IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 5 March 2024, accepted 6 April 2024, date of publication 10 April 2024, date of current version 10 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3387273

RESEARCH ARTICLE

Scalable Hybrid Switching-Driven Software Defined Networking Issue: From the Perspective of Reinforcement Learning Standpoint

MAX BLOSE[1], LATEEF ADESOLA AKINYEMI[2,7,8], (Senior Member, IEEE), STEPHEN OJO[3], (Member, IEEE), MUHAMMAD FAHEEM[4], (Member, IEEE), AGBOTINAME LUCKY IMOIZE[5], (Senior Member, IEEE), AND ARFAT AHMAD KHAN[6]

[1]Department of Electrical Engineering, University of Cape Town, Cape Town 7701, South Africa
[2]Department of Computer Science and Electrical Engineering, School of Computing and Engineering, College of Science, Engineering and Technology, University of South Africa, Johannesburg 1709, South Africa
[3]Department of Electrical and Computer Engineering, College of Engineering, Anderson University, Anderson, SC 29621, USA
[4]School of Computing (Innovations and Technology), University of Vaasa, 65200 Vaasa, Finland
[5]Department of Electrical and Electronics Engineering, Faculty of Engineering, University of Lagos, Akoka, Lagos 100213, Nigeria
[6]Department of Computer Science, College of Computing, Khon Kaen University, Khon Kaen 40002, Thailand
[7]Department of Electronic and Computer Engineering, Faculty of Engineering, Lagos State University, Epe Campus, Lagos 102101, Nigeria
[8]Centre for Augmented Intelligence and Data Science (CAIDS), University of South Africa, Johannesburg 1709, South Africa

Corresponding author: Muhammad Faheem (muhammad.faheem@uwasa.fi)

ABSTRACT The Software-Defined Networking technology promises to enhance network performance and cost reduction for service providers by providing scalability, flexibility, and programmability through the separation of the control plane from the data plane. However, the separation between the control plane and data plane in the implementation of SDN presents scalability issues, as the controller has limited computational resources. To address the SDN scalability issues identified, we create a scalable hybrid switching solution using machine learning algorithms. We propose an SDN OpenFlow model switch which collaborates 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 Bellman's equation to reiteratively update Q-values in the Q-table for every transition in the network environment state until Q-function has converged to the best Q-values. The proposed hybrid switching model is benchmarked against the standard SDN OpenFlow switch in terms of network performance metrics, including throughput, packet exchange transmission rates, CPU load, and delay. When statistically comparing simulation results, it is evident that the proposed switching model, incorporating machine learning algorithms, can effectively tackle scalability challenges in the design of SDN controller networks, especially in Data Centre environments where rapid switching speeds are crucial.

INDEX TERMS Controller, data centre, hybrid switching, machine learning, OpenFlow, scalability, SDN, sHRRL.

I. INTRODUCTION

In the realm of traditional Internet Protocol (IP) networks, there exist two interconnected planes known as the control plane and the data plane. These planes are seamlessly integrated within network devices like routers and switches. As depicted in Figure 1, the control plane assumes the responsibility of forwarding data packets and making decisions regarding their forwarding based on the switch table or routing table. Whenever a frame in layer 2 or a Packet

The associate editor coordinating the review of this manuscript and approving it for publication was Tao Zhou.

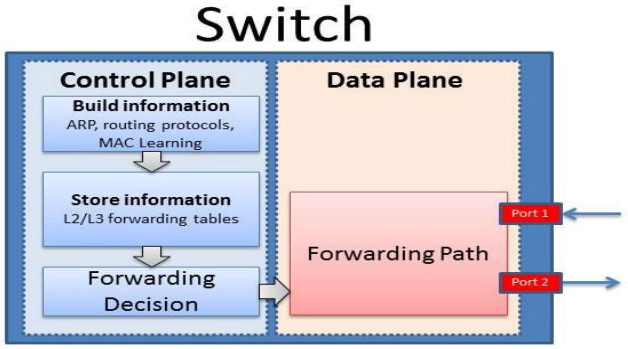


FIGURE 1. Traditional switch architecture [2].

in Layer 3 device is received from a switch port, the Media Access Control (MAC) address of the packet is stored in the Switch MAC address table, alongside the corresponding port number. The MAC address plays a crucial role in determining the transmission method of packets on the access media. Every switch or Network Interface Card (NIC) is given a distinct physical address known as the MAC address [1]. The source and destination MAC addresses found in the Ethernet frame header aid in identifying the sender and recipient of a data payload. A switch keeps the MAC addresses it has learned from other nearby switches in a MAC address table, often called a Content Addressable Memory (CAM) table, along with the port number that was used to obtain the MAC address. The switch must first verify the port and destination MAC addresses in its CAM table before forwarding an Ethernet frame to another network device. The switch will transfer the frame to all ports other than the one it was originally received on if the target MAC address is absent. Ethernet frame broadcasting only needs to be done once. The switch will know where to forward the packet the next time around once it has exchanged the initial MAC address. A traditional switch's distributed structure is shown in Figure 1, which also shows the data plane and switch control.

Traditional Layer 2 switches choose their paths based on a target address, in contrast to Layer 3 routing devices that carry out routing based on traffic.

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) [3], Open Shortest path first (OSPF) [4], and Interdomain Protocol Border Gateway Protocol version 4 (BGP-v4) [5]. 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.

A. TRADITIONAL NETWORK AND 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. Similarly, 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.

B. SOFTWARE-DEFINED NETWORKING

Software-Defined Networking (SDN) is a cutting-edge innovation technology developed by the Open Network Foundation (ONF) [6]. It advocates the decoupling of the control plane and data plane, with a control plane logically centralized. The primary objective of SDN 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 [7] illustrated in Figure 2.

The SDN method presents several significant benefits compared to conventional networking. These include a coherent central control structure that can effectively optimize network performance through its adaptable nature, programmability, and segregation of control and data planes. The scalability of the control plane constitutes a significant issue to be tackled in SDN due to its architectural framework featuring a logically centralized controller. The ongoing exchange of information between the control plane and the data plane may give rise to elevated communication costs,

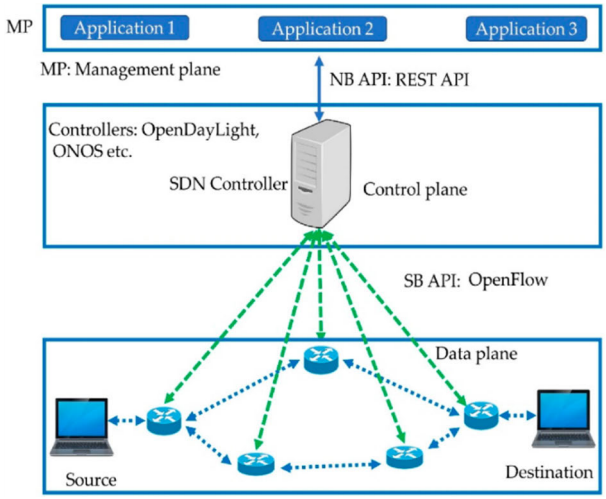


FIGURE 2. Overview of standard SDN architecture [10].

potentially causing a constraint in the expansion capability of the SDN controller. As an integral component of our research endeavour, we suggest the implementation of a Hybrid SDN model switching. This model serves as an expandable framework for combining hybrid routing techniques and Machine Learning. The objective of the design is to alleviate certain controller functions from the SDN controller by assigning them to infrastructure switches (data plane). This delegation is intended to enhance scalability by minimizing the latency involved in the flow set-up time by the SDN controller.

Within the traditional IP network framework, there are two closely interconnected planes: the control plane and the data plane. These planes are seamlessly integrated into network devices such as routers and switches. As illustrated in Figure 1, the control plane assumes the responsibility of forwarding data packets and making forwarding decisions based on either 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 acquired. The MAC address plays a pivotal role in determining the transmission method employed for packets on the access media.

The Applications Layer plays a crucial role in enabling the network operator to efficiently manage both the network elements and the network as a whole. This management is made possible through the use of protocols such as Representational State Transfer (REST) Applications Programming Interfaces (APIs), Simple Network Management Protocol (SNMP), and the Network Configuration Protocol (NETCONF) [8]. Within the SDN framework, APIs are utilized to facilitate communication between the different layers of SDN [9]. Specifically, the North-Bound Applications Programming Interface (API) serves as the interface that enables communication between the application layer and the control layer.

The South-Bound API is an additional interface that uses the open flow protocol to create a connection between the infrastructure layer and the control layer. This makes it possible for OpenFlow switches and routers, among other infrastructure layer devices, to communicate with the SDN controller.

OpenFlow represents the southbound protocol, as specified by the designated Open Network Foundation standard [11]. This specific protocol serves the purpose of transmitting a control logic function between a switch and a controller. The OpenFlow switch [12] is composed of three primary modules. These flow tables contain an action field associated with each entry in the table. The second module acts as a communication channel, facilitating the exchange of commands and packets between a controller and the switch. The third protocol is the OpenFlow protocol, which is responsible for establishing a communication channel between an OpenFlow controller and the routing and switching devices located in the data plane of the infrastructure layer.

OpenFlow is extensively utilized in the switches of SDN. SDN switches serve as integral elements of SDN architecture and bear the responsibility of directing the incoming flows towards their designated endpoints. The process of SDN flow forwarding operates by the established flow rules. To that end, each switch is equipped with a flow table. OpenFlow serves as the API which is implemented to bridge the gap between the data and control planes [11]. The SDN switch possesses a flow table and channel components. Communication is initiated between controllers and switches through this channel.

The Hybrid Switch is an innovative design that offers a viable solution for enhancing the scalability of SDN [13]. It is important to note that the term Hybrid Switch may lead to confusion as it bears a resemblance to a stacked switch. Stacked switches are a combination of SDN switches and traditional layer 2 switches, where one mode is completely turned off while the other is in operation. However, the proposed Hybrid Switch involves a comprehensive integration of the SDN open-flow switch with a conventional switch. The main objective is to enable seamless interaction and support between the two switches throughout their operational processes, while effectively expanding the SDN network.

C. SDN CHALLENGES

There is undoubtedly a consensus that software-defined networking presents a novel paradigm, which brings forth a plethora of benefits for the advancement and progress of cloud and computer networking. Nevertheless, SDN encounters a multitude of obstacles. In the subsequent section, we shall deliberate upon a few of the difficulties linked with SDN.

D. AVAILABILITY CHALLENGES

Data plane devices, such as switches, depend on a logically centralized controller to alter traffic flow rules. If an SDN controller deployment approach is used, there is a possibility that the control layer could become a single point of failure. This would result in the infrastructure layer devices being left disconnected and isolated if the connection to the control

layer is lost. Hence, it is advisable to employ a decentralized controller approach to ensure a robust level of availability for the control layer.

E. FLEXIBILITY CHALLENGES

Network flexibility refers to the ability of a network to adapt to unanticipated alterations promptly and efficiently in the network, such as the failure of a route link or the need for re-convergence, within a limited time frame. One of the foremost challenges associated with Software Defined Networking lies in effectively managing packet routing and optimizing flow processing. The prioritization of flow rules between the switches and controllers via the OpenFlow interface enables the attainment of flexibility in SDN. Actions are executed according to the corresponding packet header fields. The packets have the option to be either transmitted through the switch port, modified, or discarded, with the flow rules being stored in the hardware memory of the OpenFlow switch. Regrettably, the updating of the flow table in networks with extensive dynamic routing policies can be hindered by the slowness of Ternary Content-Addressable Memory (TCAM), causing bottlenecks, and compromising flexibility in the control plane. These obstacles could be eradicated through the utilization of the forthcoming generation of switches.

F. SECURITY CHALLENGES

The security threats commonly acknowledged concerning SDN systems primarily originate from conventional network architectures. The architecture of SDN systems requires a distinct division between the control layer and the infrastructure layer, resulting in various security concerns.

One illustration of a cyber assault that SDN is susceptible to would be the Distributed Denial of Service (DDoS) targeting the control layer [14]. This malicious assault has the potential to be launched from diverse devices within the infrastructure layer, overwhelming the control layer with an influx of queries. The controller may experience latency or packet drop due to the excessive number of queries it receives. One way to address this kind of attack is to mitigate it through the implementation of multi-SDN controllers. In this approach, a single controller can be designated as the primary and effectively distribute the workload of queries from the switches to other SDN controllers.

Man-in-the-middle attacks, also known as Multipurpose Internet Mail Extension (MIME) attacks, have the potential to exploit the communication messages exchanged between the controllers and the devices in the infrastructure layer. The assailant possesses the capability to modify the rules governing the communication between the controllers and the switches in the data layer. Subsequently, they can seize authority over both the data layer and the controllers. To prevent such kind of attack, it is advised to deploy fortified integrity measures, and strong authentication protocols, and encrypt the controller-to-OpenFlow switches communication with digital signatures.

The susceptibility of the application layer towards cyber-attacks arises from its inherent vulnerability in terms of programmability. An assailant can seize control over network traffic by gaining access to secure applications within SDN, thereby jeopardizing the integrity of the network infrastructure. It is imperative to adhere to the most effective security protocols.

G. SCALABILITY CHALLENGES

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

SDN has a notable limitation in the form of a centralized controller, which can give rise to considerable scalability challenges. As an illustration, a study conducted by Erickson [16] suggests that in a worst-case scenario, a network comprising one hundred (100) switches can experience a flow rate of ten million. Furthermore, according to the research conducted by Kanduala et al. [15], a cluster consisting of 1,500 servers can encounter an average of 100,000 flows per second. The results indicate that the SDN's control plane is prone to encountering scalability challenges as a result of its inherently centralized controller architecture.

SDN presents a significant challenge to achieving scalability. This is primarily because the infrastructure layer relies on a centralized controller to make decisions regarding flow rule settings, necessitating continuous communication between these two layers. The nature of this communication, which can involve continuous signalling messages in centralized, distributed, or hierarchical SDN network architectures, introduces substantial overheads. Consequently, these overheads can become a bottleneck and impede the scalability of the control layer. The scalability of the infrastructure layer devices can be compromised when a centralized controller design is employed to handle a substantial volume of requests.

This network expansion can lead to a rise in the volume of flow requests that need to be managed by the controller, potentially causing a bottleneck due to the limited computing and memory capacity of the controller's central processing unit. The NOX is one of the first OpenFlow controllers of the SDN ecosystem, developed at Nicira Networks. NOX for instance, is not suitable for medium-to-large network setups like a Data Centre network infrastructure as it can only handle a maximum flow request rate of 30 thousand requests per second [17].

Several studies have been conducted to address the scalability challenges faced by data planes in SDN. These studies primarily focus on factors such as buffers, memory capacity, and processing power [18], [19], [20]. However, it is important to note that this research does not encompass the SDN data plane. Instead, the research concentrates on enhancing the scalability of the SDN control plane within a Data Centre

network environment. The primary objective of this research is to analyze the throughput of the control plane, which includes evaluating the number of requests handled per second by the controller, the time required to set up flow tables, the delay associated with responding to flow requests, and the impact of the Central Processing Unit (CPU) on controller response time.

H. SCALABILITY METRICS

The notion of scalability can be perceived differently by various audiences, lacking a precise definition. Nevertheless, in the realm of computing, it can be understood as the ability of a system, network device, or process to effectively manage a growing workload. Additionally, it can be seen as the capability to expand and accommodate future growth [21].

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 [22], [23], [24], [25], [26], [27], suggest Isospeed and Isoefficient Scalability metrics in homogeneous, heterogeneous, or hybrid settings. For instance, a study conducted by Hwang and Xu [24] 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, Sun et al. [22] provide a metric to characterize the scalability function of a homogeneous system, defined as:

$$\psi(p, p') = \frac{p'W}{pW'}, \tag{1}$$

In Eq. (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. [28] 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:

$$E = \frac{s}{p}, \tag{2}$$

The speed-up S and the number of computing nodes denoted by $p$are 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)

$$s = \frac{W}{T_p}, \tag{3}$$

where $T_p = \frac{W+T_o(w,p)}{T_p}$ with $T_o(w, p)$ denotes extra communication overhead [23]. From the Software Defined Networking perspective, the separation of the infrastructure layer and the control layer is 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 [29]. 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 Eq. (4) as per [30]:

$$F\,(k) = \varphi\,(N)\,x\frac{T(N)}{C(N)}, \tag{4}$$

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:

$$\psi\,(N_1, N_2) = \frac{F(N_2)}{F(N_1)}, \tag{5}$$

I. CONTRIBUTIONS

Below is a summary of this paper's primary contributions.

- Review scalability challenges introduced by Software Data Networking's decoupling of the control plane and data forwarding plane.
- To develop a Scalable framework of Hybrid Routing with Reinforcement Learning (sHRRL), a collaboration mechanism between the 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.

J. ORGANIZATION

This paper is organized into the following sections: Section II provides an analysis of previous studies on Hybrid Switching in SDN. Section III presents a system model and problem formulation. The proposed solution and algorithms are discussed in Section IV. Section V presents the simulation results and analysis. Finally, in Section VI, concluding remarks and observations are provided.

II. RELATED WORKS

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 on the scalability of the SDN, which continues to be one of the major topics of research in the software-defined networking field. Many research papers have been published proposing various approaches to address SDN scalability issues. These include DIFANE [31], DevoFlow [32], Devolved Flow [33], and Kandoo [33]. DIFANE is a distributed flow architecture that reduces network overhead by reducing control traffic sent across the OpenFlow protocol interface. In DIFANE, wild-card 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 packet flow rules. Authors of Software-defined converters in [34], Hyper Flow [35], Maple [36], ONIXN [37], and Maestro [38] have proposed similar solutions as DevoFlow to address SDN scalability challenges.

Different SDN controller designs such as the centralized, distributed, hierarchical, and hybrid controller design approaches have been proposed in previous research efforts. In a *centralized controller design strategy*, a single controller takes full responsibility for managing all network devices within its domain. The centralized controller model is not considered 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 [32], [33]. The distributed model further splits the controllers into those which have local management responsibility and those with global domain management responsibility roles. The controllers assigned local management functions have limited knowledge or exchange of routing and control information about other local controllers. The controllers at the global layer have the 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 the 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 for being a root controller with a full view of all lower layers including the infrastructure layer. However, the hierarchical controller model has the disadvantage of length flow paths [34] 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.

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.

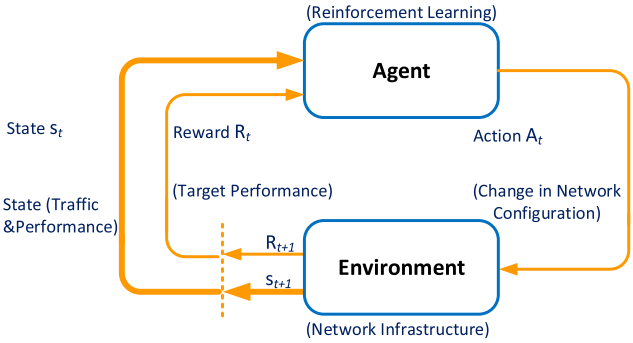


FIGURE 3. System Model Illustration [46].

Wang et al. [39] showed how a Reinforcement Learning technique could be used for routing, link cost calculation and path selection. However, their proposal only focused on delay optimization. Research efforts in [40] and [41] suggested improving traditional routing protocols by using the SDN innovation. However, their work didn't use network operation information to get smarter routing. Martin et al. in [42] employed a 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 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, following the findings in [43] following a similar research methodology.

III. SYSTEM MODEL AND PROBLEM FORMULATION

Presented in this section is a system model that utilizes a reinforcement learning algorithm to facilitate the development of a scalable hybrid switching system based on software-defined networking.

A. REINFORCEMENT LEARNING MODEL

Reinforcement learning algorithms, as discussed in [44], employ a value function to guide the agent in selecting the most favourable routes. This value function estimates the desirability of the agent being in a specific state. The reinforcement learning process can be mathematically represented as a Markov Decision Process (MDP) [45]. MDPs are a type of stochastic decision-making framework that models the decision-making process of a dynamic system within an environment. In this framework, the output can be either random or controlled by a decision-maker who iteratively makes successive decisions. Markov Decision Process, where there are a finite number of states and actions, is characterized by a finite set of *Actions* A, a finite set of *States*S, a *Reward* function $R : S \times A \longrightarrow \Re$ and a state transition function $T : S \times A \times S \longrightarrow \Re$ where $T\left(s, a, s'\right)$ 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 behavior 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 [47]. It is denoted by the symbol π. An agent follows a policy, meaning if an agent follows policy π at time t, then π (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 π $(a|s)$. 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 Eq. (6), The equation depicts a sum of rewards received over finite horizon h.

$$R_t = \sum_{k=0}^{h} r_{t+1}, \tag{6}$$

However, other tasks tend to be infinite and may need to be resolved through future reward discounts as per Eq. (7).

$$R_t = \sum_{k=0}^{\alpha} \gamma^k r_{t+1}, \tag{7}$$

where Υ represents a discount rate between $0 \leq \gamma < 1$. The purpose of a 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:

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\}, \tag{8}$$

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

$$V^{\pi}(s) = \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right\}, \tag{9}$$

We maximizeV^{π} the current and future states through the optimal value function V^* as per Eq. (10).

$$V^*(s) = \max_{\pi} V^{\pi}(\forall s), \tag{10}$$

The optimal policy π^* can be defined as the best action to take at each state of the environment to maximize the returns over time [48] and can be expressed by Eq. (11) as:

$$\pi^* = argmax V^{\pi}(\forall s), \tag{11}$$

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 [47]. To identify the ideal value function, we can use value iteration, a dynamic programming technique [47]. 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 Eq. (12) as:

$$\pi^*(s) = argmax V^*(s'), \tag{12}$$

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 [49], where we do not have earlier information on the environment flows. Instead, an agent needs to learn from the environment through the rewards experienced by taking diverse activities.

B. TEMPORAL-DIFFERENCE LEARNING

One of the alternatives we can utilize to learn about the environment is through Temporal-Difference learning [50]. Richard et al. [51], 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. Temporal-difference learning intends 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 [51] and Dynamic Programming learning strategy. Monte Carlo strategy alters their estimates after the ultimate output is received, while Temporal-Difference learning alters expectations to coordinate afterwards, more precisely, expectations of the future before the ultimate result is known.

The Temporal-Difference learning challenge can be solved through the evaluation of the 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 in [51], and update the reinforcement learning agent on the present estimations related to Dynamic programming methods in [49]. Temporal-Different training algorithms update present estimate V(s_t) through value function estimates of temporally successive states discussed in [50]. 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 Eq. (13) as follows:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right], \tag{13}$$

Different learning algorithms can be used for the Temporal-Difference learning method. The most common algorithms are the State Action Reward State Action (SARSA) algorithm [52], 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

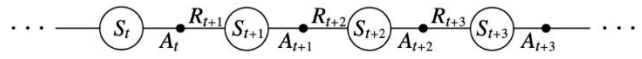


FIGURE 4. SARSA state-action transition [53].

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 [50]. In Fig 4, 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.

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 relies 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 approaches "0". In the next section, we explore the Q-learning approach, which is in line with our research effort.

C. Q-LEARNING

According to Watkins and Dayan [54], 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 [55] presented in Eq. (14) 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 [55]. $A = \pi r^2$

$$V(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} s' \left(P\left(s,a,s'\right) V\left(s'\right) \right) \right) \quad (14)$$

In Bellman Eq. (14), we represent s as a particular state in our network environment, a denotes an action to be taken to route a packet, $s'$is the state to which a packet is routed from the state s, γ, denotes a discount rate, $R\,(s,a)$ is returned reward value after a transition to a state as a result of an action a selected by the agent. $V\,(s)$is a value of being in a specific state s, could be the best-taken choice 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 Eq. (12), 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 Eq. (10), as the anticipated reward for choosing action a amid state s.

$$Q^*\,(s,a) = R\,(s,a) + \gamma \sum_{s'} \left(P\left(s,a,s'\right) V^*(s')\right), \quad (15)$$

We can utilize the condition in Eq. (15) to measure the quality of a specific action selected by the agent. In Eq. (10) 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:

$$V^*\,(s) = \max_a Q^*(s,a), \quad (16)$$

As a result, we can recursively rewrite Eq. (15) as follows:

$$Q^*\,(s,a) = R\,(s,a) + \gamma \sum_{s'} \left(P\left(s,a,s'\right) \max_{a'} Q^*\left(s',a'\right) \right), \quad (17)$$

According to Watkins and Dayan [54], whenever temporal difference (0) is utilized in predicting the expected reward of the state under the policy π, Q-learning successively evaluates the best action-value function $Q^*\,(s,a)$. 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 state 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 Eq. (17) and subtract the previous $Q^*\,(s,a)$ from it.

$$TD_t\,(s,a) = R\,(s,a) + \gamma \sum_{s'} \left(P\left(s,a,s'\right) \max_{a'} Q^*(s',a') \right) - Q^*\,(s,a)\,, \quad (18)$$

This in turn gives us a new $Q^*\,(s,a)$. Eq. (18) 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 (19) as follows:

$$Q_t\,(s,a) = Q_{t-1}\,(s,a) + \alpha TD_t\,(s,a)\,, \quad (19)$$

Algorithm 1: $Q-Learning$

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., $\varepsilon-$ greedy)

7: Take action A, observe R, S'

8: $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma max_a\, Q(S', a) - Q(S, A)]$

9: $S \leftarrow S'$

10: Until S is terminal

FIGURE 5. Estimating Q^* with Q-learning algorithm.

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, we can mathematically characterize $TD_t\,(s, a)$ into Eq. (20) which defines the Q-learning algorithm.

$$Q_t\,(s,a) = Q_{t-1}\,(s,a) + \alpha(R\,(s,a) + \gamma \max_{a'} Q\left(s', a'\right) - Q_{t-1}\,(s,a)), \quad (20)$$

where $R\,(s,a) + \gamma \max_{a'} Q\left(s', a'\right) - Q_{t-1}\,(s,a)$ represents learned value and $Q_{t-1}\,(s,a)$ denotes old value.

IV. PROPOSED SOLUTION

In figure 5, 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 $\in$ -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 $1-\ \in$ and a random exploration of the network environment by choosing an action based on the current state with insignificant probability $\in$. 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.

A. EXPLOITATION AND EXPLORATION

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 [54] is used. Epsilon greedy strategy defines an exploration rate $\in$ to be initiated to one. The exploration rate $\in$ is the possibility that the agent will randomly discover the environment rather than exploit the already-known information about the network environment. When an exploration rate $\in= 1$, 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 $\in$ 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. 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 exploring the current network environment state within the next time step.

B. 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 Eq. (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 the Bellman equation.

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

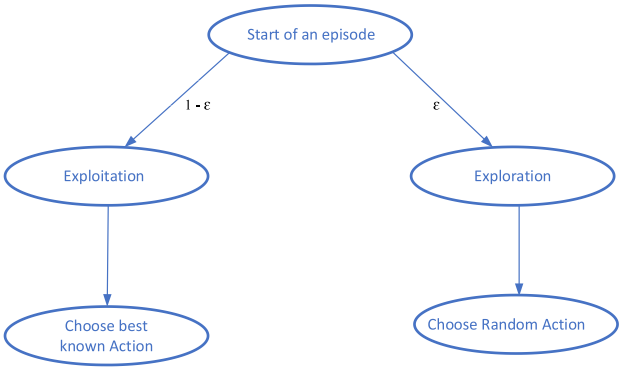


FIGURE 6. Exploration and exploitation.

Algorithm 2: Epsilon − Greedy Q − Learning Algorithm

Data: α: *Learning rate,* γ: *Discount rate,* ϵ: *A small number*

Results: $A -$ *Table containing* $Q(s,a)$ *pairs defining estmated optimal policy* π^*

Initialize $Q(s,a)$ *arbitrarily, except* $Q(terminal)$;

$Q(terminal) \leftarrow 0$;

for each episode do

  Initialize state s

  for each step in the epistode do

    do

      $a \leftarrow select - action(Q,s,\epsilon)$;

      Take action a, then observe reward R and next state s′

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

      $s \leftarrow s'$;

  while s is not terminal;

end;

end;

FIGURE 7. Q-Learning algorithm.

action to be selected to route the packet from the source node to the destination node.

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.

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 greedy policy $\in$. 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 8.

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 8.

C. HYBRID SDN DESIGN MODEL

We follow the same concept and similar findings presented in [46] for the Hybrid-SDN model topology design depicted in the communication diagram between the controller and switches as per Figure 9. We implement a system of collaboration between the infrastructure layer of traditional switches and the software-defined networking OpenFlow switches.

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 values 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.

D. 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

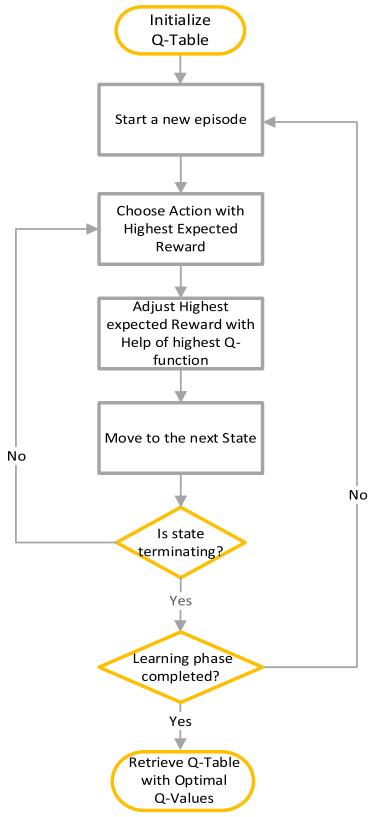


FIGURE 8. Q-Learning routing algorithm flowchart.

traffic forwarding with the aid of the proposed Q-learning model which reinforces the best path routing process for forwarding network switches.

E. STATE SPACE DESIGN APPROACH

The reinforcement learning agent collaborates with the software-defined networking controller to manage the network environment holistically. 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 possible path in the routing table is represented by a Q-value in the state space.

F. 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 with routing packets via the current path.

G. REWARDS DESIGN APPROACH

In this work, we consider the delay for a reward when sending a packet from a source node to a destination node.

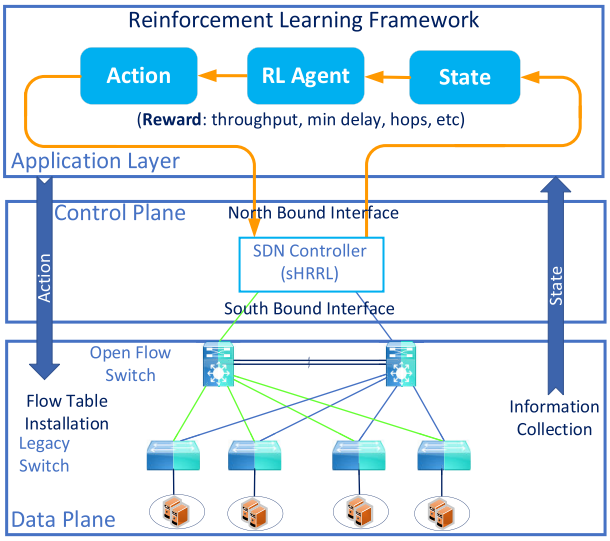


FIGURE 9. Hybrid SDN topology with hosts, switches, controller, and RL framework [46].

To evaluate the optimal actions 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.

H. 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 the 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, source and destination parameters, and network size overseen by the controller.

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 $\in$-greedy policy$\in$. We execute the action a_t and observe the returned reward r_{t+1}, followed by a transition to a 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 several episodes are completed, we get the Q table in figure 10.

In the Q-Table, the columns are used as actions and the rows are used as states. To construct the Q-table, a

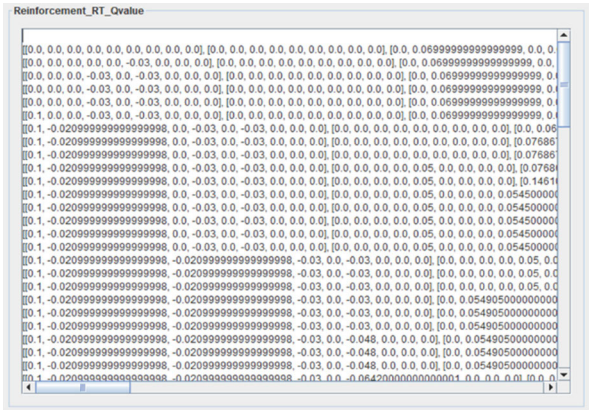


FIGURE 10. Q-values table.

FIGURE 11. SDN Virtual topology construction.

9 × 4 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 to 1. 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 concerning 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. The greedy policy is understood to be the updating policy, while the Epsilon greedy policy is considered to be the acting policy.

I. TOOLS

The hardware and software components utilized in this experiment are presented in Table 1. To meet the objectives of this research effort, an open-source framework, CloudSim [56], 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. 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 a Q-learning computation approach was employed for optimal routing. In this CloudSim topology model, all the hosts were virtually connected to switches in a Data Centre, as shown in Fig. 11.

We use the Hybrid Controller design methodology in this work because it has the benefit 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.

V. RESULTS ANALYSIS AND DISCUSSION

The overall performance of the proposed hybrid switching technique is evaluated between the open-flow switch and the Hybrid switch. The analysis is based on network performance metrics including average throughput, packet exchange transmission rate, CPU load and delay. The results obtained from the simulation are analyzed statistically and compared by plotting performance results in a graph.

A. THROUGHPUT

The assessment was initiated by measuring the flow rate of the path at 500Kbps, with a gradual increase of 500Kbps up to 5Mbps, to evaluate the throughput. The results of the throughput, as shown in Figure 12 and Figure 16, indicate that the Hybrid Switch routing scheme achieves similar scalability results compared to the Open Flow switching scheme. This similarity can be attributed to the utilization of the shortest path routing algorithm by both schemes. However, the Open Flow Switch is the first to reach its saturation point as it does not utilize all available paths, thereby being surpassed by the Hybrid Switching approach in terms of the received flow rate.

B. CPU LOAD

The comparison of CPU utilization between the OpenFlow Controller and Hybrid Switch is further explored in Figure 13 and Figure 17. Once again, the Hybrid Switch demonstrates superior performance in terms of CPU utilization compared to the OpenFlow Switch. This notable increase in CPU usage can be attributed to the autonomous establishment of switch flows, eliminating the need for the switch control plane to engage in any calculations for the establishment of new paths.

C. PACKET EXCHANGED

The evaluation, as depicted in Figure 15 and Figure 19, commenced with a path flow rate of 78 bytes and gradually

TABLE 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
	2D Pie Charts Generated from Microsoft Excel, using data generated from CloudSim software.

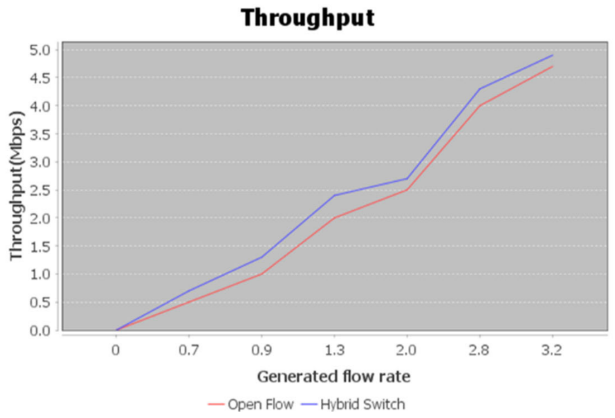


FIGURE 12. Average throughput.

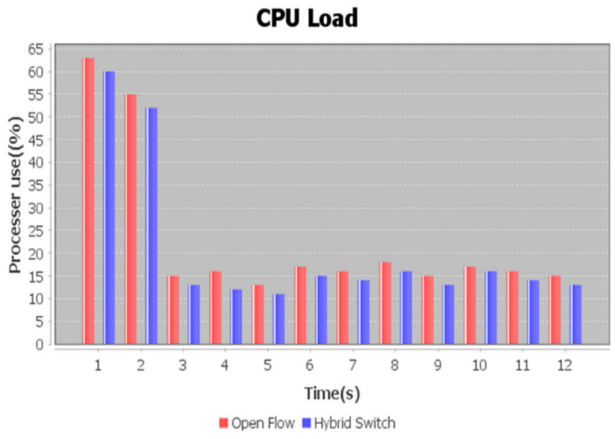


FIGURE 13. Average CPU load.

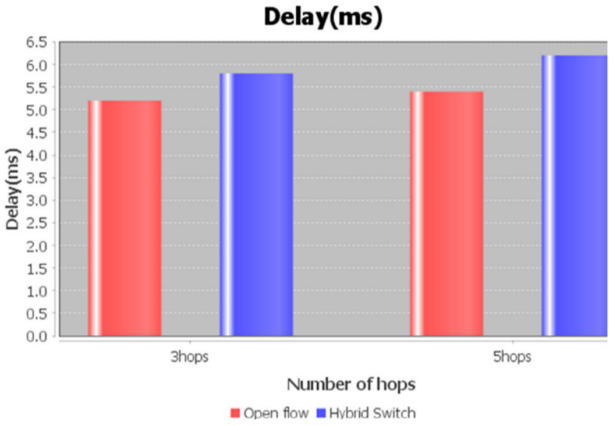


FIGURE 14. Average path flow set-up time.

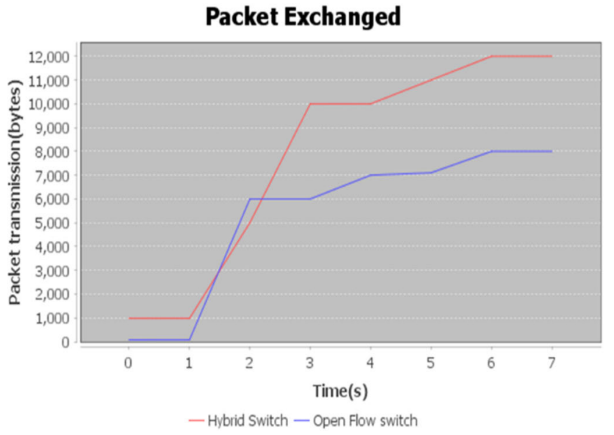


FIGURE 15. Average packet exchange rate.

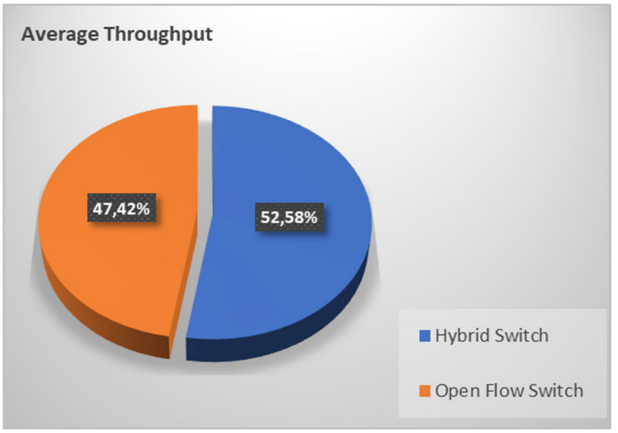


FIGURE 16. Average throughput ratio.

increased to 12,000 bytes or 12kbytes over approximately 7 seconds. Notably, during the exchange of the initial network packets, the OpenFlow switch's processed packet count experienced a peak between the one and two-second intervals. This occurrence can be attributed to the hybrid switch's requirement to initiate the execution of the sHRRL training framework. Within this framework, the Q-function updates the Q-table with newly calculated Q-values to determine the optimal route for packet routing between the source and destination nodes. The ultimate objective is to achieve an optimal reward from the SDN controller. It is worth highlighting that the hybrid switch transmitted a higher number of packets, surpassing the count by more than 18%.

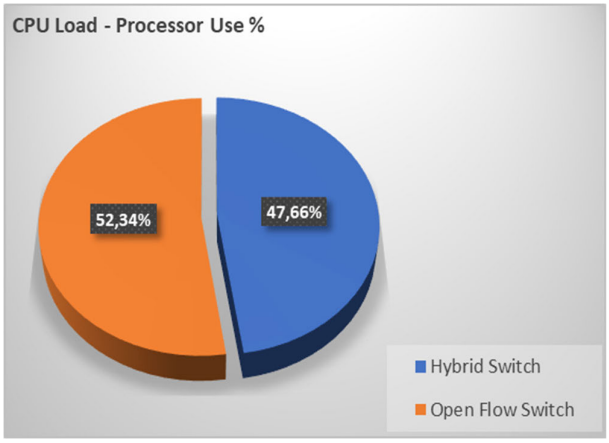


FIGURE 17. CPU Load – processor usage ratio.

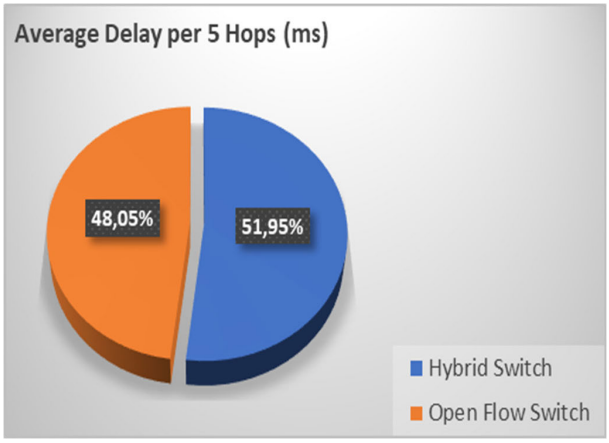


FIGURE 18. Average latency ratio.

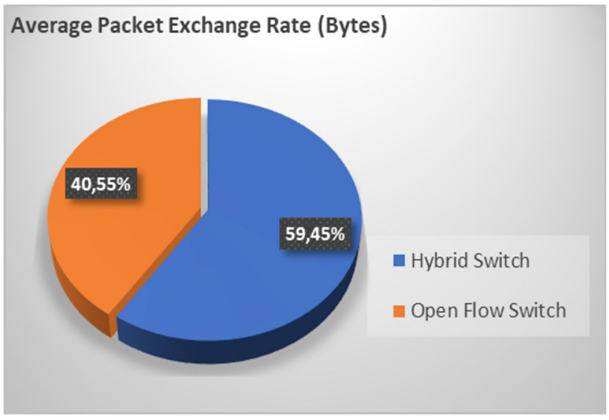


FIGURE 19. Average packet exchange ratio.

D. DELAY

The average time taken to establish a traffic flow configuration after conducting a simulation to transmit packets from the source to the destination node is represented in Figure 14 and Figure 18. Hybrid switches demonstrate a relatively high latency, although it remains relatively low. This delay is attributed to the interaction between the hybrid switch, SDN OpenFlow, and traditional switches before initiating traffic routing within the network.

VI. CONCLUSION

Machine learning in Software-Defined Networking is destined to be a key-factor in enabling efficient dynamic routing of applications. However, several obstacles 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 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 an 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 change in the network environment. To route data packets from the originating node to the receiving node, certain function metrics were considered, 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 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 supports hybrid switching in optimal path selection. Unlike the OpenFlow switch where the 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.

VII. FUTURE WORKS

Although the presented simulation results demonstrated significant improvement in scalability, they can still be developed and explored in different scenarios. For example, future work can be considered in enhancing scalability through additional routing algorithms such as deep reinforcement learning algorithms. We can also reduce model training time through additional code with prior knowledge of the network environment, and reduction of epsilon value with every episode. This can result in less exploration and more exploitation, from the trade-off presented in section IV. An open-source framework called CloudSim was used as a test bed for modelling and simulating our experiment. In our routing algorithm simulation through CloudSim, all our nodes, hosts and access links had similar characteristics, including bandwidth capacities. We believe that precise test results may be achieved if a more genuine or physical network infrastructure with different applications and traffic models prioritized in different classes of service, was utilized. SDN controllers can be associated with different types of threats and malicious attacks. These include network attacks such as Botnets, Denial of Service and distributed denial of service, man-in-the-middle attacks and many others against the SDN controller and OpenFlow switches. This study can be further enhanced by looking at the options on how the security aspects of SDN controllers and switches can be mitigated against such threats. To demonstrate the robustness of the proposed routing algorithm, the recovery time and routing performance after link failure using Hybrid Switching can be explored further. The network operation after a disturbance due to a certain failure, the time taken to correct itself to come back to normal operation can be improved to be as smaller as possible and that can be achieved by controller and hybrid switching working together. Due to limited time, we left all these fascinating challenges for future work efforts.

VIII. ANNEXURE A

A. LIST OF ABBREVIATIONS

DevoFlow: Devolved Flow is a fork of the OpenFlow model that gently disconnects control from global visibility while maintaining a high level of visibility without increasing overhead.

HyperFlow: HyperFlow provides scalability while keeping network control logically centralized.

DIFANE: DIstributed Flow Architecture for Networked Enterprises

Maple: Maple is an SDN programming system that introduces novel components to make SDN programming with algorithmic policies scalable.

*ONIXN:*An end-to-end distributed network solution for large-scale production networks, designed to address issues such as global network visibility, fault recovery, and consistent network status distribution.

Maestro: This is the network operating system for managing network control applications. Maestro interfaces enable the implementation of modular network control applications that can access and change the status of the network and manage their interactions across multiple protocols, such as OpenFlow.

NOX: This is one of the first OpenFlow controllers of the SDN ecosystem, developed at Nicira Networks.

ACKNOWLEDGMENT

The authors would like to thank their affiliated universities for providing research funding to accomplish this study.

REFERENCES

[1] Cisco Networking Academy. (Mar. 31, 2014). *Introduction to Basic Switching Concepts and Configuration*. Accessed: Mar. 10, 2023. [Online]. Available: https://www.ciscopress.com/articles/article.asp?p=2181836
[2] *Implementation of SDN using OpenDayLight Controller*. Accessed: Apr. 30, 2023. [Online]. Available: https://www.researchgate.net/publication/320347086_Implementation_of_SDN_using_OpenDayLight_Controller/figures?lo=1
[3] *RIPv2*. Accessed: Jun. 14, 2023. [Online]. Available: https://tools.ietf.org/html/rfc1058
[4] *OSPFv2*. Accessed: Jun. 14, 2023. [Online]. Available: https://tools.ietf.org/html/rfc2178
[5] *BGP-4*. Accessed: Jun. 14, 2023. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1654
[6] F. Benamrane, M. Ben Mamoun, and R. Benaini, "Performances of OpenFlow-based software-defined networks: An overview," *J. Netw.*, vol. 10, no. 6, pp. 329–337, Jun. 2015, doi: 10.4304/jnw.10.6.329-337.
[7] O. Blial, M. Ben Mamoun, and R. Benaini, "An overview on SDN architectures with multiple controllers," *J. Comput. Netw. Commun.*, vol. 2016, pp. 1–8, Sep. 2016, doi: 10.1155/2016/9396525.
[8] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Inf. Comput. Sci., Inst. Softw. Res., Univ. California, Irvine, CA, USA, 2000. [Online]. Available: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm
[9] *Traditional Vs Software Defined Networking*. Accessed: Apr. 26, 2019. [Online]. Available: http://www.rfwireless-world.com/Terminology/traditional-networking-vs-software-defined-networking.html
[10] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in SDN-OpenFlow networks," *Comput. Netw.*, vol. 71, pp. 1–30, Oct. 2014, doi: 10.1016/j.comnet.2014.06.002.
[11] Open Networking Foundation. (2014). *Openflow Table Type Patterns*. Accessed: Mar. 12, 2020. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdnresources/onf-specifications/openflow/OpenFlowAugust

[12] *Ruckus FastIron SDN Configuration Guide*, Ruckus, Sunnyvale, CA, USA, Release 08.0.61, Jun. 2018.

[13] A. Lee and X. Wang, "A hybrid software defined networking architecture for next-generation IoTs," *KSII Trans. Internet Inf. Syst.*, vol. 12, no. 2, pp. 932–945, Feb. 2018, doi: 10.3837/tiis.2018.02.024.

[14] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Software-defined networking security: Pros and cons," *IEEE Commun. Mag.*, vol. 53, no. 6, pp. 73–79, Jun. 2015, doi: 10.1109/MCOM.2015.7120048.

[15] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data centre traffic: Measurements & analysis," in *Proc. 9th ACM SIGCOMM Conf. Internet Meas. Conf. (IMC)*. New York, NY, USA: ACM, 2009, pp. 202–208, doi: 10.1145/1644893.1644918.

[16] D. Erickson, "The beacon openflow controller," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, Aug. 2013, pp. 13–18, doi: 10.1145/2491185.2491189.

[17] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of SDN/OpenFlow controllers," in *Proc. 9th Central Eastern Eur. Softw. Eng. Conf. Russia*. New York, NY, USA: ACM, Oct. 2013, p. 1, doi: 10.1145/2556610.2556621.

[18] R. Narayanan, S. Kotha, G. Lin, A. Khan, S. Rizvi, W. Javed, H. Khan, and S. A. Khayam, "Microflows and microflows: Enabling rapid network innovation through a split SDN data plane," in *Proc. Eur. Workshop Softw. Defined Netw.*, 2012, pp. 79–84, doi: 10.1109/EWSDN2012.16.

[19] K. Kannan and S. Banerjee, *Compact TCAM: Flow Entry Compaction in TCAM for Power Aware SDN*. Berlin, Germany: Springer, 2013, pp. 439–444, doi: 10.1007/978-3-642-35668-1_32.

[20] Q. Zuo, M. Chen, K. Ding, and B. Xu, "On generality of the data plane and scalability of the control plane in software-defined networking," *China Commun.*, vol. 11, no. 2, pp. 55–64, Feb. 2014, doi: 10.1109/cc.2014.6821737.

[21] S. A. Shah, J. Faiz, M. Farooq, A. Shafi, and S. A. Mehdi, "An architectural evaluation of SDN controllers," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2013, pp. 3504–3508, doi: 10.1109/ICC.2013.6655093.

[22] X.-H. Sun, Y. Chen, and M. Wu, "Scalability of heterogeneous computing," in *Proc. Int. Conf. Parallel Process. (ICPP)*, Washington, DC, USA, 2005, pp. 557–564, doi: 10.1109/ICPP.2005.69.

[23] A. Y. Grama, A. Gupta, and V. Kumar, "Isoefficiency: Measuring the scalability of parallel algorithms and architectures," *IEEE Parallel Distrib. Technol., Syst. Appl.*, vol. 1, no. 3, pp. 12–21, Aug. 1993, doi: 1063-6SS2/93/0800-00123.000.

[24] L. Pastor and J. L. B. Orero, "An efficiency and scalability model for heterogeneous clusters," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2001, pp. 427–434, doi: 10.1109/CLUSTR.2001.960009.

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

[26] X.-H. Sun and D. T. Rover, "Scalability of parallel algorithm-machine combinations," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 6, pp. 599–613, Jun. 1994, doi: 10.1109/71.285606.

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

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

[29] M. Karakusa and A. Durresia, "A survey: Control plane scalability issues and approaches in software-defined networking (SDN)," *Comput. Netw.*, vol. 112, pp. 279–293, 2017.

[30] N. J. Gunther, "Guerrilla capacity planning," in *A Tactical Approach To Planning for Highly Scalable Applications and Services*. New York, NY, USA: Springer, ch. 4, pp. 41–69, doi: 10.1007/978-3-540-31010-5.

[31] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 351–362, Aug. 2010, doi: 10.1145/1851275.1851224.

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

[33] S. H. Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proc. 1st workshop Hot Topics Softw. Defined Netw.* New York, NY, USA: ACM, Aug. 2012, doi: 10.1145/2342441.2342446.

[34] J. C. Mogul and P. Congdon, "Hey, you darned counters!: Get off myasic!" in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.* New York, NY, USA: ACM, 2012, pp. 25–30.

[35] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, "Maple: Simplifying SDN programming using algorithmic policies," in *Proc. ACM SIGCOMM Conf. SIGCOMM* New York, NY, USA: ACM, 2013, pp. 87–98.

[36] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proc. 2nd USENIX Conf. Hot Topics Manage. Internet, Cloud, Enterprise Netw. Services*. Berkeley, CA, USA: USENIX Association, 2012, p. 10.

[37] 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 *Proc. 9th USENIX Conf. Operating Syst. Design Implement.* Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6. [Online]. Available: http://www.icsi.berkeley.edu/pubs/networking/onix10.pdf

[38] Z. Cai, A. L. Cox, and T. S. E. Ng, "Maestro: A system for scalable OpenFlow control," Dept. Comput. Sci., Tech. Rep. TR10-11, 2010. [Online]. Available: https://hdl.handle.net/1911/96391

[39] C. Wang, L. Zhang, Z. Li, and C. Jiang, "Sdcor: Software-defined cognitive routing for the Internet of Vehicles," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 3513–3520, Jul. 2018, doi: 10.1109/JIOT.2018.2812210.

[40] D. Gopi, S. Cheng, and R. Huck, "Comparative analysis of SDN and conventional networks using routing protocols," in *Proc. CITS*, 2017, pp. 108–112, doi: 10.1109/CITS.2017.8035305.

[41] A. Shirmarz and A. Ghaffari, "An adaptive greedy flow routing algorithm for performance improvement in software-defined network," *Int. J. Numer. Model., Electron. Netw., Devices Fields*, vol. 33, no. 1, pp. 1–21, Jan. 2020, doi: 10.1002/jnm.2676.

[42] I. Martín, S. Troia, J. A. Hernández, A. Rodríguez, F. Musumeci, G. Maier, R. Alvizu, and Ó. González de Dios, "Machine learning-based routing and wavelength assignment in software-defined optical networks," *IEEE Trans. Netw. Service Manage.*, vol. 16, no. 3, pp. 871–883, Sep. 2019, doi: 10.1109/TNSM.2019.2927867.

[43] M. Blose and L. A. Akinyemi, "Development of scalable hybrid switching based software defined networking," in *Proc. 23rd Int. Symp. Wireless Pers. Multimedia Commun. (WPMC)*, Tokyo, Japan, Oct. 2020, pp. 1–5, doi: 10.1109/WPMC50192.2020.9309465.

[44] X. Xu, L. Zuo, and Z. Huang, "Reinforcement learning algorithms with function approximation: Recent advances and applications," *Inf. Sci.*, vol. 261, pp. 1–31, Mar. 2014, doi: 10.1016/j.ins.2013.08.037.

[45] R. S. Sutton and A. G. Barto, "Reinforcement learning," in *An Introduction*, 2nd ed. pp. 459–464, 2018, Accessed: Jan. 22, 2023. [Online]. Available: https://dokumen.pub/reinforcement-learning-an-introduction-2nbsped-0262039249-9780262039246.html

[46] M. Blose, L. A. Akinyemi, F. Nicolls, and N. Ventura, "Development of scalable hybrid switching based SDN using reinforcement learning," in *Proc. Southern Afr. Telecommun. Netw. Appl. Conf. (SATNAC)*, KwaZulu-Natal, South Africa, vol. 29, Aug. 2023, pp. 163–168. [Online]. Available: https://www.satnac.org.za/proceedings

[47] (2023). *Exploration vs. Exploitation—Learning the Optimal Reinforcement Learning Policy*. Accessed: May 12, 2023. [Online]. Available: https://deeplizard.com/learn/video/eMxOGwbdqKY

[48] *Deep Learning Playlist Overview & Machine Learning Intro*. Accessed: May 16, 2023. [Online]. Available: https://deeplizard.com/learn/video/rP4oEpQbDm4

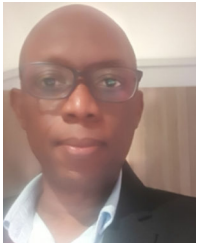
[49] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction—Dynamic Programming Methods*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018, ch. 4, pp. 73–88, Accessed: Jan. 22, 2023. [Online]. Available: https://dokumen.pub/reinforcement-learning-an-introduction-2nbsped-0262039249-9780262039246.html

[50] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction—T Temporal-Difference Learning*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018, ch. 6, pp. 119–138. https://dokumen.pub/reinforcement-learning-an-introduction-2nbsped-0262039249-9780262039246.html [Accessed 22, Jan. 2023]

[51] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction—Bootstraping*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018, ch. 7, pp. 141–157, Accessed: Jan. 22, 2023. [Online]. Available: https://dokumen.pub/reinforcement-learning-an-introduction-2nbsped-0262039249-9780262039246.html

[52] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998, Accessed: Mar. 3, 2023. [Online]. Available: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

[53] *Temporal Difference Learning in Reinforcement Learning*. Accessed: Apr. 24, 2023. [Online]. Available: https://medium.com/nerd-for-tech/temporal-difference-learning-in-reinforcement-learning-cf13ed159fcb
[54] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, May 1992, doi: 10.1007/bf00992698.
[55] *Reinforcement Learning-Developing Intelligent Agents*. Accessed: 16, 2023. [Online]. Available: https://deeplizard.com/learn/video/qhRNvCVVJaA
[56] *CloudSim: A Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services*. Accessed: Dec. 21, 2022. [Online]. Available: https://github.com/Cloudslab/cloudsim



MAX BLOSE received the B.Eng. degree in electrical and electronic engineering from Tshwane University of Technology, Pretoria, South Africa. He is currently pursuing the M.Eng. degree in electrical engineering with the University of Cape Town, South Africa. He is a System Integrator with Vodacom. He was an IP Network Engineer with Telkom. He was a VSAT Engineer with Telkom. His research interests include machine learning, SDN, IP core networks, data analytics, and wireless communications.

LATEEF ADESOLA AKINYEMI (Senior Member, IEEE) received the B.Sc. degree (Hons.) in electronic and computer engineering (computational electronics) and the M.Sc. degree in electronic and computer from Lagos State University, Lagos, Nigeria, the M.Sc. degree in electrical and electronics engineering (communication engineering option) from the University of Lagos, Akoka, Nigeria, and the Ph.D. degree in electrical engineering from the Department of Electrical Engineering, Faculty of Engineering and the Built Environment, University of Cape Town, Western Cape, South Africa. He is currently a Lecturer, a Researcher, and a Scholar with the Department of Electronic and Computer Engineering, Faculty of Engineering, Lagos State University, Epe Campus, Lagos. He is also a Postdoctoral Research Fellow with the College of Science, Engineering and Technology, the School of Engineering, the School of Computing, the Department of Computer Science, and the Department of Electrical Engineering, Florida Campus, University of South Africa, Johannesburg, South Africa, and the Centre for Augmented Intelligence and Data Science (CAIDS). His research interests include wireless communications, computational electronics, modeling and simulations of quantum-inspired nano-particles and devices, microwave engineering and antennas, data science, analytics, artificial intelligence-inspired algorithms, computational intelligence, machine learning, eHealth and eMobile, and cybersecurity and its application.

STEPHEN OJO (Member, IEEE) received the B.Sc. degree (Hons.) in electrical and electronics engineering from the Federal University of Technology Akure, Nigeria, in 2014, and the M.Sc. and Ph.D. degrees in information systems from Girne American University, Cyprus, in 2017 and 2021, respectively. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, College of Engineering, Anderson University, Anderson, SC, USA. Before joining Anderson University, he was a Lecturer with Girne American University, where he taught courses in distributed computing, advanced programming, and electric circuits. He was also a Research Scholar with Vodafone Telecommunication Company, Cyprus, where he developed a multiplicative-based model for signal propagation in wireless networks. He was awarded the Mobil and Full Ph.D. scholarships throughout his undergraduate program. He is also a full-time Faculty Member with Anderson University, where he teaches computer programming, electric circuits, machine learning, artificial intelligence in wireless mobile networks, and biomedical applications. He has authored or coauthored several peer-reviewed journals. His research interests include wireless networks, machine learning for wireless mobile networks, machine learning, and AI in biomedical devices.

MUHAMMAD FAHEEM (Member, IEEE) received the B.Sc. degree in computer engineering from the University College of Engineering and Technology, Bahauddin Zakariya University (BZU), Multan, Pakistan, in 2010, the M.S. degree in computer science from Universiti Teknologi Malaysia (UTM), Johor Bahru, Malaysia, in 2012, and the Ph.D. degree in computer science from the Faculty of Engineering, UTM, in 2021. From 2012 to 2014, he was a Lecturer with the COMSATS Institute of Information and Technology, Pakistan. He was an Assistant Professor with the Department of Computer Engineering, Abdullah Gul University (AGU), Kayseri, Turkey, from 2014 to 2022. He is currently a Researcher with the School of Computing Science (Innovations and Technology), University of Vaasa, Vaasa, Finland. He has numerous publications in journals and international conferences. His research interests include the areas of cybersecurity, Industry 4.0, smart cities, smart grids, and underwater sensor networks.

AGBOTINAME LUCKY IMOIZE (Senior Member, IEEE) received the B.Eng. degree (Hons.) in electrical and electronics engineering from Ambrose Alli University, Nigeria, and the M.Sc. degree in electrical and electronics engineering from the University of Lagos, Nigeria. He is currently a Lecturer with the Department of Electrical and Electronics Engineering, University of Lagos. He is also a Research Scholar with Ruhr University Bochum, Germany, under the Nigerian Petroleum Technology Development Fund (PTDF) and the German Academic Exchange Service (DAAD) through the Nigerian-German Postgraduate Program. Before this, he was a Lecturer with the Bells University of Technology, Nigeria. He was awarded the Fulbright Fellowship as a Visiting Research Scholar with the Wireless@VT Laboratory, Bradley Department of Electrical and Computer Engineering, Virginia Tech, USA, from 2017 to 2018. He was also the Core Network Products Manager with ZTE Corporation and a Network Switching Subsystem Engineer with Globacom, Nigeria. His research interests include 6G wireless communication, wireless security systems, and artificial intelligence. He is the Vice Chair of the IEEE Communication Society Nigeria Chapter.

ARFAT AHMAD KHAN received the B.Eng. degree in electrical engineering from the University of Lahore, Pakistan, in 2013, the M.Eng. degree in electrical engineering from Government College University Lahore, Pakistan, in 2015, and the Ph.D. degree in telecommunication and computer engineering from the Suranaree University of Technology, Thailand, in 2018. From 2014 to 2016, he was an RF Engineer with Etisalat, United Arab Emirates. From 2018 to 2022, he was a Lecturer and a Senior Researcher with the Suranaree University of Technology. He is currently a Senior Lecturer and a Researcher with Khon Kaen University, Thailand. His research interests include optimization and stochastic processes, channel and mathematical modeling, wireless sensor networks, ZigBee, green communications, massive MIMO, OFDM, wireless technologies, signal processing, and advanced wireless communications.