3-0vm3-1,vm3-1,1000000000000000,1
-1.0,vm3-1,0,1,default,vm3-0,1000000000000000,1
-1.0,vm3-1,0,1,vm3-1vm3-0,vm3-0,1000000000000000,1
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  2.181 s
```

Figure 4. 2 Hybrid SDN Topology Virtual Machine Nodes

In addition, Hybrid SDN topology has been designed and implemented in CloudSim, as illustrated in Figure 4.2. The topology has been employed with the necessary simulation parameters, such as virtual machine names, network bandwidth, storage, MIME, POP, RAM, etc.

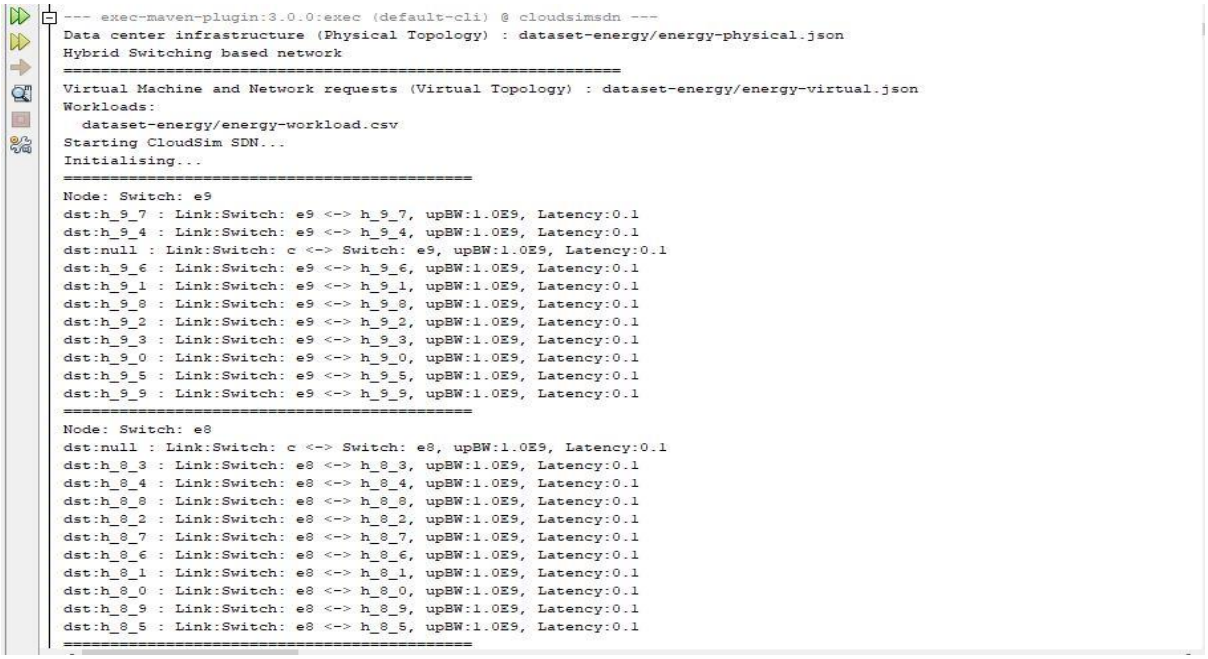

Figure 4. 3 Virtual Connection with Switches

In this CloudSim topology model, all the hosts were virtually connected to switches in a Data Centre, as shown in Fig. 4.3. We use the Hybrid Controller design methodology in this work because it has the benefits of allowing the infrastructure layer devices to work together with the SDN controllers in a flow tables set-up. We also use computer simulations with the CloudSim configuration library, Maven dependency library, and Java tools to see how scalability can be improved in SDN. We use new Hybrid switch-based reinforcement learning with computational techniques to find the best route.

Figure 4. 4 SDN topology construction

4.5 State space design approach

The reinforcement learning agent collaborates with the software defined networking controller to manage the network environment in a holistic manner. The agent evaluates the primary performance metrics such as delay, bandwidth, and throughput at a specified time t. The agent's observation is focused on the network's state s_t and the rewards r_t for each transition in the network environment state. Each state s_t contains a routing table which has an existing feasible path for every packet flow. Each path in the routing table is represented by a Q-value in the state space.

4.6 Action Space Design Approach

In the proposed action space design strategy, the RL agent selects an action a_t based on the network environment state s_t and the expected corresponding reward r_t. The action is

set according to feasible paths or routes, which include the current best feasible path. Based on the values calculated and the network state, one of these feasible routes is selected to replace or proceed routing packets via the current path.

4.7 Rewards Design Approach

In this work, we take into account the delay for a reward when sending a packet from a source node to a destination node. To evaluate the optimal action a_t in selecting a particular route, we employ a reward r_{t+1}. Based on the reward policy, the reinforcement Q-learning agent receives a positive reward of one hundred or a negative reward of -10 for each action a_t performed at each state s_t.

In this work, we consider a delay for a reward in sending a packet from the source node to the destination node. To determine how optimal an action a_t is in selecting a specific route, a reward r_{t+1} is utilized. Based on the reward policy, the reinforcement Q-learning agent receives a positive reward of one hundred or a negative reward of -10 for each action a_t performed at each state s_t.

```
27      private final int reward = 100;
28      private final int penalty = -10;
```

4.8 Q-Tables

We examine the issue of forwarding packets from a source to a destination node utilizing some function metrics, including throughput, packet transmission rate, latency, and processing time. Firstly, we provide link state information to the reinforcement learning agent from the controller's control plane. Input parameters include discount rate γ, learning rate α, source and destination parameters, and network size under controller domain.

At first, we supply the reinforcement learning agent with link state information from the control plane. This input information includes discount rate γ, learning rate α, source and destination parameters, and network size overseen by the controller. In order to identify the most suitable approach, we begin by initializing Q-table values with some arbitrary values $Q(s, a)$ to zero for all state-action (s, a) pairs. At any state s, we first query ε-greedy policy ε. We execute the action a_t and observe the returned reward r_{t+1}, followed by a transition to subsequent state s'. The old Q-value for the previous state is updated in the Q-table utilizing Q-learning updated rules. At the end of the training process, after a number of episodes are completed, we get the Q table as follows:

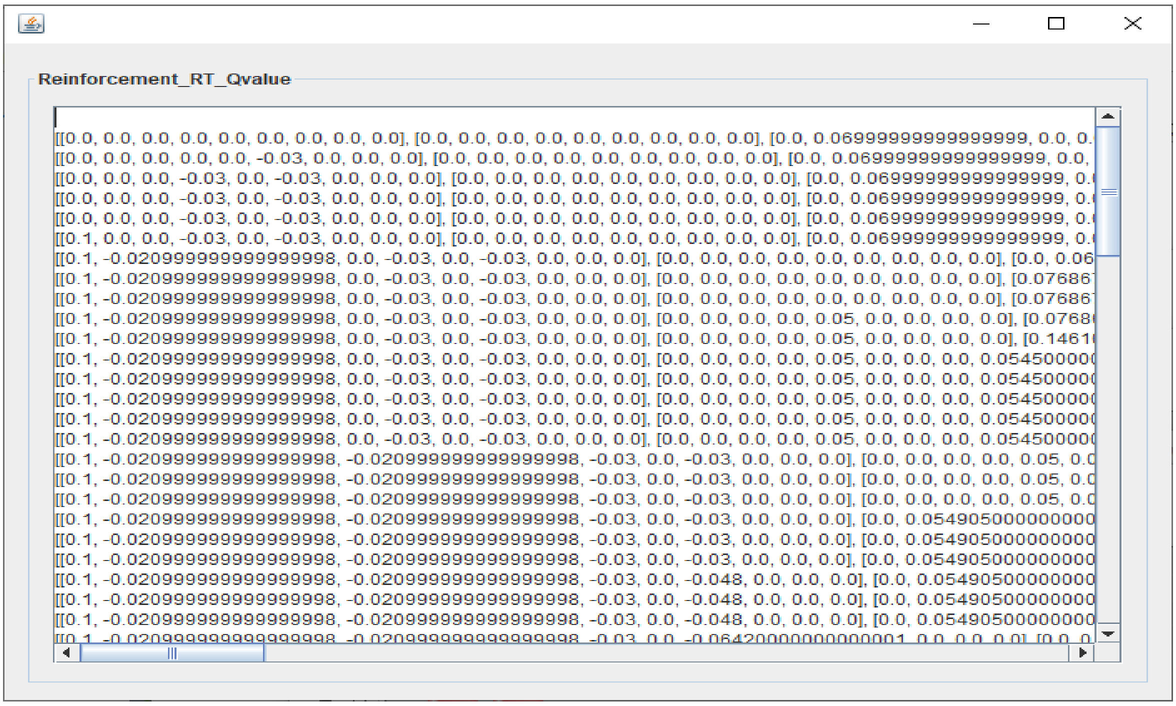

Figure 4. 5 Q-Values Table

In the Q-Table, the columns are used as actions and the rows are used as states. To construct the Q-table, a 9x4 array of state and action space was created. Subsequently, an epsilon greedy policy strategy, discussed in the preceding section, was employed to balance exploration and exploitation in the Q-Table. In this instance, the exploitation probability is 1-

ε, while the exploration probability is ε. In this instance, we create a random number in the range 0 to1. If the generated random number exceeds the epsilon value, the agent takes advantage of the information it already has to direct traffic to the target node. In this situation, the agent will execute the action that has the highest value in relation to the current state. In contrast, we would execute a random action via exploration if the number generated is lower than the epsilon value. Due to the off-policy nature of Q-learning, the action policy and update function are distinct. Greedy policy is understood to be the updating policy, while Epsilon greedy policy is considered to be the acting policy.

```
29  class Qtable {
30          ArrayList<ArrayList<Double>> qValues = new ArrayList<ArrayList<Double>>();
31          ArrayList<ArrayList<Double>> Rewards = new ArrayList<ArrayList<Double>>();
32          int actions;
33          int states;
34
35          private final double learningRate = 0.1;
            private final double rewardDecay = 0.9;
            private final double greedy = 0.9;
38
39          public Qtable(int actions, int states) {
40            this.actions = actions;
41            this.states = states;
42
              initializeQtable();
              initializeRewardsMatrix();
45
46          }
47
```

Figure 4. 6 Q-Table Algorithm

The last policy employed is the greedy policy, which involves training the agent and selection of the highest state value and action value from a Q-table. To ensure that the agent can explore enough state space and has sufficient network knowledge, 1,000 training sessions and a hundred (1,00) evaluation episodes are employed. Initial learning rate begins at a rate of 0.1, and the gamma or discount rates start at 0.9, and the epsilon probability decay rates begin at 0.9.

4.9 Model Training

When reinforcing the SDN Controller, we establish a sequence of training episodes. The goal is to reduce the dependency on exploration rather than exploitation by decreasing the epsilon value per episode. Initially, we choose an action using the epsilon greed policy.

The Q-function is then updated using the Bellman equation(3.9). At the end of the episode, the current state is changed, and the new state s' is observed. The next step is to repeat the action a_t, which is to observe the predicted reward r_{t+1} and a transition to a new state s_{t+1}. The action a_t is repeated and the expected reward r_{t+1} is observed. Upon successful completion of the training, the Q-Table is updated with new calculated values. Consequently, the agent will utilize these new values to direct packets through the network from the source to the destination node to receive the most advantageous reward from the Software Defined Networking controller.

Algorithm* 2**: ***Epsilon − Greedy Q − Learning Algorithm
1: **Data**: α: *Learning rate,* γ: *Discount rate,* ϵ: *A small number*
2: **Results**: $A - Table\ containing\ Q(s,a)\ pairs\ defining\ estmated\ optimal\ policy\ \pi^*$
3: $Initialize\ Q(s,a)\ arbitrarily, except\ Q(terminal);$
4: $Q(terminal) \leftarrow 0;$
5: $for\ each\ episode\ do$
6: $Initialize\ state\ s$
7: $for\ each\ step\ in\ the\ epistode\ do$
8: do
9: $a \leftarrow select - action(Q, s, \epsilon);$
10: $Take\ action\ a, then\ observe\ reward\ R\ and\ next\ state\ s'$

11: $Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma \max_a Q(s',a) - Q(s,a)\right];$

12: $s \leftarrow s';$

13: $while\ s\ is\ not\ terminal;$

14: $end;$

15: $end;$

Figure 4. 7 Model Training based on Epsilon Greedy Q-learning Algorithm

4.10 Q-Learning Route Selection Algorithm

This section zooms in at how solve a routing problem in our network environment by utilizing sHRRL, a dynamic routing protocol to route data packets from the source to the destination node utilizing our suggested Q-learning routing algorithm.

In Fig 4.8, we show the proposed Q-learning routing algorithm characterized by a Q-table of nine and four states and actions respectively as well as a minimum of one hundred episodes The proposed sHRRL algorithm is modelled to select alternative routes based on randomly choosing least congested best available routes through the Q-learning reinforcement approach. The SDN controller interacts with a reinforcement Q-learning agents to find the next best action, given the current state. The action is firstly chosen at random to maximize the reward and find an optimal-forwarding route.

```
class Qlearning_routing_algorithm {
  public static void main(String[] args) {

    Qtable one = new Qtable(9, 4);
    Random random  = new Random();

    for(int i=0; i < 100; i++){
      int state  = random.nextInt(4);
      int action = random.nextInt(9);
      int nextState = random.nextInt(4);
      one.updateQTable(state, nextState, action, 0.9);
      one.print();
    }
  }
}
```

Figure 4. 8 Q-learning Routing algorithm

In our sHRRL Q-learning route selection algorithm, each node determines which adjacent switch to forward a packet to, for a minimum average packet delivery time, based on current flow table information exploitation. To reflect the current network state, each switch keeps updating the routing or the flow table to make efficient routing decisions. Each switch's routing information is stored in a Q-table for expected packet routing time between adjacent switches. The forwarding switch translates the estimated forwarding time of the source node's packet into an estimated remaining packet travelling time, which is then forwarded back to the receiving switch. This is known as the reinforcement action, and it is utilized by the receiving switch to update the Q-value in the Q-table.

The Q-table's updated Q-values are aligned with the estimated forwarding time after an initial learning cycle. This can be interpreted as a more accurate representation of the current state of the network environment, resulting in a more accurate routing decisions and improved network performance.

4.11 Analysis and Discussion of Results

To end with, the hybrid switching technique proposed is assessed for its overall performance between Open Flow switch and Hybrid switch. This analysis is conducted on the basis of network performance metrics, such as average throughput, packet exchange rate, CPU load, and latency. The simulation results are analyzed statistically, and the performance results are compared by plotting them on a graph.

4.11.1 Hybrid Switch vs. Open Flow Switch Packet Exchanged

Fig 4.9 illustrates the correlation between the number of packets exchanged and the time. The OpenFlow switch was compared with the hybrid switch and the correlation between the two was verified. The path flow rate was 78 bytes, and the evaluation was extended to 12,000 bytes, or 12Kbytes for a duration of approximately 7 seconds. The analysis revealed that when the initial network packets were exchanged, the open flow switch's processed packet count began

to increase between the first-second and two-second time step *t*.

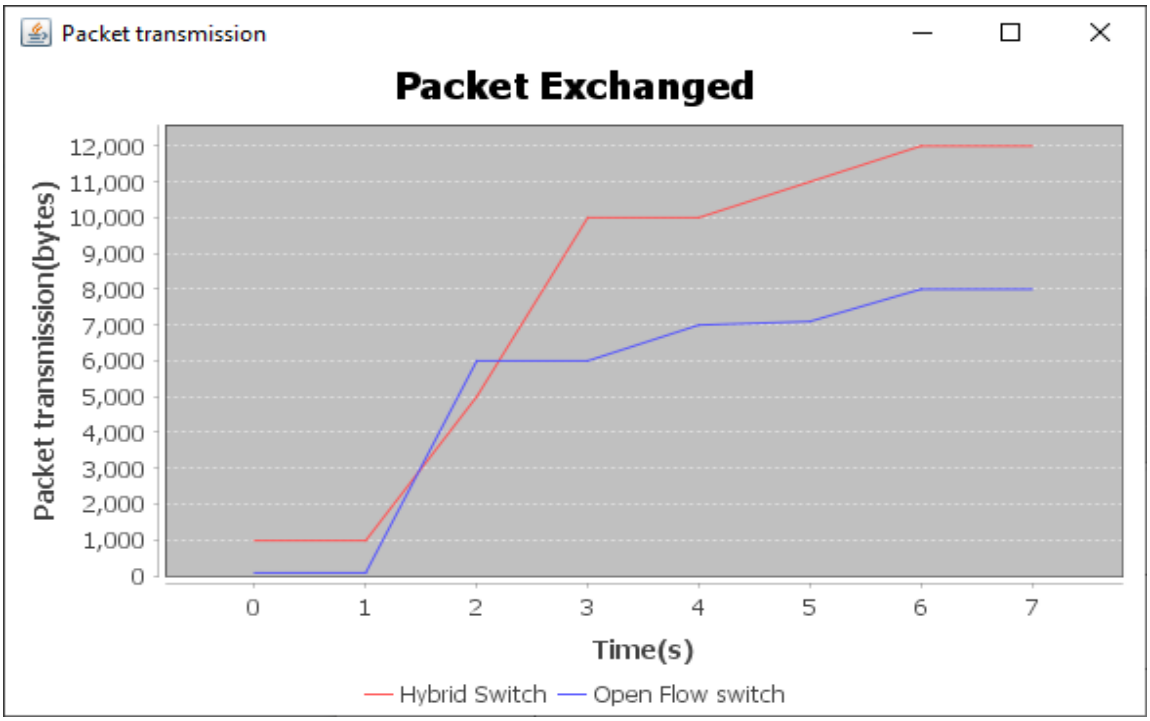

Figure 4. 9 Packet Exchanged for Hybrid Switch vs. Open Flow Switch

Table 4. 2 Hybrid Switch vs. Open Flow Switch Packet Exchange

Packet transmission(bytes)		
Time in seconds	**Packet exchanged. (Hybrid Switch)**	**Packet exchanged. (Open flow Switch)**
0	980	78
1	980	79
2	5000	6000
3	10000	6000
4	10000	7000
5	11000	7100
6	12000	8000
7	12000	8000

This is because the hybrid switch has to begin its training process with the implementation of the sHRRL training framework. The Q-function must be updated with calculated Q-values to ensure that the Hybrid switch is using the most optimal route to route packets from the source to the destination node and receive the most advantageous reward from the Software Defined Networking controller.

It was observed that the OpenFlow switch does not utilize all the available paths, thus it is outperformed by the hybrid switch. While the OpenFlow switch saturates at a rate of 8000 bytes per second, the hybrid switch begins to saturate at a rate of 12000 bytes per second. This is due to the application of the sHRRL framework and the implementation of the Q-learning algorithms in the hybrid switch for the calculation of an optimal-forwarding route, as well as the use of the shortest path algorithm achieved through converged Q-values in the Q-Table.

4.11.2 Throughput

We started off with a 500Kbps path flow rate, and scaled up to 5Mbps throughput. As demonstrated in Figure 4.10, the Hybrid Switch routing system achieves comparable scalability results to the Open Flow switching. This is due to the fact that both schemes utilize the shortest path routing algorithms. In contrast, the Open Flow Switch saturates first as it does not utilize all available paths and is therefore outplayed by the Hybrid Switching approach as far as received flow rate is concerned. The benefits of the hybrid switching approach are that the local control plane takes control of flow path configuration without controller involvement. This significantly reduces the amount of time it takes to establish a flow path.

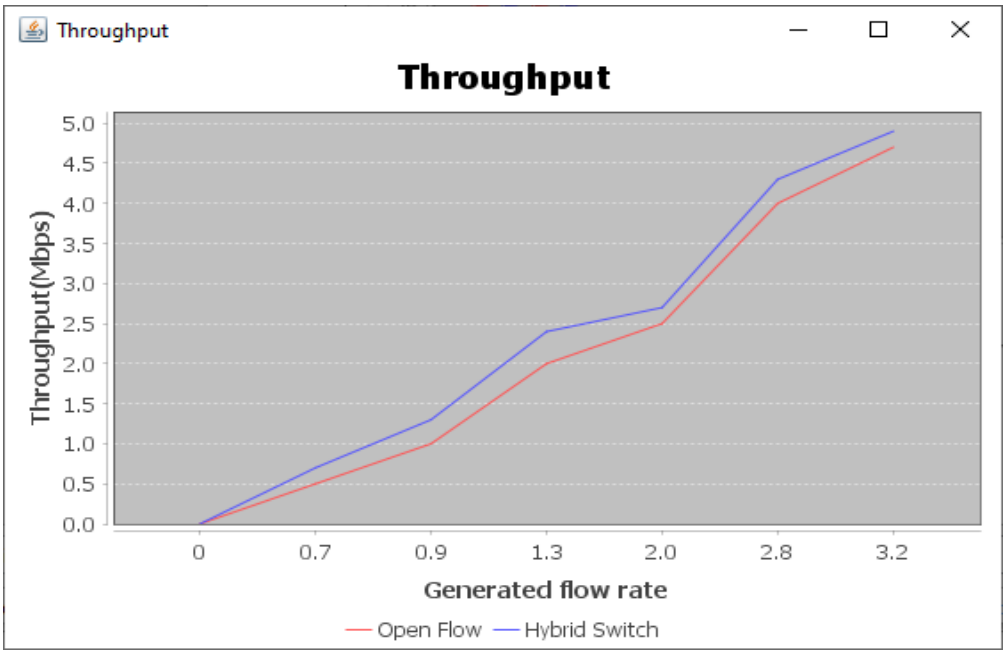

Figure 4. 10 Average Throughput Comparison between Hybrid Switch and Open Flow Switch

Table 4. 3 Average Throughput Parameters for Hybrid Switch vs. Open Flow Switch

Throughput (Mbps)		
Generated Flow rate	**Throughput (Hybrid Switch)**	**Throughput (Open flow Switch)**
0	0	0
0.7	0.7	0.5
0.9	1.3	1.0
1.3	2.4	2.0
2.0	2.7	2.5
2.8	4.3	4.0
3.2	4.9	4.7

4.11.3 Average Path Setup Time

The first step in evaluating the reduction of traffic flow at the controller is to delegate the flow configuration between the Open Flow Switch and the Hybrid Switch. These flows were initiated in a sequential manner and were maintained at a rate of 2Mbps until the end of the episode.

In Figure 4.11 below, we plot the average time required for the traffic flow configuration after the simulation has been executed for packet delivery between the source and destination node.

Hybrid switch has been found to have a relatively high latency. Although the latency is relatively low, the delay is attributed to the fact that the hybrid switch has interacts with SDN OpenFlow switch and traditional switch before initiating traffic routing within the network environment.

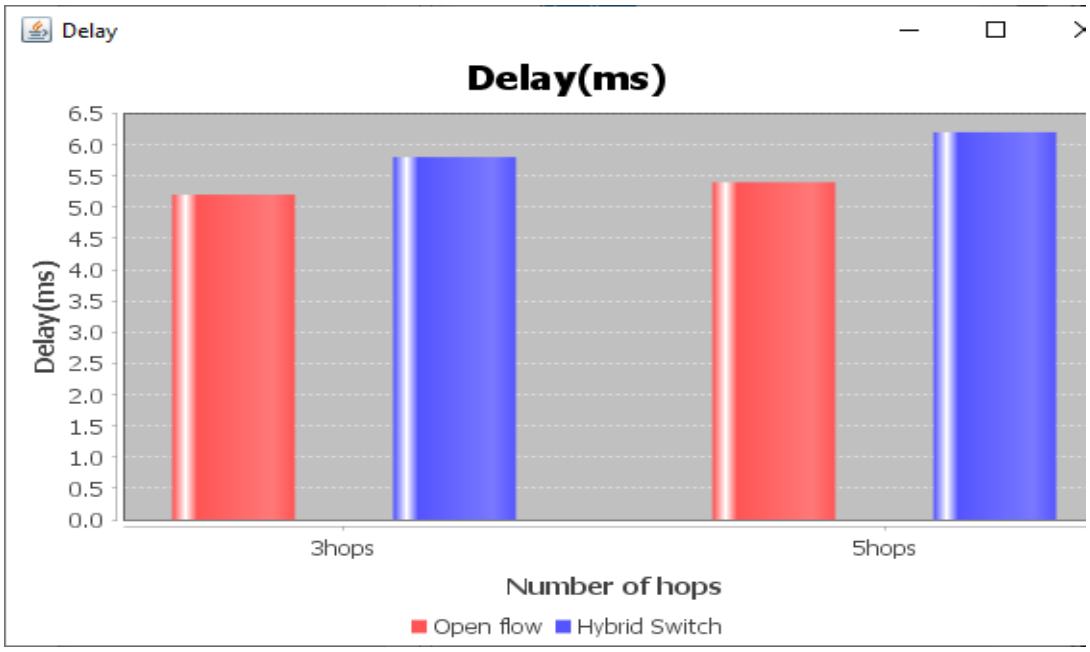

Figure 4. 11 Number of hops vs. Delay

Table 4. 4 Delay vs. Number of Hops

Delay		
Hop count	**Delay (Hybrid Switch)**	**Delay (Open flow Switch)**
3	5.8	5.2
5	6.2	5.4

4.11.4 OpenFlow Switch vs. Hybrid Switch CPU load

The CPU enhancement observed in Fig 4.12 is due to the autonomous establishment of switch flows, which does not require the switch controller to perform any calculation for the establishment of new path. The OpenFlow controller's CPU load is further compared to the

CPU load of the Hybrid switch. Once again, the Hybrid switch surpasses the Open Flow switch in terms of CPU utilization.

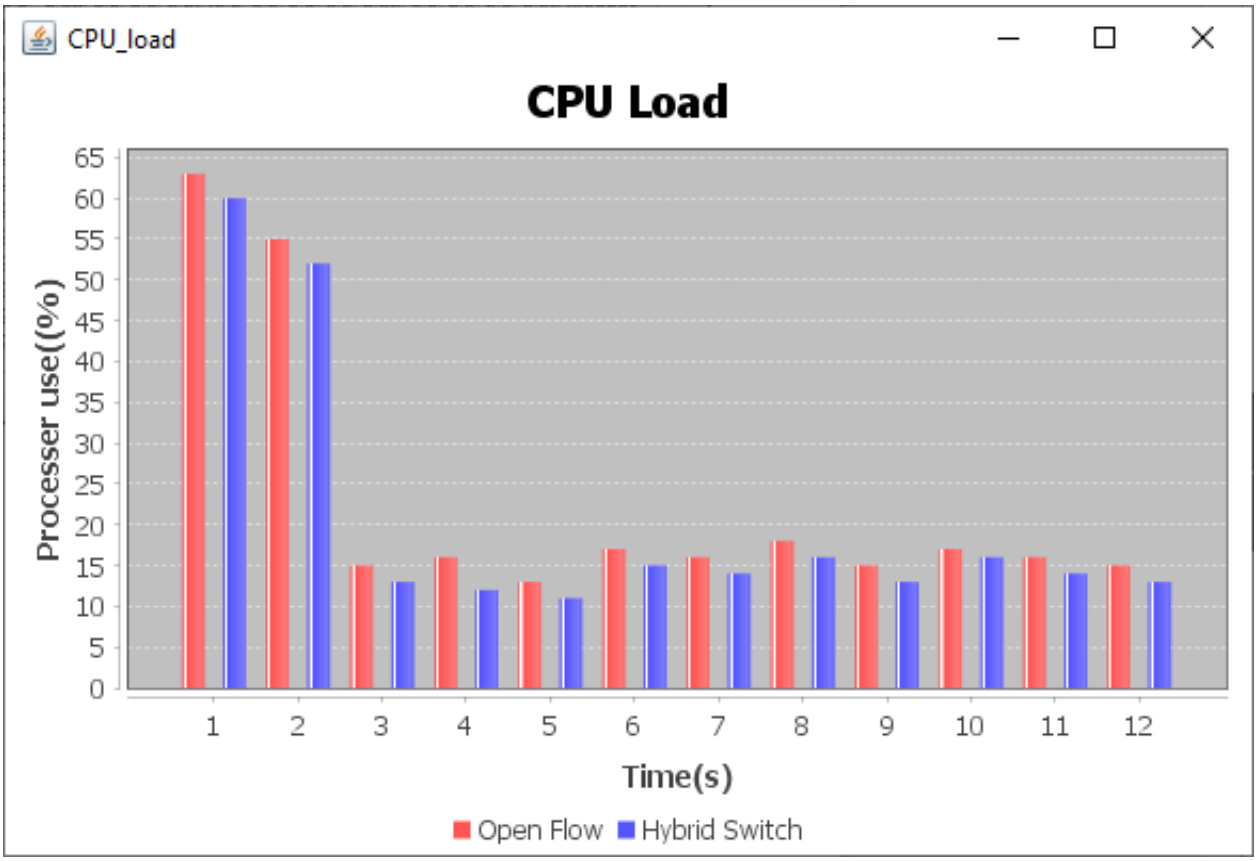

Figure 4. 12 Open Flow Switch vs. Hybrid Switch CPU load

Table 4. 5 OpenFlow Switch vs. Hybrid Switch CPU load

CPU Load (%)		
Time(s)	**Processor use (%) (Hybrid Switch)**	**Processor use (%) (Open flow Switch)**
1	60	63
2	52	55
3	13	15
4	12	16
5	11	13
6	15	17

7	14	16
8	16	18
9	13	15
10	16	17
11	14	16
12	13	15

4.12 Chapter Summary

This chapter compared the performance of a proposed hybrid switching technique to that of an SDN Controller OpenFlow switch, taking into account a variety of network performance metrics, such as average throughput, packet exchange rate, CPU load, and delay. The simulation results were then analysed statistically, and the results were compared by plotting the performance results on a graph. The Q-learning algorithm was employed to design a Scalable framework for hybrid routing with reinforcement learning, and the results showed that the proposed framework was more efficient in routing packets than an OpenFlow switch approach.

When the packet exchanged rate was compared between the Hybrid switch and the Open Flow switch, the average ratio of packets transmitted by the Hybrid switch was found to be higher than the Open Flow switch average packet exchange rate. This was attributed to the increased route exploration and the Q-learning that Hybrid switch was able to exploit from the Q-table to set up flow table. Additionally, the throughput of the Hybrid switch was observed to be higher than that of the Open Flow switch. The benefit of the Hybrid switch lies in its ability to proactively install the Q-values into the Q-table for optimal path selection, when compared to the forwarding control plane-related traffic requests to a controller for each data packet that must be forwarded between the source node and the destination node. By reducing

the number of control-plane traffic forwarded to a controller, the amount of packets forwarded to the control plane is reduced, thus increasing scalability.

A further comparison of CPU load was conducted comparing OpenFlow controller with Hybrid switch. Hybrid switch was found to be more efficient than OpenFlow controller in terms of CPU usage. The observed CPU increase is attributed to the autonomous establishment of switch flows setup by the switch controller, which does not require any calculation for the establishment of new paths. The above findings and analysis lead to the conclusion of this research work, where the conclusion and future work will be discussed in detail.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Introduction

In this research effort, a conclusion based on the experimental results for the problem statement discussed in Chapter 1, is presented. We also discuss recommended future work that can be persuaded to further this study. This includes recommended future work on controller-switch convergence after a link failure, optimal routing algorithm model, security aspects in the controller and switch, as well as traffic analysis for QoS satisfaction to improve network performance and better end-user quality of experience in the Data Centre environment.

5.2 Conclusion

Machine learning in Software-Defined Networking is destined to be a key-factor in enabling efficient dynamic routing of applications. However, there are a number of obstacles that must be overcome before such optimal routing can be achieved in the context of the next-generation of networks. In this research, we sought to facilitate a scalability through the hybrid Switching software-defined networking utilizing reinforcement learning algorithms to route packets in a data centre network environment. The goal was to establish interoperability between the SDN OpenFlow switch and the traditional switch to form the Hybrid Switching, utilizing a RL algorithm to set up flow tables for forwarding rules.

The overall performance of the proposed hybrid switching technique was evaluated against the SDN Open Flow switch based on network performance metrics including average throughput scalability, packet exchange transmission rate, CPU load, and delay between the

OpenFlow switch and the hybrid switch. Our simulation results demonstrated various conclusions. We demonstrated that when a reinforcement learning algorithm based on the Q-learning is used in the SDN Hybrid switching, it performs superior when compared to an SDN controller with an open flow switch.

To attain a higher level of performance than the OpenFlow switch, the Q-values from a Q-table were employed in the proposed sHRRL algorithm to take the most appropriate action based on the anticipated optimal reward for every transition or changes in the network environment. To route data packets from the originating node to the receiving node, certain function metrics were taken into account, including packet exchange rate, throughput, latency, and processing time. At first, a link state information was provided to the reinforcement learning agent through the control plane. The supplied input included a discount rate, a learning rate, as well as a source and destination parameters and the network size within the controller domain.

To explorer the optimal path, we started by initializing the Q-table with some arbitrary values Q(s, a) to zero for all state-action (s, a) pairs. At any given state, we first probed the ε greedy policy, to determine whether we need to exploit or explore the current network state. Based on the greedy policy, we execute an action and observe the expected reward and network transition to the next state. At the end of the model training episode, the Q-learning algorithms were updated with the latest Q-values in the Q-table and the Bellman Equation to obtain the expected future reward.

When comparing the packet exchange rate of the hybrid switch to the Open Flow switch, it was observed that the packets exchanged by the hybrid switch were higher than those exchanged by the Open Flow switch. This was attributed to a more route exploration through the Q-learning algorithms utilized in the hybrid switching. Additionally, the Hybrid switching

throughput was observed to be higher than that of the Open Flow switch. The advantage of the Hybrid switching is a reinforcement learning algorithm installed which support hybrid switching in optimal path selection. Unlike the OpenFlow switch were OpenFlow southbound API interface has to be flooded with the control plane traffic for flow rules requests, every time a data packet has to be routed by the OpenFlow switch data plane. The introduction of reinforcement learning algorithms leads to the reduction in the amount of control plane packets sent to the controller, which frees up the controller's memory and enhances scalability in a software-defined networking environment.

The outcome of the simulation demonstrated that the reinforcement learning framework, sHRRL, based on the Q-learning algorithms, enhances the performance of the hybrid switching when compared to the Open Flow switch. Consequently, we are confident that the suggested hybrid switching approach implemented through a machine learning algorithm, can address scalability issues in the design of SDN controller networks, particularly in a data centre where high switching speeds are of utmost importance.

5.3 Future Work

The simulation results presented demonstrate a significant increase in scalability, however, they can be further developed and tested in a variety of scenarios. For instance, future work could be focused on increasing scalability by implementing additional routing algorithms, such as deep reinforcement learning algorithms. Additionally, we can reduce model training time by implementing additional code with prior understanding of the network environment and reducing epsilon value per episode. This can lead to less exploration and greater exploitation, as suggested by the trade-off outlined in Section 3.5. of the preceding Chapter.

CloudSim, an open-source framework, was employed as a platform for modelling and simulation of our experiment. Through the routing algorithm simulation performed through

CloudSim, it was observed that all nodes, hosts, and access links had comparable characteristics, including bandwidth capacity. It is suggested that more accurate test results may be obtained if a more realistic or physical network infrastructure were to be used, with different application and traffic models being prioritized in different service classes.

The use of Software-Defined Networking controllers can be associated with a variety of malicious threats and attacks. These threats may include botnets, denial of service, distributed denial of service, man-in-the-middle attacks, and a variety of other attacks against SDN controllers and OpenFlow switches, among others. This analysis can be expanded upon by examining the potential mitigation options for the security aspects of SDN controllers and switches.

To illustrate the robustness of the routing algorithm proposed, we can further investigate the recovery time as well as routing performance after a link failure utilizing the Hybrid Switching. The network performance after a disturbance caused by a specific failure, as well as the time it takes to correct itself and return to normal operation, can be further optimized to a minimum possible latency. We believe that this can be accomplished through the employment of machine learning and collaboration of the SDN control layer and the hybrid switching.

We also consider using Deep Neural Network-Based (DNN) RL techniques such as Deep Q-Network, Deep Deterministic Policy Gradient (DDPG), and Deep Proximal Policy Optimization (DPPO) techniques in our future research work involving reinforcement learning. We believe these techniques can offer better performance, enhanced stability and most importantly higher scalability compared to the table-based approach used in this research effort.

Given the limited timelines available, we leave all these intriguing challenges for future research efforts.

6 References

[1] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "BalanceFlow: Controller load balancing for OpenFlow networks, "in Proc. IEEE 2nd Int. Conf. Cloud Comput. Intell. Syst. (CCIS), Oct/Nov. 2012, vol. 2, pp. 780-785.

[2] Open Network Foundation,"Software Defined Networking: The New Norm for Networks" ONF White Paper, 2012.

[3] Curtis AR, Mogul JC, Tourrilhes J, Yalagandula P, Sharma P. DevoFlow: Scaling Flow Management for High-Performance Networks. SIGCOMM Cmpit. Comm. Rev. Aug 2011, 41(4):254-265.

[4] Tavakoli et al. Hotnets, 2009.

[5] Aster*x, ACLD demo 2010; Wang et al., HotICE 2011.

[6] Kim et al., INM/WREN, 2010.

[7] NEC, "Nec ProgrammableFlow UNIVERGE PF5820," 2013. [Online]. Available: http://www.nec.com/en/global/prod/pflow/imagesdocuments/ProgrammableFlow Switch PF5820.pdf.

[8] NoviFlow, "NoviSwitch 1248 High Performance OpenFlow Switch," 2013. [Online]. Available: http://205.236.122.20/gestion/NoviSwitch1248Datasheet.pdf.

[9] Stephens B, Cox A, Felter W, Dixon C, Carter J. PAST: Scalable ethernet for data centers. Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, CoNEXT '12, ACM, 2012; 49–60, doi:10.1145/ 2413176.2413183.

[10] Hassas Yeganeh S, Ganjali Y. Kandoo: A Framework for Efficient and Scalable Offloading of Control Aplications. Proceedings of the First Workshop on Hot Topics in Software Defined Netowks, HotSDN '12, ACM: New York, NY, USA, 2012; 19-24.

[11] Wang J, Yu M, Freedman MJ. Scalable flow-based networking with difane. SIGCOMM Comput. Commun. Rev. Aug 2010; 41(4).

[12] Zheng K, Wang L, Yang B, Sun Y, Uhlig S. Lazyctrl: A scalable hybrid network control plane design for cloud data centers. IEEE Transactions on Parallel and Distributed Systems 2016; PP(99):1–1.

[13] Mekky H, Hao F, Mukherjee S, Zhang ZL, Lakshman T. Application-aware data plane processing in sdn. Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, HotSDN '14, ACM: New York, NY, USA, 2014; 13–18.

[14] SDN Versus Traditional Networking Explained. https://www.cisco.com/c/dam/en_us/solutions/industries/docs/gov/software_defined_networking.pdf

[15] Software-Defined Networking and Network Programmability: Use Cases for Defense and Intelligence Communities. https://www.ibm.com/services/network/sdn-versus-traditional-networking.

[16] FOERSTER, K.-T., SCHMID, S., AND VISSICCHIO, S. Survey of consistent software-defined network updates. *IEEE Communications Surveys & Tutorials* (2018).

[17] NUNES, B. A. A., MENDONCA, M., NGUYEN, X.-N., OBRACZKA, K., AND TURLETTI, T. A survey of software-defined networking: Past, present, and future of programmable networks. IEEE Communications Surveys & Tutorials 16, 3 (2014), 1617–1634.

[18] S. Yeganeh, A. Tootoonchian, Y. Ganjali, On scalability of software-defined networking, Communications Magazine, IEEE 51 (2) (2013) 136–141. doi:10.1109/MCOM.2013. 6461198.

[19] Diego Kreutz, Fernando M. V. Ramos, Paulo Verissimo, Christian Esteve Rothenberg, "Software-Defined Networking: A Comprehensive Survey," VERSION 2.01, arXiv: 1406.0440v3 [cs.NI] 8 Oct 2014.

[20] L. J. Cowen, Compact routing with minimum stretch, in: Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '99, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999, pp.255–260. URL http://dl.acm.org/citation.cfm?id=314500.314566.

[21] Hybrid OpenFlow, the Brocade Way a ipSpace.net by @ioshints 2012. URL http://blog.ipspace.net/2012/06/hybridopenflow-brocade-way.html.

[22] The Practical Path to SDN: Brocade OpenFlow Hybrid Port Mode 2013. URL http://community.brocade.com/t5/SDN-NFV/The-Practical-Path-to-SDN-Brocade-OpenFlow-Hybrid-Port-Mode/ba p/417

[23] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, R. Sherwood, On controller performance in software-defined networks, in: Proceedings of the 2Nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, Hot-ICE'12, 2012, pp. 10–10.

[24] S. A. Shah, J. Faiz, M. Farooq, A. Shafi, S. A. Mehdi, An architectural evaluation of sdn controllers, in: 2013 IEEE International Conference on Communications (ICC), 2013, pp. 3504–3508. doi:10.1109/ICC.2013.6655093.

[25] F. Benamrane, R. Benaini, et al., Performances of openflowbased software-defined networks: an overview, Journal of Networks 10 (6) (2015) 329–337.

[26] P. Isaia, L. Guan, Performance benchmarking of sdn experimental platforms, in: 2016 IEEE NetSoft Conference and Workshops (NetSoft), 2016, pp. 116–120. doi:10.1109/ NETSOFT.2016.7502456.

[27] K. He, J. Khalid, S. Das, A. Gember-Jacobson, C. Prakash, A. Akella, L. E. Li, M. Thottan, Latency in software defined networks: Measurements and mitigation techniques, in: Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '15, ACM, New York, NY, USA, 2015, pp. 435–436. doi:10.1145/2745844.2745880. URL http://doi.acm.org/10.1145/2745844.2745880.

[28] Cisco Networking Academy, "Introduction to Basic Switching Concepts and Configuration" Mar 31, 2014.

[29] https://www.researchgate.net/publication/320347086_Implementation_of_SDN_using_Open Day Light_Controller/figures?lo=1 [Accessed 30 April 2023].

[30] RIPv2 https://tools.ietf.org/html/rfc1058. Accessed 14 June 2023.

[31] OSPFv2 https://tools.ietf.org/html/rfc2178. Accessed 14 June 2023.

[32] BGP-4. https://datatracker.ietf.org/doc/html/rfc1654. Accessed 14 June 2023.

[33] Othmane Blial, Mouad Ben Mamoun, Redouane Benaini, An Overview on SDN Architecture With Multiple Controllers [J]. Journal of Computer Networks and Communications, 2016.

[34] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[35] Xenofon Foukas, Mahesh Marina et al, "Software Defined Networking Concepts", August 07 2014.

[36] Sarwar Raza (HP), David Lenrow, "Open Networking Foundation North Bound Interface Working Group (NBI-WG) Charter", Version 1.1, 2013.

[37] Open Networking Foundation, "Openflow table type patterns." https://www.opennetworking.org/images/stories/downloads/sdnresources/onf specifications/openflow/OpenFlowAugust 2014. [Accessed 12 March 2023].

[38] Ching-Hao, Chang and Dr. Ying-Dar Lin, "OpenFlow version roadmap" September 11, 2015.

[39] Dabbagh, M.; B. Hamdaoui; M. Guizani; A. Rayes; "Software-Defined Networking Security: Pros and Cons," *IEEE Communications Magazine*, vol. 53, iss. 6, May 2015.

[40] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, R. Chaiken, The nature of data center traffic: Measurements & analysis, in: Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference, IMC '09, ACM, New York, NY, USA, 2009, pp. 202–208. doi:10.1145/1644893.1644918.
URL http://doi.acm.org/10.1145/1644893.1644918

[41] D. Erickson, The beacon OpenFlow controller, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, Hot SDN '13, 2013, pp. 13–18.

[42] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, R. Smeliansky, Advanced study of sdn/openflow controllers, in: Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia, CEESECR '13, ACM, New York, NY, USA, 2013, pp. 1:1–1:6. doi:10.1145/2556610.2556621. URL http://doi.acm.org/10.1145/2556610.2556621.

[43] R. Narayanan, S. Kotha, G. Lin, A. Khan, S. Rizvi, W. Javed, H. Khan, S. A. Khayam, Macroflows and microflows: Enabling rapid network innovation through a split SDN data plane, in: 2012 European Workshop on Software Defined Networking, 2012, pp. 79–84. doi:10.1109/EWSDN. 2012.16.

[44] K. Kannan, S. Banerjee, Compact TCAM: Flow Entry Compaction in TCAM for Power Aware SDN, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 439–444. doi: 10.1007/978-3-642-35668-1_32. URL http://dx.doi.org/10.1007/978-3-642-35668-1_32.

[45] Q. Zuo, M. Chen, K. Ding, B. Xu, On generality of the data plane and scalability of the control plane in software defined networking, China Communications 11 (2) (2014), pp. 55–64.

[46] S. A. Shah, J. Faiz, M. Farooq, A. Shafi, S. A. Mehdi, An architectural evaluation of sdn controllers, in: 2013 IEEE International Conference on Communications (ICC), 2013, pp. 3504–3508. doi:10.1109/ICC.2013.6655093.

[47] X. H. Sun, Y. Chen, M. Wu, Scalability of heterogeneous computing, in: Proceedings of the 2005 International Conference on Parallel Processing, ICPP '05, IEEE Computer , Washington, DC, USA, 2005, pp. 557–564. doi:10.1109/ICPP.2005.69. URL http://dx.doi.org/10.1109/ICPP.2005.69.

[48] A. Y. Grama, A. Gupta, V. Kumar, Isoefficiency: Measuring the scalability of parallel algorithms and architectures, IEEE concurrency (3) (1993) 12–21.

[49] L. Pastor, J. Bosquwe Orero, An efficiency and scalability model for heterogeneous clusters, in: Cluster Computing, 2001. Proceedings. 2001 IEEE International Conference on, 2001, pp. 427–434. doi:10.1109/CLUSTR.2001.960009 doi:10.1109/ICPP.2005.69. URL http://dx.doi.org/10.1109/ICPP.2005.69.

[50]] K. Hwang, Z. Xu, Scalable Parallel Computing: Technology, Architecture, Programming, McGraw-Hill, Inc., New York, NY, USA, 1998.

[51] X. H. Sun, D. T. Rover, Scalability of parallel algorithm machine combinations, IEEE Trans. Parallel Distrib. Syst. 5 (6) (1994) 599–613. doi:10.1109/71.285606. URL http://dx.doi.org/10.1109/71.285606.

[52] V. Kumar, A. Grama, A. Gupta, G. Karypis, Introduction to Parallel Computing: Design and Analysis of Algorithms, Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1994.

[53] Murat Karakusa, Arjan Durresia, “A Survey: Control Plane Scalability Issues and Approaches in Software Defined Networking”. *Department of Computer and Information Science Indiana University Purdue University Indianapolis Indianapolis, IN 46202, USA, November 10, 2016.*

[54] Neil J. Gunther, “Guerrilla Capacity Planning” A Tactical Approach to Planning for Highly Scalable Applications and Services, pages 41-69, 2007.

[55] Wang J, Yu M, Freedman MJ. Scalable flow-based networking with difane. SIGCOMM Comput. Commun. Rev. Aug 2010; 41(4).

[56] Hassas Yeganeh S, Ganjali Y. Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications. Proceedings of the First Workshop on Hot Topics in Software Defined Netowks, HotSDN ’12, ACM: New York, NY, USA, 2012; 19-24.

[57] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, “DevoFlow: scaling flow management for highperformance networks,” Comput. Commun. Rev., vol. 41, no. 4, pp.254–265, Aug. 2011.

[58] S. Hassas Yeganeh and Y. Ganjali, “Kandoo: A framework for efficient and scalable offloading of control applications,” in Proceedings of the First Workshop on Hot Topics in Software Defined Networks, ser. HotSDN ’12. New York, NY, USA: ACM, 2012, pp. 19–24.

[59] J. C. Mogul and P. Congdon, “Hey, you darned counters!: Get off myasic!” in Proceedings of the First Workshop on Hot Topics in Software Defined Networks, ser. HotSDN ’12. New York, NY, USA: ACM, 2012, pp. 25–30.

[60] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, “Maple: simplifying SDN programming using algorithmic policies,” in Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM, ser. SIGCOMM ’13. New York, NY, USA: ACM, 2013, pp. 87-98.

[61] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, “On controller performance in software-defined networks,” in Proceedings of the 2nd USENIX conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, ser. Hot-ICE’12. Berkeley, CA, USA: USENIX Association, 2012, pp. 10–10.

[62] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, “Onix: a distributed control platform for large-scale production networks,” in Proceedings of the 9th USENIX conference on Operating systems design and implementation, ser. OSDI’10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6.

[63] Z. Cai, A. L. Cox, and T. S. E. Ng, “Maestro: A System for Scalable OpenFlow Control,” Rice

University, Tech. Rep., 2011.

[64] Hassas Yeganeh S, Ganjali Y. Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications. Proceedings of the First Workshop on Hot Topics in Software Defined Netowks, HotSDN '12, ACM: New York, NY, USA, 2012; 19-24.

[65] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: scaling flow management for highperformance networks," Comput. Commun. Rev., vol. 41, no. 4, pp.254–265, Aug. 2011.

[66] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in Proceedings of the First Workshop on Hot Topics in Software Defined Networks, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 19–24.

[67] M. Blose and L. A. Akinyemi, "Development Of Scalable Hybrid Switching Based Software Defined Networking," *2020 23rd International Symposium on Wireless Personal Multimedia Communications (WPMC)*, Okayama, Japan, 2020, pp. 1-5, doi: 10.1109/WPMC50192.2020.9309465.

[68] M. Blose and L. A. Akinyemi, "Scalable Hybrid Switching-Inspired Software Defined Networking Challenges: From the Perspective of Machine Learning Approach," *2024 Conference on Information Communications Technology and Society (ICTAS)*, Durban, South Africa, 2024, pp. 186-191, doi: 10.1109/ICTAS59620.2024.10507139.

[69] M. Blose, L. A. Akinyemi, S. Ojo, M. Faheem, A. L. Imoize and A. A. Khan, "Scalable Hybrid Switching-Driven Software Defined Networking Issue: From the Perspective of Reinforcement Learning Standpoint," in *IEEE Access*, vol. 12, pp. 63334-63350, 2024, doi: 10.1109/ACCESS.2024.3387273.

[70] Max Blose, Lateef Akinyemi, Fred Nicolls, Neco Ventura," Development of Scalable Hybrid Switching-Based Software Defined Networking Using Reinforcement Learning", The Southern Africa Telecommunication Networks and Applications Conference (SATNAC), vol.25, issue 1 pp. 163-168, 2023.

[71] S. O. Oladejo, S. O. Ekwe, L. A. Akinyemi and S. A. Mirjalili, "The Deep Sleep Optimizer: A Human-Based Metaheuristic Approach," in *IEEE Access*, vol. 11, pp. 83639-83665, 2023, doi: 10.1109/ACCESS.2023.3298105.

[72] L. A. Akinyemi, O. P. Oshinuga, O. A. Eshilokun, S. O. Oladejo and S. O. Ekwe, "Development of Phylum Chordata Sound Recognition System using Machine Learning and Deep Learning Techniques: A Case Study of Catfish," *2023 International Conference on Electrical, Computer*

and Energy Technologies (ICECET), Cape Town, South Africa, 2023, pp. 1-6, doi: 10.1109/ICECET58911.2023.10389248.

[73] L. A. Akinyemi, O. P. Oshinuga, S. O. Ekwe and S. O. Oladejo, "Enhancing Chronic Kidney Disease Prediction Through Data Preprocessing Optimization and Machine Learning Techniques," *2023 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, Cape Town, South Africa, 2023, pp. 1-6, doi: 10.1109/ICECET58911.2023.10389513.

[74] S. O. Oladejo, S. O. Ekwe and L. A. Akinyemi, "Multi-Tier Multi-Domain Network Slicing: A Resource Allocation Perspective," *2021 IEEE AFRICON*, Arusha, Tanzania, United Republic of, 2021, pp. 1-6, doi: 10.1109/AFRICON51333.2021.9570854.

[75] A. T. Ajibare, D. Ramotsoela, L. A. Akinyemi and S. O. Oladejo, "RF EMF Radiation Exposure Assessment of 5G Networks: Analysis, Computation and Mitigation Methods," *2021 IEEE AFRICON*, Arusha, Tanzania, United Republic of, 2021, pp. 1-6, doi: 10.1109/AFRICON51333.2021.9570865.

[76] S. O. Ekwe, L. A. Akinyemi, S. O. Oladejo and N. Ventura, "Social-Aware Joint Uplink and Downlink Resource Allocation Scheme Using Genetic Algorithm," *2021 IEEE AFRICON*, Arusha, Tanzania, United Republic of, 2021, pp. 1-6, doi: 10.1109/AFRICON51333.2021.9570887.

[77] A. T. Ajibare, S. O. Oladejo, S. O. Ekwe, L. A. Akinyemi and D. Ramotsoela, "Radiofrequency electromagnetic radiation exposure assessment, analysis, computation, and minimization technique in 5G networks: A perspective on QoS trade-offs," in *SAIEE Africa Research Journal*, vol. 114, no. 4, pp. 114-127, Dec. 2023, doi: 10.23919/SAIEE.2023.10319380.

[78] S. O. Ekwe, S. O. Oladejo, L. A. Akinyemi and N. Ventura, "A Socially-Inspired Energy-Efficient Resource Allocation Algorithm for Future Wireless Network," *2020 16th International Computer Engineering Conference (ICENCO)*, Cairo, Egypt, 2020, pp. 168-173, doi: 10.1109/ICENCO49778.2020.9357387.

[79] O. O. Shoewu, M. A. Adedoyin, L. A. Akinyemi and L. I. Oborkhale, "Fuzzy-logic Based Path Loss Models for Metropolitan Environment," *2018 IEEE International Workshop on Signal Processing Systems (SiPS)*, Cape Town, South Africa, 2018, pp. 71-76, doi: 10.1109/SiPS.2018.8598324.

[80] O. O. Shoewu, L. A. Akinyemi and L. Oborkhale, "Towards Developing Path loss Models for Dryland and Wetland Environments," *2019 IEEE AFRICON*, Accra, Ghana, 2019, pp. 1-7, doi: 10.1109/AFRICON46755.2019.9134041.

[81] Shoewu, O. O., Akinyemi, L. A., & Edozie, R. (2022). UAV Cellular Communication in 5G

New Radio Wireless Standards. In Unmanned Aerial Vehicle Cellular Communications (pp. 25-45). Cham: Springer International Publishing.

[82] Oladejo, S. O., Ekwe, S. O., & Akinyemi, L. A. (2021). MULTI-TIER MULTI-TENANT NETWORK SLICING: A MULTI-DOMAIN GAMES APPROACH. ITU Journal on Future and Evolving Technologies, 2(6).

[83] Wang, C., Zhang, L., Li, Z., and Jiang, C. (2018). Sdcor: Software defined cognitive routing for internet of vehicles. IEEE Internet of Things Journal, 5(5):3513–3520.

[84] Gopi, D., Cheng, S., and Huck, R. (2017). Comparative analysis of SDN and conventional networks using routing protocols. In CITS, pages 108–112. IEEE.

[85] Shirmarz, A. and Ghaffari, A. (2020). An adaptive greedy flow routing algorithm for performance improvement in software-defined network. International Journal of Numerical Modelling: Electronic Networks, Devices and Fields, 33(1).

[86] Martın, I., Troia, S., Hern´andez, J. A., Rodr´ıguez, A., Musumeci, F., Maier, G., Alvizu, R., and de Dios, O´ . G. (2019). Machine learning-based routing and wavelength assignment in software-defined optical networks. IEEE Transactions on Network and Service Management, 16(3):871–883.

[87] C. Watkins and P. Dayan, "Q-learning", Machine Learning, vol. 8, pp. 279-292, 1992

[88] Richard S, Sutton and Andrew G. Barto, “Reinforcement Learning: An Introduction”, Second Edition, 2018, pp. 47 – 68.

[89] https://deeplizard.com/learn/video/eMxOGwbdqKY [accessed 12 May 2023].

[90] https://deeplizard.com/learn/video/rP4oEpQbDm4 [accessed 16 May 2023].

[91] Richard S, Sutton and Andrew G. Barto, “Reinforcement Learning: An Introduction - Dynamic Programming Methods”, Second Edition, 2018, pp. 73 – 88.

[92] Richard S, Sutton and Andrew G. Barto, “Reinforcement Learning: An Introduction – T Temporal-Difference Learning”, Second Edition, 2018, pp. 119 – 138.

[93] Richard S, Sutton and Andrew G. Barto, “Reinforcement Learning: An Introduction - Bootstraping”, Second Edition, 2018, pp. 141 – 157.

[94] Richard S, Sutton and Andrew G. Barto, “Reinforcement Learning: An Introduction - Monte Carlo Methods”, Second Edition, 2018, pp. 91 – 115.

[95] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. MIT Press, 1998.

[96] https://medium.com/nerd-for-tech/temporal-difference-learning-in-reinforcement-learning-cf13ed159fcb [Accessed 24 April 2023].

[97] https://deeplizard.com/learn/video/qhRNvCVVJaA [Accessed 16 May 2023].

[98] https://github.com/Cloudslab/cloudsim [Accessed 21 December 2022].

7 Appendix A

7.1 Source Code

7.1.1 Data Centre Environment:

```
/**
         * Creates the datacentre.
         *
         * @return the datacentre
         */
        protected static NetworkOperatingSystem nos;
        protected static PowerUtilizationMaxHostInterface maxHostHandler = null;
        protected static SDNDatacenter createSDNDatacenter(String name, String physicalTopology,
NetworkOperatingSystem snos, VmAllocationPolicyFactory vmAllocationFactory) {
                // In order to get Host information, pre-create NOS.
                nos=snos;
                List<Host> hostList = nos.getHostList();
                String arch = "x86"; // system architecture
                String os = "Linux"; // operating system
                String vmm = "Xen";
                double time_zone = 10.0; // time zone this resource located
                double cost = 3.0; // the cost of using processing in this resource
                double costPerMem = 0.05; // the cost of using memory in this resource
                double costPerStorage = 0.001; // the cost of using storage in this
                                                                                // resource
                double costPerBw = 0.0; // the cost of using bw in this resource
                LinkedList<Storage> storageList = new LinkedList<Storage>(); // we are not adding
SAN

        // devices by now

                DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
                                arch, os, vmm, hostList, time_zone, cost, costPerMem,
                                costPerStorage, costPerBw);

                // Create Datacenter with previously set parameters
                SDNDatacenter datacenter = null;
                try {
                        VmAllocationPolicy vmPolicy = vmAllocationFactory.create(hostList);
                        maxHostHandler = (PowerUtilizationMaxHostInterface)vmPolicy;
                        datacenter = new SDNDatacenter(name, characteristics, vmPolicy,
storageList, 0, nos);

                        nos.setDatacenter(datacenter);
                } catch (Exception e) {
                        e.printStackTrace();
                }

                return datacenter;
        }

        /**
```

```
	 * Creates the broker.
	 *
	 * @return the datacenter broker
	 */
	protected static SDNBroker createBroker() {
		SDNBroker broker = null;
		try {
			broker = new SDNBroker("Broker");
		} catch (Exception e) {
			e.printStackTrace();
			return null;
		}
		return broker;
	}

	static String WORKLOAD_GROUP_FILENAME = "workload_10sec_100_default.csv"; // group 0~9
	static String WORKLOAD_GROUP_FILENAME_BG = "workload_10sec_100.csv"; // group 10~29
	static int WORKLOAD_GROUP_NUM = 50;
	static int WORKLOAD_GROUP_PRIORITY = 1;

	public static void submitGroupWorkloads(SDNBroker broker, int workloadsNum, int groupSeperateNum, String filename_suffix_group1, String filename_suffix_group2) {
		for(int set=0; set<workloadsNum; set++) {
			String filename = filename_suffix_group1;
			if(set>=groupSeperateNum)
				filename = filename_suffix_group2;
			filename = set+"_"+filename;
			broker.submitRequests(filename);
		}
	}
```

7.1.2 Training Model

```
	 void calculateQ() {
    Random rand = new Random();
    for (int i = 0; i < 1000; i++) { // Train cycles
      // Select random initial state
      int crtState = rand.nextInt(statesCount);
      while (!isFinalState(crtState)) {
        int[] actionsFromCurrentState = possibleActionsFromState(crtState);
        // Pick a random action from the ones possible
        int index = rand.nextInt(actionsFromCurrentState.length);
        int nextState = actionsFromCurrentState[index];
        // Q(state,action)= Q(state,action) + alpha * (R(state,action) + gamma * Max(next state, all actions) - Q(state,action))
        double q = Q[crtState][nextState];
        double maxQ = maxQ(nextState);
        int r = R[crtState][nextState];
```

```
            double value = q + alpha * (r + gamma * maxQ - q);
            Q[crtState][nextState] = value;
            crtState = nextState;
          }
        }
    }
```

7.1.3 Q-Learning Routing Algorithm

```
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package org.cloudbus.cloudsim.sdn.Qlearning_routeselection;
/**
 *
 * @author admin
 */
import java.util.*;
class Qlearning_routing_algorithm {
  public static void main(String[] args) {
    Qtable one = new Qtable(9, 4);
    Random random  = new Random();
    for(int i=0; i < 100; i++){
      int state  = random.nextInt(4);
      int action = random.nextInt(9);
      int nextState = random.nextInt(4);
      one.updateQTable(state, nextState, action, 0.9);
      one.print();
    }
  }
}
class Qtable {
          ArrayList<ArrayList<Double>> qValues = new ArrayList<ArrayList<Double>>();
          ArrayList<ArrayList<Double>> Rewards = new ArrayList<ArrayList<Double>>();
          int actions;
          int states;
          private final double learningRate = 0.1;
          private final double rewardDecay = 0.9;
          private final double greedy = 0.9;
          public Qtable(int actions, int states) {
            this.actions = actions;
            this.states = states;
            initializeQtable();
            initializeRewardsMatrix();
          }
          public void updateQTable(int currentState, int nextState, int action, double gamma){
```

```
            double reward = takeAction(currentState);
            qValues.get(currentState).set(action, qValues.get(currentState).get(action)
             + learningRate * (reward + (gamma * Collections.max(qValues.get(currentState)) -
qValues.get(currentState).get(action))) * 1.0);
          }
          public int getAction(){
            Random random = new Random();
            int actionIndex = random.nextInt(actions);
            return actionIndex;
          }
          public double takeAction(int currentState){
            int action = getAction();
            double stateActionReward = Rewards.get(currentState).get(action);
            return stateActionReward;
          }
          public void initializeQtable(){
            for (int i = 0; i < states; i++) {
              ArrayList<Double> statesColumn = new ArrayList<Double>();
              for (int j = 0; j < actions; j++) {
                statesColumn.add(0.0);
              }
              qValues.add(statesColumn);
            }
          }
          public void initializeRewardsMatrix() {
            //Attack or not rewards
            Rewards.add(new ArrayList<Double>(Arrays.asList(1.0, -0.3, -0.3, -0.3, -0.3, -0.3, -0.3, -0.3,
-0.3)));
            Rewards.add(new ArrayList<Double>(Arrays.asList(0.3, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5)));
            Rewards.add(new  ArrayList<Double>(Arrays.asList(-0.7,  0.7,  0.7,  0.7,  0.7,  0.7,  0.7,  0.7,
0.7)));
            Rewards.add(new ArrayList<Double>(Arrays.asList(-100.0, -100.0, -100.0, -100.0, -100.0, -
100.0, -100.0, -100.0, -100.0)));
          }
          public void print() {
            System.out.println(qValues);
          }
        }
```

7.1.4 Q-Learning and Q-Table Algorithms

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
```

```
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.Collections;
import java.util.Date;
import java.util.Random;
import org.junit.jupiter.params.shadow.com.univocity.parsers.csv.CsvWriter;
import environment.Environment;
import environment.Position;
public class QTable {
	State[] states;
	int nbstates;
	public double alpha = 1 ;
	public double gamma = 0.7 ;
	//gamma 0 = better solution
	//gamma 1 = faster solution
	int reward = 100; //rÃ©compense en atteignant l'objectif
	int penalty = -20; //sanction en cas d'actrion illÃ©gale
	//assimilable Ã  tableau en trois dimensions, dÃ©terminant la rÃ©compense, la position et la
finalitÃ© de l'action
	//selon une action ) partir d'un Ã©tat
	public double[][] Q;
	public int[][] R;
	public Position[][] P;
	boolean[][] done;

	public QTable(Environment map, double alpha, double gamma) {
		nbstates = (map.exitsnumber*6)*(map.lanes+2)*(map.exitsnumber);
		states = new State[nbstates];
		int largeur = map.exitsnumber*6;
		System.out.println("initialisation d'une table de "+nbstates+" etats");

		this.alpha = alpha;
		this.gamma = gamma;

		int c = 0;
		while(c <nbstates) {
			for(int o = 0; o<map.exitsnumber; o++) {//nombre d'objectifs possibles
				for(int l = 0; l<largeur; l++) {//parcours de la largeur
					for(int h = 0; h<map.lanes+2; h++) {//parcours de la hauteur
						states[c] = new State(new Position(l, h),
map.findPositionExit(o));
						c++;
					}
				}
			}
		}
```

```
		char[][] road = map.getRawRoad(); //servira Ã  detecter les cases illÃ©gales
		R = new int[nbstates][4];
		Q = new double[nbstates][4];
		P = new Position[nbstates][4];
		done = new boolean[nbstates][4];
		for(int i = 0; i <nbstates; i++)//initialisation de Q Ã  zero
			for(int j = 0; j <4; j++)
				Q[i][j] = 0;
		/**
		 * INITIALISATION ARBITRAIRE DE R et placement de P
		 * on dÃ©signe aussi la prochaine position correspondant Ã  chaque action
		 */
		for(int j = 0; j < 4; j++) {//chaque action
			for(int i = 0; i<nbstates; i++) {//pour chaque Ã©tat
				if(j == 0) { //HAUT
					if(states[i].pos.getY() == 0) {
						R[i][j] = penalty;
						P[i][j] = states[i].pos;
					}else {
							Position nextPos = new
Position(states[i].pos.getX(), states[i].pos.getY()-1);
							if(road[nextPos.getX()][nextPos.getY()] ==
'X') {
								R[i][j] = penalty;
							}else if(nextPos.getX() ==
states[i].objective.getX() && nextPos.getY() == states[i].objective.getY()) {
								R[i][j] = reward;
								done[i][j] = true;

							}else
								R[i][j] = -1;

							P[i][j] = nextPos;
					}
				}

				if(j == 1) { //DROITE
					Position nextPos;
					if(states[i].pos.getX() == largeur-1) {//bordure droite
						nextPos = new Position(0, states[i].pos.getY());
					}else {
						nextPos = new Position(states[i].pos.getX()+1,
states[i].pos.getY());
					}

					if(road[nextPos.getX()][nextPos.getY()] == 'X') {
						R[i][j] = penalty;
```

```
                                        }else if(nextPos.getX() == states[i].objective.getX() &&
nextPos.getY() == states[i].objective.getY()) {
                                                R[i][j] = reward;
                                                done[i][j] = true;
                                        }else {
                                                R[i][j] = -1;
                                        }
                                        P[i][j] = nextPos;
                                }
                                if(j == 2) { //BAS
                                        if(states[i].pos.getY() == map.lanes+1) {
                                                //action illÃ©gale OUTOFBORDER
                                                R[i][j] = penalty;
                                                P[i][j] = states[i].pos;
                                        }else {
                                                Position nextPos = new Position(states[i].pos.getX(),
states[i].pos.getY()+1);
                                                if(road[nextPos.getX()][nextPos.getY()] == 'X') {
                                                        //ation illÃ©gale
                                                        R[i][j] = penalty;
                                                }else if(nextPos.getX() == states[i].objective.getX()
&& nextPos.getY() == states[i].objective.getY()) {
                                                        R[i][j] = reward;
                                                        done[i][j] = true;
                                                }else {
                                                        R[i][j] = -1;
                                                }
                                                P[i][j] = nextPos;
                                        }
                                }
                                if(j == 3) { //GAUCHE
                                        Position nextPos;
                                        if(states[i].pos.getX() == 0) {//bordure gauche
                                                nextPos = new Position(largeur-1,
states[i].pos.getY());
                                        }else {
                                                nextPos = new Position(states[i].pos.getX()-1,
states[i].pos.getY());
                                        }
                                        if(road[nextPos.getX()][nextPos.getY()] == 'X') {
                                                //ation illÃ©gale
                                                R[i][j] = penalty;
                                        }else if(nextPos.getX() == states[i].objective.getX() &&
nextPos.getY() == states[i].objective.getY()) {
                                                R[i][j] = reward;
                                                done[i][j] = true;
                                        }else {
                                                R[i][j] = -1;
```

```
                                        }
                                        P[i][j] = nextPos;
                                }
                        }
                }
        }
        public int getIdOfState(State current) {
                for(int i = 0; i<nbstates; i++) {
                        if(states[i].objective.x == current.objective.x && states[i].objective.y ==
current.objective.y && states[i].pos.x == current.pos.x && states[i].pos.y == current.pos.y) {
                                return i;
                        }
                }
                System.out.println("ERROR state not found");
                return 0;
        }
        /**
         * @param id
         * @return valeur en tre 0 et 3 correspondant Ã  la meilleure action selon Q
         */
        public int getBestAction(int id) {
                ArrayList<Integer> list = new ArrayList<Integer>();
                double maxQ = Q[id][0];
                for(int i = 0; i<4; i++) {
                        if(Q[id][i] > maxQ) {
                                list.clear();
                                list.add(i);
                                maxQ = Q[id][i];
                        }else if(Q[id][i] == maxQ)
                                list.add(i);
                }
                if(list.size() == 1)
                        return list.get(0);
                else {
                        Random rand = new Random();
                        int n = rand.nextInt(list.size());
                        return list.get(n);
                }
        }
        /**
         * @param id of a state
         * @return the max Qvalue possible from a state
         */
        public double maxQ(int id) {
                double maxQ = -100;
                for(int i = 0; i<4; i++) {
                        if(Q[id][i] > maxQ) {
                                maxQ = Q[id][i];
```

```
                        }
                }
                return maxQ;
        }
        public void printString() {
                for(int i = 0;i<nbstates; i++) {
                        for(int j = 0; j<4; j++) {
                                System.out.println("Etat " +i+ " Action "+j+" Reward="+ Q[i][j]+ "
Pos "+P[i][j]);
                        }
                }
        }
        /**
         * updates QTable after a given state/action
         * @param action
         * @param state
         * @param next
         */
        public void updateQ(int action, int state, int next) {
                Q[state][action] = Q[state][action] + alpha*(R[state][action] + gamma*maxQ(next) -
Q[state][action]);
        }
}
```