applied sciences

MDPI

Article

Performance and Scalability Analysis of SDN-Based Large-Scale Wi-Fi Networks

Mohsin Ali [1], Ali Imran Jehangiri [1], Omar Imhemed Alramli [2], Zulfiqar Ahmad [1,*], Rania M. Ghoniem [3,*], Mohammed Alaa Ala'anzy [4] and Romana Saleem [1]

[1] Department of Computer Science and Information Technology, Hazara University, Mansehra 21300, Pakistan
[2] Department of Networks and Telecommunications, Faculty of Information Technology, Misurata University, Misurata P.O. Box 2478, Libya
[3] Department of Information Technology, College of Computer and Information Sciences, Princess Nourah bint Abdulrahman University, P.O. Box 84428, Riyadh 11671, Saudi Arabia
[4] Department of Communication Technology and Networks, Universiti Putra Malaysia (UPM), Serdang 43400, Malaysia
* Correspondence: zulfiqarahmad@hu.edu.pk (Z.A.); rmghoniem@pnu.edu.sa (R.M.G.)

Abstract: The Software-Defined Networking (SDN) paradigm is one that is utilized frequently in data centers. Software-Defined Wireless Networking, often known as SDWN, refers to an environment in which concepts from SDN are implemented in wireless networks. The SDWN is struggling with challenges of scalability and performance as a result of the growing number of wireless networks in its coverage area. It is thought that SDN techniques, such as Mininet-Wi-Fi and Ryu Controller for wireless networks, can overcome the problems with scalability and performance. Existing Wi-Fi systems do not provide SDN execution to end clients, which is one reason why the capability of Wi-Fi is restricted on SDN architecture. Within the scope of this study, we analyzed Wi-Fi networks operating on SDN using the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). By utilizing a testbed consisting of Ryu Controller and Mininet-Wi-Fi, we were able to test Wi-Fi over SDN and evaluate its performance in addition to its scalability. When evaluating the performance of a network, we take into account a number of different metrics, such as bandwidth, round-trip time, and jitter. In order to assess the level of performance, the SDN-based Wi-Fi controller Ryu is linked to an increasing number of access points (1, 2, 3, and 4) as well as stations (10, 30, 50, and 100). The experimental findings obtained using Mininet-Wi-Fi indicate the scalability and dependability of the network performance provided by the SDN Wi-Fi network controller Ryu in an SDN environment. In addition, the round-trip time for TCP packets grows proportionally with the number of hops involved. A single access point is capable of simultaneously supporting up to fifty people at once.

Keywords: SDN; routing; Wi-Fi; controller; wireless network

Citation: Ali, M.; Jehangiri, A.I.; Alramli, O.I.; Ahmad, Z.; Ghoniem, R.M.; Ala'anzy, M.A.; Saleem, R. Performance and Scalability Analysis of SDN-Based Large-Scale Wi-Fi Networks. *Appl. Sci.* **2023**, *13*, 4170. https://doi.org/10.3390/app13074170

Academic Editors: Ireneusz Kubiak, Tadeusz Wieckowski and Yevhen Yashchyshyn

Received: 3 March 2023
Revised: 20 March 2023
Accepted: 21 March 2023
Published: 24 March 2023

1. Introduction

Software-Defined Networking (SDN) is a significant networking technology that uses programmable interfaces to smoothly link application provisioning in the cloud with the network [1]. This revolution suggests indicators or perhaps different web-based applications. Managers who use massive volumes of data, such as Google and Facebook, are switching to SDN for their fundamental systems [2–4]. In consideration of layered engineering, the Open Systems Interconnection (OSI) paradigm was implemented in the field of communication. This model approved level-independent advancements and established prohibitive linked issues in heterogeneous frameworks. The major objective of SDN is to revive this theory-based approach to build the organization necessary to contain new settlements and boost creative practices at the same time [5–7].

SDN is a new paradigm that needs significant advancements in everything from system administration to communication networks. Although there are many different

needs for the SDN market, it is apparent that this innovation is an appealing alternative to traditional systems administration and continues to offer useful solutions for inquiry. Currently, wireless technology is taking over the whole world. Currently implemented scenarios predict an increase in demand for SDN-based Wi-Fi. SDN has less support than wireless technologies like Wi-Fi, Wi-MAX 802.16, WLAN 802.11, and Bluetooth 802.15 [5]. We searched through the research literature but were unable to find any significant implementation of these technologies in SDN [6–9]. This research project has produced methods that will deploy Wi-Fi on SDN. Limitations on current applications include the following: On the SDN architecture, there are not many Wi-Fi features accessible. The installation of SDN for end users is not supported by current Wi-Fi networks [7]. Users may construct, configure, alter, and troubleshoot using SDN, which effectively enables programmatic network administration. Thus, it is a significant idea that will shape networks in the future [10]. The static network designs, or classical architectures, are more difficult than the new ones that are developed programmatically, are overcome by the notion of SDN, making them more flexible and simple to alter and debug [11]. The work already done on traditional networks cannot adequately address a number of new challenges that the SDN paradigm introduces.

Scalability: In SDN, the issue of complexity and scalability has been brought up. In large data centers, the demand for bandwidth reaches its peak during peak hours. Networks are unable to handle heavy data loads in high-speed traffic flows. Vendors have created their own architectures and specifications for networking devices [6]. Scalability issues can be resolved by laboriously standardizing SDN networking hardware.

Placement of the controller: This is also known as the network operating system, and it is an important aspect for research and discussion in SDN. Whether the position is fixed or dynamic, it needs to be adjusted to the network environment. In controller layout [12], a number of layout indicators are discussed along with their significance in relation to the surrounding network layout [13,14].

Resilience: The centralized controller architecture comes with a number of built-in restrictions. The main problem with a centralized controller is the single point of failure. Data throughput across the entire network is more efficient thanks to the centralized design [15]. People might be more open to giving up greater flexibility for greater efficiency. Research on the SDN distributed controller architecture demonstrates that it can offer more flexibility.

The abovementioned challenges are addressed through this research work by providing the performance and scalability analysis of SDN-based large-scale Wi-Fi networks. In our proposed work, the Wi-Fi-based scenarios are generated and tested using the network protocols (TCP and UDP) for their stability in various dynamic environments.

The main contributions of this work are listed below:

- We analyzed the performance of the SDN Wi-Fi network using simple, linear, and tree topologies.
- For the purpose of performance analysis, we made use of a large-scale SDN-based Wi-Fi network.
- We were able to observe the scalability of the SDN Wi-Fi network in addition to its resilience by taking into consideration a variety of dynamic situations of the network.
- We explored the performance metrics of the SDN Wi-Fi controller via TCP and UDP protocols, such as bandwidth, flow setup delay, jitter, and round-trip time.

The remaining part of this paper is organized as follows: Section 2 delivers the related work. Section 3 highlights the background knowledge. Section 4 gives the details of the proposed SDN-based architecture. Section 5 provides experiments, results, and discussion. Finally, Section 6 concludes the work.

2. Related Work

SDN controllers, wired networks, Wi-Fi networks, and simulation toolkits used for SDN Wi-Fi networks are all aspects of related study that are being investigated here. On

the topic of performance evaluation of wired SDN networks, there are a good number of research articles available:

The effectiveness of floodlight controllers and POX controllers was analyzed and contrasted by Bholebawa et al. [16]. The authors used Mininet to imitate a variety of topologies and then measured the round-trip time as well as the bandwidth. They came to the conclusion that Floodlight was doing significantly better than POX. POX is recommended in addition to Floodlight because of its usability, notwithstanding the fact that Floodlight achieves good performance.

Performance of "Ryu", "POX", and "Pyretic" SDN controllers was analyzed and evaluated by Shamim et al. [17]. Mininet was the instrument the creators used to recreate a wired SDN. The round-trip time was the most important performance parameter. Pyretic performs better than Ryu and POX, according to the findings of this research report, which compares the three different controllers.

Fancy et al. [18] advise using Floodlight and POX to compare the two systems' performances. They take into consideration measures such as throughput and delay. Experiments on the controllers have been conducted in a variety of geographical settings. The authors reach the conclusion that Floodlight is effective than POX in terms of its performance. The execution of Floodlight, on the other hand, requires a predetermined quantity of memory space. Because POX is dependent on Python, it is the superior option in this scenario.

In [19], the authors contrast the performance of the SDN Controllers "Beacon, MuL, Mestro, POX, NOX, Floodlight and Ryu" by making use of the characteristics such as throughput, reliability, safety, and other similar factors. According to the findings of the study, Beacon, NOX, Floodlight, POX, and Ryu are all effective under typical traffic conditions. On the other hand, MuL and Maestro do not appear to be efficient under the same conditions.

Rastogi et al. [20] examine the similarities and differences between the python-based SDN controllers POX and RYU. Emulation of networks and the production of traffic were both possible with Mininet's use. The authors concluded that POX offers superior performance when it comes to layer 1 switching. On the other hand, it appears that RYU is superior to POX when it comes to layer 2 exchanging.

There have only been a few studies that have evaluated the performance of "Software defined Wireless networks".

"Ryu, POX, ONOS, and Floodlight" were tested to see how well they performed in an emulated wireless network by Islam S. and his colleagues [11]. They evaluated each of these based on jitter and throughput to determine which had the superior performance. It seems that Floodlight has a rather low amount of jitter. The throughput of SDN controllers, on the other hand, does not vary greatly from one to the next.

The authors of the paper [21] demonstrated the scalability of an SDN-based Wi-Fi Network running on Mininet by analyzing performance characteristics in a variety of different dynamic scenarios. The authors concluded that increasing the number of hosts for the TCP protocol results in a drop in bandwidth and performance, whereas increasing the number of hosts for the UDP protocol has the opposite effect: jitter increases, yet the bandwidth almost entirely maintains its previous level. In wired networks, it would appear that SDN controllers have been thoroughly analyzed in terms of performance. On the other hand, SDWN are a relatively new topic, and large-scale SDN Wi-Fi networks require performance evaluation using a variety of SDN controllers. This can be inferred from an examination of the state of the art.

The study in [22] proposes an algorithm for efficiently and intelligently changing packet directions in SDN networks. The proposed model estimates path costs based on five criteria: adaptive network packet size, accurate packet numbers, the overall required time interval, QoS link capacity (bandwidth), and the number of hops (shortest path), enabling the SDN controller to minimize decision time for selecting flows. The proposed algorithm is evaluated with a dataset containing information about routing delay. The model incorporates three criteria, namely packet size, number, and time, to determine the

optimal packet delay and subsequently calculate the cost of each path. Results from a benchmark comparison between the proposed algorithm and state-of-the-art alternatives show a significant reduction in delay time, estimated to be a few milliseconds, for selecting an optimal recovery path. Consequently, the proposed algorithm can reduce bottleneck routes and resource utilization, leading to increased QoE (Quality of Experience) for both objective and subjective video streaming. Specifically, the proposed model reduces delay time for route selection up to 96.3%, resulting in greater end-user satisfaction.

A software component of SDN known as the SDN controller has the responsibility for managing the flow control of a network. In SDN, the controller is the primary entity that stands among the applications and the networking devices. Each and every communication that takes place between apps and network devices goes through the SDN controller. There are number of SDN controllers; some well-known SDN controllers [11,17,19] are shown in Table 1.

Table 1. Comparison of SDN controllers.

S.No	Controllers	Platform Support	Programing Languages	Modularity	Centralized/ Distributed	Partner	WI-FI Supported	Open Switch Support	Documentation
1	Floodlight [11,17,19]	Linux	Java	Fair	Centralized	Big Switch Network	No	Yes	Good
2	Ryu [11,17,19]	Linux, Windows and MAC OS	Python	Fair	Centralized	Nippo Telegrah	Yes	Yes	Good
3	Beacon [11,17,19]	Linux, Windows and MAC OS	Java	Fair	Centralized	Stanford University	No	No	Fair
4	POX [11,17,19]	Linux, Windows and MAC OS	Python	Low	Centralized	Nicira	No	No	Poor
5	ONOS [11,17,19]	Linux	Java	High	Distributed	Cisco	No	Yes	Good

The vast majority of the SDN studies that have been proposed in published papers are small-scale Wi-Fi networks that use SDN. However, due to the growing adoption of Wi-Fi networks and the significance of these networks for the Internet of Things, SDN must be integrated with Wi-Fi networks. Integration of Wi-Fi networks with SDN controllers is thought to be able to overcome a significant number of performance difficulties that are associated with Wi-Fi networks. For this reason, a performance analysis of an actual large-scale SDN Wi-Fi network is necessary for upcoming Internet of things networks [10].

SDN was initially created for wired networks; however, wireless networks can also benefit from its principles. In order to manage and control the access points and switches that make up a WiFi network, many SDN controllers, including Ryu, support Wi-Fi networks. There are several well-known SDN controllers with varying degrees of Wi-Fi network support, including Ryu, Floodlight, Beacon, POX, and ONOS. Ryu has a solid track record of supporting Wi-Fi networks and offering a wide range of protocol support, including OpenFlow and Netconf, which are frequently used in Wi-Fi networks. Floodlight is suitable for managing Wi-Fi networks because it, like Ryu, supports OpenFlow and Netconf. However, Floodlight's focus is more on large-scale data center networks, which may not be the primary use case for Wi-Fi networks. On the other hand, Beacon is a research-focused SDN controller that is not frequently used in real-world environments. Although it manages Wi-Fi networks and supports OpenFlow, it might not offer Ryu's level of performance, scalability, and reliability. Similar to this, POX is a compact SDN controller that is frequently employed in academic and research settings. While it can be used to manage Wi-Fi networks, Ryu's more powerful controllers have more functionality and scalability [23–25].

As a result, in order to solve the problems that were discussed before, we offer an analysis of the performance and scalability of SDN-based large-scale Wi-Fi networks. In the work that we have presented, the Wi-Fi-aware scenario is built, and then the TCP and UDP protocols are used to test it so that we can evaluate the stability of the scenario in various dynamic settings. The findings are produced and then graphically assessed by building a comparison model of the existing scenarios with the present ones in order to locate a forward-looking approach.

3. Background Knowledge

SDWN is the application of SDN concepts implemented in wireless networks. SDN promotes network control over changeable data routes [8]. Due to the fact that forwarding hardware and control logic are kept in separate locations, it is easier to build new software-controlled protocols, application programs, and administration systems for networks [6].

3.1. Software-Defined Networking

A new technology called SDN uses programmable interfaces to seamlessly integrate network and cloud application provisioning [7]. In light of layered engineering, the Open Systems Interconnection (OSI) concept was applied to communication. This model approved level-independent advancements and established prohibitive linked problems in heterogeneous frameworks. The major objective of SDN is to revive this theory-based approach to create the organization necessary to contain new settlements and boost creative practices at the same time [5,26,27]. SDN is a revolution that suggests numerous web-based applications or perhaps only indications. Managers who use massive volumes of data, such as Google and Facebook, are switching to SDN for their fundamental systems. Similarly, large ISPs see SDN as a suitable response to their current communications [6].

Network communication management and design have been completely transformed by the SDN network architecture. Three layers make up the SDN architecture, and each one serves a unique purpose. The network applications that deliver services to end users are housed in the application layer, which is the first layer. Firewalls, intrusion detection systems, and load balancing are a few examples of these applications. The control layer, which is in charge of managing the network's resources and setting up network gadgets, is the second layer. It has a central controller that uses protocols like OpenFlow to talk to network devices. The controller gathers information about network traffic from the data plane and decides how to send it in accordance with the policy guidelines set forth by the applications running at the application layer. The data layer, which is the third layer overall, is in charge of forwarding network traffic. It is made up of network components such as switches and routers that the controller in the control plane uses to send forwarding instructions. SDN offers a network design that is more adaptable, scalable, and manageable by distinguishing the data place from the control plane. It allows network administrators to quickly react to shifting business requirements and take immediate action in response to security risks [28,29].

3.2. SDN Controllers

The controller is a key component in SDN that links network devices together [30]. Several well-known SDN controllers are given below:

3.2.1. ONOS Controller

The ONOS controller is an open-source controller that is often applied in a variety of research and testing settings. "Open Network Operating System" (ONOS) is a distributed network controller that uses the Replicated Agreement for a Fault-Tolerant (RAFT) scheme for switching and mapping to its key controller. This scheme is stored in ONOS's central database. In order to ensure the consistency throughout its cluster, the leader node is the one that starts the process of checking for updates and delivering those updates to the

nodes that are closer to it. In order to keep the scalability of ONOS intact during data transmission, the RAFT algorithm is executed [7,31,32].

A web interface for the ONOS controller is available online. Using a web-based graphical user interface, the user can quickly manipulate the architecture of the network. It stores information on the network architecture, roles (slave or master), switches, links, hosts, flows, and pathways in a network that was built using the ONOS controller. The representation of the network topology displays information about the hosts, as well as linkages between the switches, data flow channels, controllers, and the capacity of the links. The web-based graphical user interface is helpful for performing analysis on SDN networks [10,33,34].

3.2.2. NOX and POX Controller

The term "Network Operating Systems" refers to the initial OpenFlow controller (NOX). It performs the duties of a system control stage and provides an abnormal state automatic interface for the administration and improvement of system control applications. The progression of the issue and its classification as a product problem are both illustrated by its structure. NOX is incorporated as a biological component into the SDN technology [12,35].

Python-based POX is an open-source development platform for "Software-defined networking" (SDN) which implements an application development platform, including OpenFlow SDN controllers. This platform was developed using the Software Development Network (SDN). The OpenFlow protocol is the standard for industry for controller-to-switch communication, and easily implemented because of the controller's user-friendly interface. The POX controller has the capability of running a variety of applications, including a load balancer, center point, switch, and firewall [12,36].

3.2.3. Floodlight Controller

The Big Switch Network developed Floodlight, an open-source controller based on the Java programming language. An OpenFlow-enabled switch is utilized by the floodlight controller. It is permissible for use in a wide variety of contexts because it is distributed with an Apache license [7]. Floodlight is designed on a novel architecture that not only allows for the management of system network topologies, devices, path finding, and state storage, but also supports these functions. The Floodlight controller is quite popular and comes highly recommended by industry professionals [10,31,37].

3.2.4. Ryu Controller

Python, an open-source language, is used to build Ryu. The part of the SDN structure that helps with memory states, matching, and administration of OpenFlow is what the Ryu depends on. A reliant component of the SDN architecture, the Ryu Controller aids in OpenFlow conversion and memory status, communication, and administration [7,37].

This controller uses a segmented SDN architecture that helps with OpenFlow conversion, memory states, communication, and management [7]. Its features include monitoring OpenFlow operations via a Virtual Local Area Network that uses State Transfer (REST), unexpected accessibility, and even the ability to control a variety of functions.

3.3. OpenFlow

The "Open Network Foundation" developed the communication protocol known as "OpenFlow". It grants access to a network's switches and routers' data planes. The southbound API, known as OpenFlow, is in charge of enabling communication between the controller and other network devices. The OpenFlow controller, which regulates network switch behavior, makes use of the OpenFlow protocol. Ports, OpenFlow channels, and OpenFlow tables make up OpenFlow switches. Through an OpenFlow channel, the link between the controller and switch is made [38]. Figure 1 illustrates the differences between an SDN switch and a conventional switch.

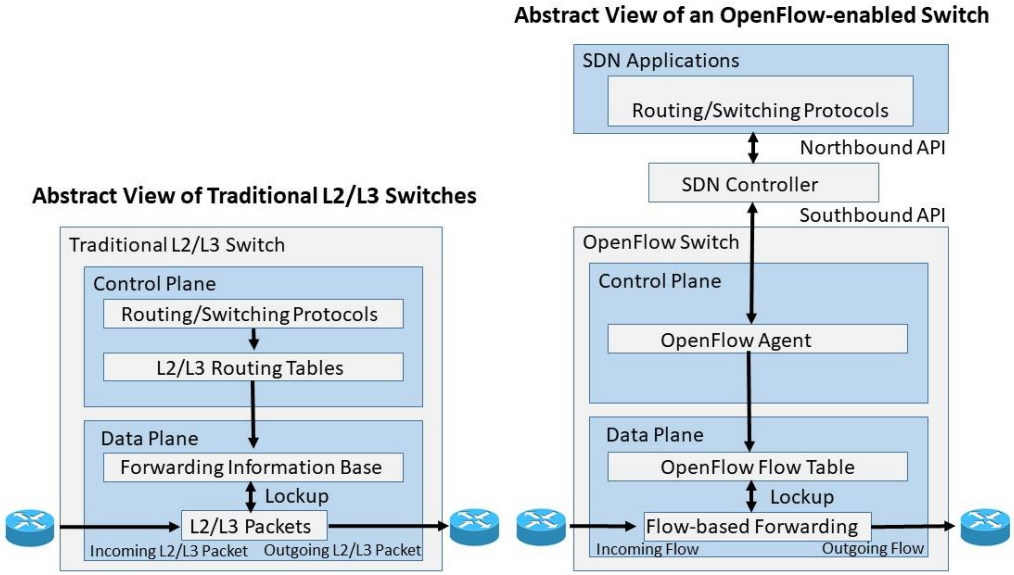

Figure 1. The differences between an SDN switch and a traditional switch [17,20,39].

According to the OSI model, the data connection layer is in charge of the operation of the switch. In an environment based on SDN, the job of switches is to communicate data from one place to another. Within a network, switches have the ability to communicate data to a large number of other network devices. A switch has a database with a name-forwarding table that stores the MAC addresses of other network devices. This table is accessible through the switch's web interface. The "Media Access Control" (MAC) address of a device is equivalent to its hardware address [7].

OpenFlow is a protocol that is implemented by SDN controllers so that they can connect with network nodes. The process of forwarding data through an OpenFlow switch is determined by "flows", which are sequences of packets that conform to a set of predefined rules [10].

3.4. Mininet

Mininet is an open-source tool for simulating networks [10,40]. It is possible to establish a genuine virtual network with just a single command, complete with a true kernel, switch, and application code. It is compatible with the OpenFlow protocol, which provides a means of communication between the data forwarding plane and the control plane. It offers support for a wide range of topologies, which enables users to experiment with a variety of various SDN scenarios. In addition to its many other benefits, Mininet cannot be enrolled as a direct replacement for Wi-Fi. There is also a graphical user interface for Mininet that goes by the name of Miniedit [10].

3.5. Mininet-Wi-Fi

The Mininet network emulator expanded its capabilities by adding support for virtual Wi-Fi stations. The "standard Linux wireless driver" and the 80211 hwsim wireless emulation driver are the foundations upon which Mininet's connection points are constructed [11]. The purpose of Mininet-Wi-Fi is to advance research on both wireless and SDWN by offering high-fidelity simulations of wireless networks that have been tested in real networks within an environment that is completely under control [5]. An experimental platform known as Mininet-Wi-Fi gives its customers the ability to build a network at any time using only a computer and no other hardware [15]. This presents a significant convenience for the users of the platform. Additionally, real software and network protocol codes can be executed by Mininet-Wi-Fi so that a variety of protocols can be tested and the performance of devices and signals can be analyzed [21]. The Mininet-Wi-Fi is unique in that it gives devices in a particular location the ability to communicate with one another

wirelessly. Users are able to test and measure node mobility and signal propagation thanks to the availability of a number of different types of radio signal propagation [15,41].

4. SDN Based Architecture

This section introduces the use case scenario, the experimental testbed, the topologies that were investigated, and the various dynamic use cases.

4.1. Use Case Scenario

In this section, we took a real-world scenario into consideration by analyzing the smart network facility at Hazara University. The following are the configurations of the smart network at Hazara University.

There is one router, two layer 2 switches, twenty-four layer 3 switches, forty-six access points, and 470 MB of bandwidth available on this network. The number of registered users of Smart Net exceeds 6000. However, on average, only 1200 users are connected to the network at any given time, with the maximum number of users connected to the network at any given time being up to 4500.

The use of SDN-based architecture and the evaluation of performance using TCP and UDP protocols are key aspects of the proposed approach for monitoring and evaluating the scalability and resilience of a wireless network. Figure 2 illustrates the suggested architecture, which utilizes SDN to control the network traffic and optimize the routing of packets. This architecture can be used to handle a variety of dynamic scenarios and improve the performance of the network. TCP and UDP are the two most commonly used protocols for data transmission over the internet. TCP is a connection-oriented protocol that provides reliable delivery of data by verifying that all packets have been received in the correct order. On the other hand, UDP is a connectionless protocol that does not provide reliability but is faster and more efficient for applications that do not require reliability. By utilizing these protocols in the performance evaluation of the proposed SDN wireless network, it is possible to measure the efficiency of the network in terms of packet delivery and throughput. This information can then be used to identify potential bottlenecks and optimize the network for better performance. Monitoring the network over SDN allows for greater visibility into the network and the ability to dynamically control the flow of traffic. We are able to monitor Wi-Fi over SDN and notice its scalability in addition to its resilience if we evaluate the output parameters of a variety of different dynamic scenarios [11].

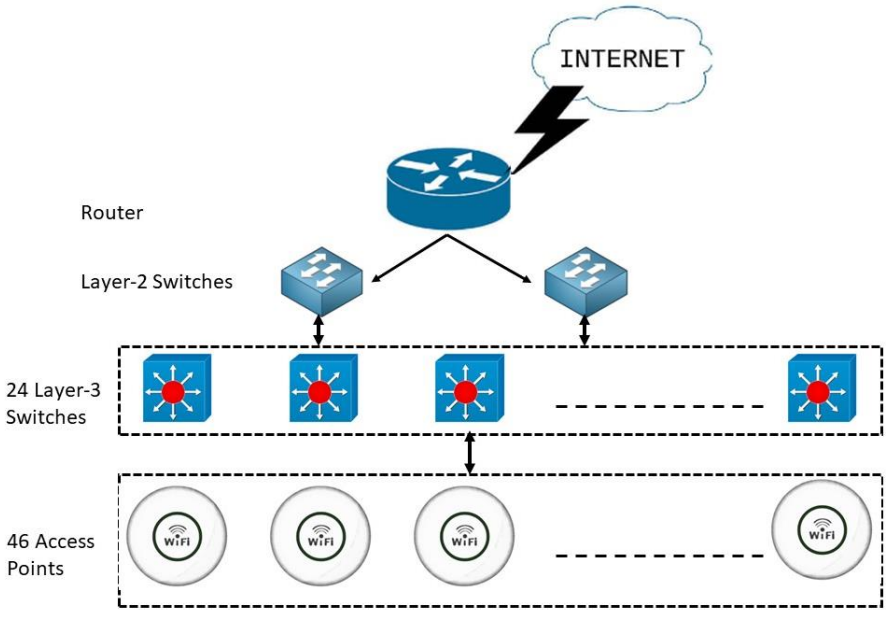

Figure 2. SDN-based architecture.

4.2. Experimental Testbed

The testing environment comprised an Intel Xeon server with a CPU running at 1.70 GHz and 128 gigabytes of Random Access Memory (RAM). The system makes use of Kernel-based Virtual Machine (KVM) virtualization in order to run the Mininet-Wi-Fi VM that was provided by the developers of Mininet-Wi-Fi. Ubuntu 20.04 is the guest operating system, and there are 32 gigabytes of RAM available. The host machine is powered by Ubuntu 18.04 and has 128 gigabytes of Random Access Memory [5].

In this project, we used three different topologies, the linear, the tree, and the Custom topologies, to carry out experiments. The link bandwidth that was utilized for these tests was 370 megabyte per second (mb/s), and the maximum window size (socket buffer) that was available was 32 mb/s. In each of the experiments, an equal number of switches and hosts (1 switch and 10 stations per switch) was utilized [10].

The remote SDN controller was Ryu (version 4.14), which was installed. The iperf tool was installed for the purpose of performance measurement [42]. The tools iperf and ping were used to collect the data that were used for the performance evaluation metrics, such as flow bandwidth, jitter, and flow setup latency [11]. The Ryu controller was chosen because it is a highly flexible, customizable, and reliable SDN controller that supports a wide range of protocols and runs very well. As compared to other SDN controllers, Ryu allows network administrators to easily develop and implement customized network management applications. It provides a comprehensive set of "application programming interfaces" (APIs), libraries, and plug-ins that make it easy to add new features and functions to the platform. Ryu is modular and scalable, which means it can easily adapt to changes in the network topology and handle increased network traffic. It has extensive protocol support, which includes OpenFlow, Netconf, and OF-config, among others. This makes it compatible with a wide range of network devices, including switches, routers, and firewalls, and allows administrators to manage their network devices in a consistent and efficient manner. Ryu also provides excellent performance and reliability. Its lightweight architecture and efficient packet processing capabilities ensure that it can handle large volumes of network traffic without compromising on performance [11].

In SDN-based architecture, layer 2 and layer 3 switches have various capabilities and roles. In order to forward frames within a LAN, a layer 2 switch operates at the Data Link Layer of the OSI model. These switches can only forward traffic based on MAC addresses and are often used to connect devices within the same LAN segment. With SDN controllers, layer 2 switches can be configured to dynamically update forwarding tables based on network traffic. The Network Layer of the OSI model is where a layer 3 switch operates. Here, it employs IP addresses to forward packets across LANs and subnets. These switches are capable of performing routing tasks, choosing which packets to forward based on their IP addresses, and supporting routing protocols. With SDN controllers, layer 3 switches can be instructed to generate and update routing tables.

4.3. Performance Parameters

A Mininet-based SDN Network experiment was run for TCP and UDP flows in order to analyze the behavior of Wi-Fi. Bandwidth, jitter, and flow setup delay are factors that are taken into account while observing TCP and UDP flows behavior in a wireless environment. The installation of flow rules at switches and the creation of network event notifications from the controller side are not instantaneous tasks in a typical switch–controller cycle. A large-scale network's performance is severely impacted by flow setup delays. Low flow setup latencies are therefore needed for large-scale SDN network performance.

4.3.1. Bandwidth Utilization

Connection-oriented TCP takes the same route from source to destination for every flow, increasing the possibility of one link being overloaded and bottlenecked, which may be demonstrated by carefully monitoring bandwidth. The possibilities of a congested path are minimal with connectionless UDP because packets are spread over numerous pathways,

which allows them to reach their destinations. Here, the amount of time it takes for the flow to entirely reach its destination, where jitter is a significant factor, is what matters. Therefore, bandwidth and jitter are the best characteristics to watch for UDP traffic.

The iperf tool has been utilized in order to perform measurements on the data flow in both directions. Iperf also provides information regarding the utilization of available bandwidth in terms of the speed at which data is sent between the client and the server. In each and every one of the trials, the Mininet-Wi-Fi default TCP window size of 85.3 kb was utilized [43].

4.3.2. Flow Setup Latency

In a typical switch-controller cycle, creating a network event notification on the controller side and installing flow rules at switches are not instantaneous activities. Rather, both of these activities take place over the course of the cycle. The performance of large-scale networks is negatively impacted in significant ways by the flow setup latencies. Therefore, in order to maximize the performance of large-scale SDN networks, low flow setup latencies are desired.

The Round-Trip Transmission Time, abbreviated as RTT, was factored into the calculation of the flow setup latency. The first pings in SDN always take longer than the ones that come after them because they have to make route inquiries to the controller in the beginning. As a consequence of this, the RTT of the first packet has been investigated in order to acquire a sufficient comprehension of flow setup latency. The RTT of each execution of the following command is added together and used in the calculation of these [43].

Mininet-Wi-Fi > sta1 ping -c1 sta2

The process of setting up new user-initiated flows is made much easier by our solution, which dramatically minimizes the number of switches that must be altered to accommodate the new flow. On our testbed, we implemented our concept and monitored its performance using various metrics. According to the findings, it makes a significant contribution to performance enhancement, especially in the tail, by reducing the median and 99-percentile latencies to 5.9 and 7 milliseconds, respectively, under conditions that are identical to those previously tested.

4.3.3. Jitter

Jitter refers to the difference in the amount of time it takes for two hosts to respond. Jitter was computed by using the tools for testing UDP performance that are provided by iperf. For instance, the following command will check the jitter between stations sta1 and sta6. UDP packets were delivered for a period of thirty seconds, and a jitter report was generated at one second intervals throughout that time. The evaluation used a plot with a duration of thirty seconds [43].

Mininet-Wi-Fi > sta1 iperf -s h16 -u -i 1

sta6 iperf -c sta1 -u -d

When using a large-scale Wi-Fi network, computing jitter shows the effect it has on the performance of the network when the size of the network has been increased.

4.4. Programming the Topology

We needed to create different topologies in Mininet-Wi-Fi in order to answer the following research questions of the proposed study.

RQ#1: What effect does an increase in the number of switches and access points have on the bandwidth, latency, and scalability of the SDN controller performance?

RQ#2: How do three topologies affect the SDN Wi-Fi network's flow setup latency?

RQ#3: What effect does the scalability of switches have on the latency of flow setup?
Several Python scripts were written in order to create the topologies of the proposed study.

4.4.1. Scripts for RQ#1

To evaluate the proposed SDN architecture's potential to scale for our test case, initially, one access point, one switch, and one remote controller were created. The script initially starts the topology with 10 mobile stations and eventually raises the number to 100.

The second script creates two switches, two access points, and one remote controller. The script initially starts the topology with 10 mobile stations and eventually raises the number to 100.

A third script adds a remote controller, two switches, and three access points. The script initially starts the topology with 10 mobile stations and eventually raises the number to 100.

Four access points, two switches, and one remote controller are created by the fourth script. The script initially starts the topology with 10 mobile stations and eventually raises the number to 100.

OpenFlow switches and access points were created by all scripts, and a Ryu controller was used as the remote SDN. In a 160 $\times$ 160 m rectangle, the stations were programmed to travel randomly from one spot to another. Using the iperf program, we determined the flow bandwidth and jitter for the TCP and UDP protocols. The maximum network throughput was measured by the command-line program iperf. On two randomly chosen stations, one of which serves as the server and the other as the client, the command was carried out. For instance, we used the commands below:

For UDP: *Iperf -c (IP_of_Server) 10.0.0.16 -u -p 6653 -t 30 (Client side)*
Iperf -s -u -p 6653 -i 1 > udp_result (Server side)

For TCP: *Iperf -c (IP_of_Server) 10.0.0.150 -p 6653 -t 30 (Client side)*
Iperf -s -p 6653 -i 1 > tcp_result (Server side)

4.4.2. Scripts and Commands for RQ#2

Within the scope of this work, we conducted experiments utilizing three distinct topologies: simple, linear, and custom tree topologies. There are exactly the same number of stations used across all of the tests.

Simple topology: First, we examined the most basic possible configuration, which is a network with a single access point, as depicted in Figure 3. To mimic a straightforward topology, we ran the command as follows:

$ mn -controller = remote, ip = 127.0.0.1, port = 6633-topo = single, 10 -switch = ovs

The preceding command instructs the application to conduct a search for a remote SDN controller located within the same subnet and having a port number of 6633 or 6653. Mininet-Wi-Fi is capable of performing a wide range of functions, including but not limited to connecting to the controller, constructing switches, hosts, and links, and creating network connectivity.

After Mininet-Wi-Fi confirmed the successful installation of a session, we proceeded to measure the flow setup delay by executing the following command:

sta1 ping -c1 sta10

The users (sta1 through sta10) are able to communicate with one another because they are connected to different ports of a single access point. At the access point (AP), a table-miss event is triggered whenever a user sends the first packet of a new flow to another user. This packet was generated by the first user and is headed for the second user. We

made use of the source and destination IP addresses to build an appropriate output, which enabled us to determine the flow setup latency.

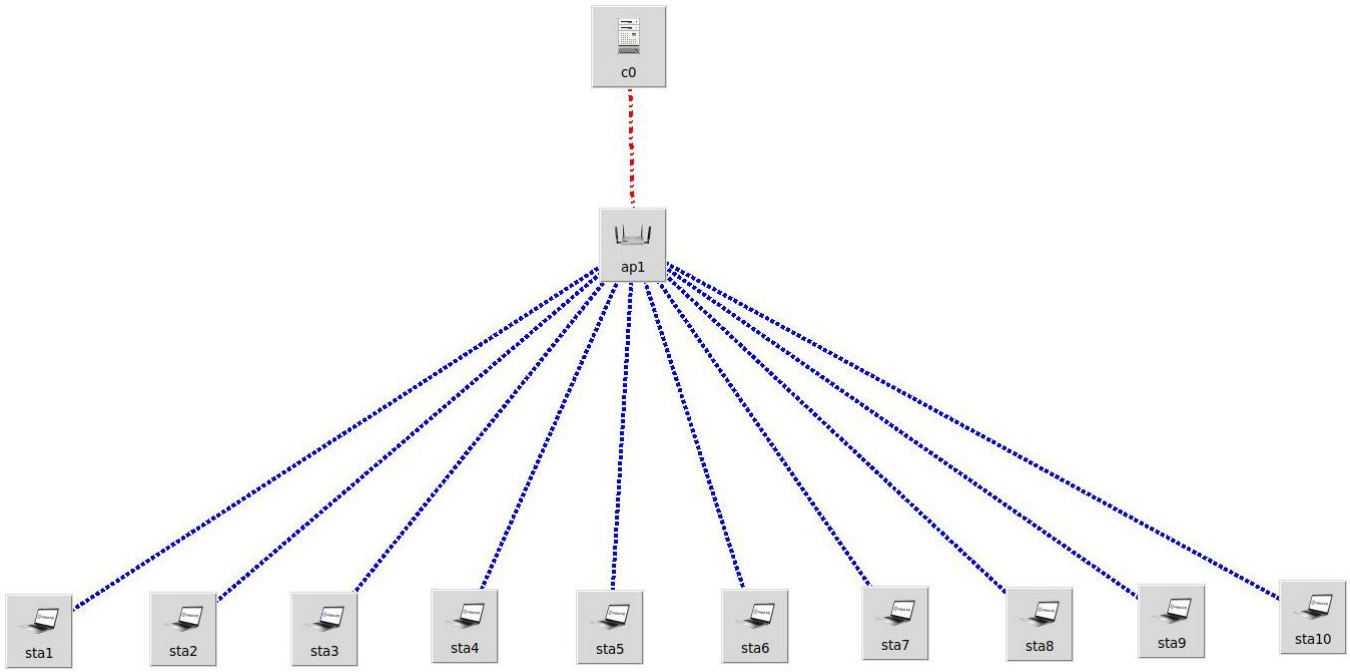

Figure 3. Simple topology.

Linear topology: In the context of SDN, a linear topology refers to a network architecture that is built with switches that are connected "back-to-back", each of which has a single host.

mn –wifi controller = remote,ip = 127.0.0.1, prot = 6633 — topo = linear, 10 -switch = ovs

As seen in Figure 4, the aforementioned instruction told Mininet-Wifi to create a linear topology.

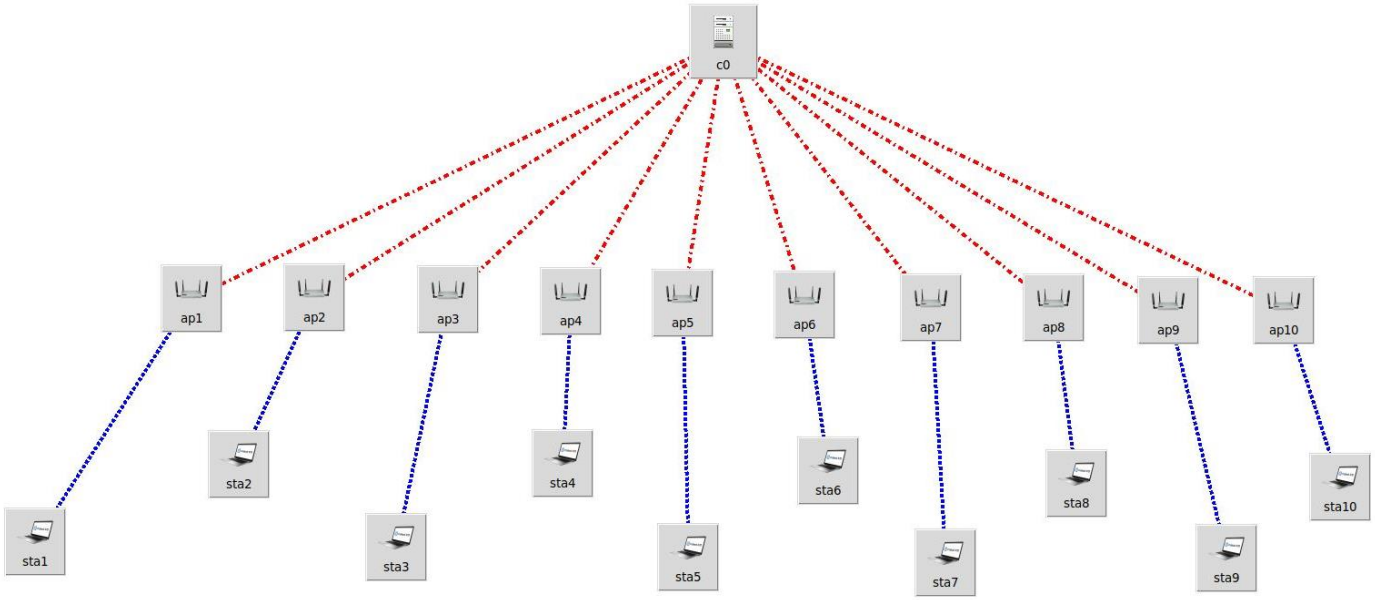

Figure 4. Mininet-Wi-Fi-created linear topology.

Users are joined to Sta1 and Sta10 in a linear topology that consists of ten access points (numbered AP1 through AP10). The users at Sta1 are the ones who generate flows in the direction of the users at Sta10. In the same manner as before, we produced ping traffic between Sta1 and Sta10 by executing the following command:

sta1 ping -c1 sta10

It is necessary for the controller to configure each of the 10 access points along the path in order to build up a flow, as shown in Figure 5. When AP1 receives a ping request, it checks to see if any flow rules match the packet. If there are none that do, it generates and sends a packet in a message to the controller. The controller will determine the necessary rule for each switch along the way, and then it will send that rule to the access points that correspond to those switches. As a direct consequence of this, the ping request will be transmitted to its intended location. A method quite identical to this one was carried out for the ping-reply.

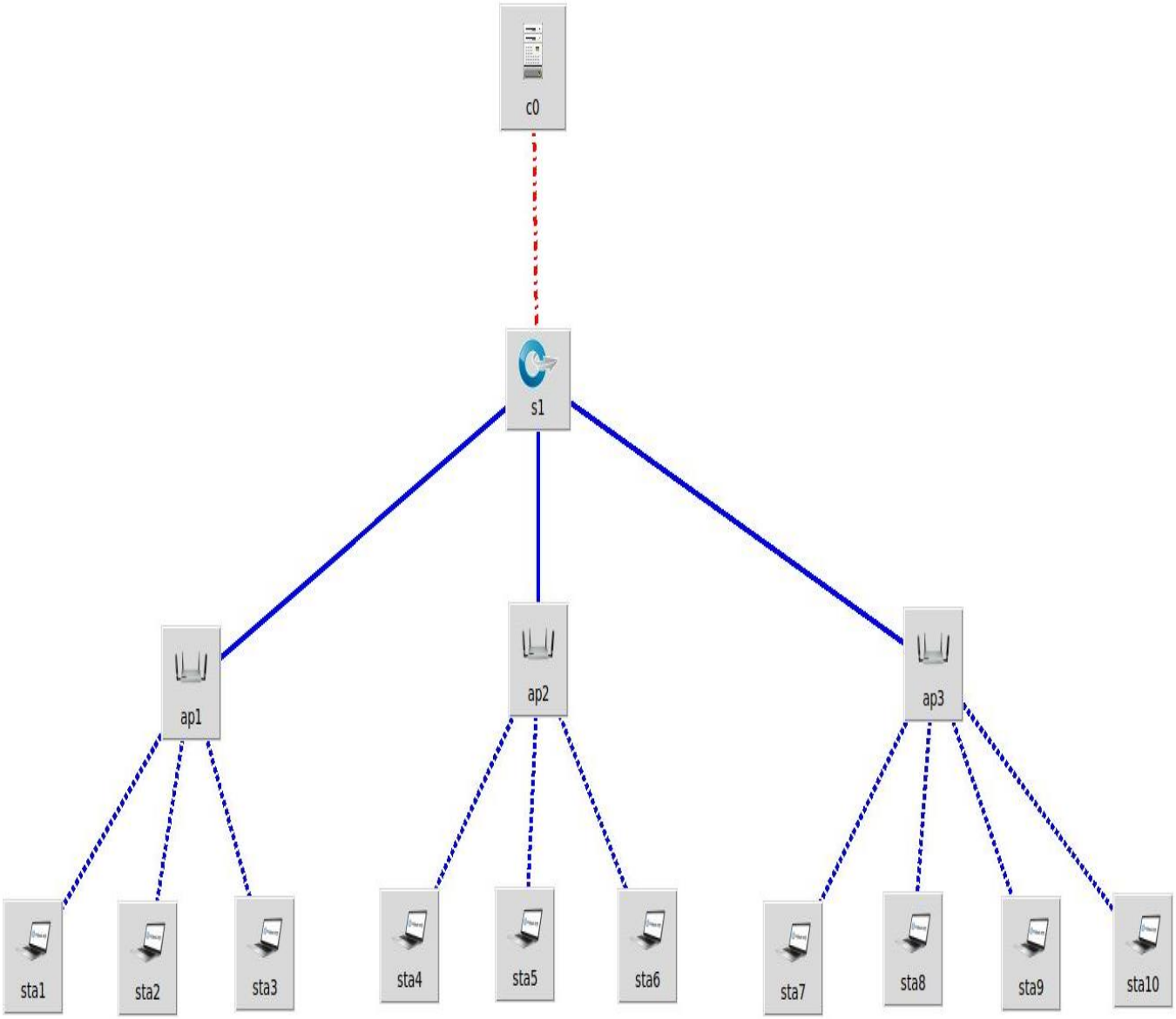

Figure 5. Custom tree topology.

4.4.3. Scripts and Commands for RQ#3

Finally, we investigated the flow setup delay in a tree topology that has a higher degree of complexity. To generate the tree structure depicted in Figure 5, we crafted a script in the programming language Python. We carried out a number of iterations of the script, gradually raising the number of switches in the hierarchy from one to twelve and recording the flow setup latency for each of these iterations. The number of access points and the number of stations were both kept constant during each execution.

Custom tree topology: We investigate the flow setup delay in a topology based on a bespoke tree. To generate the tree structure depicted in Figure 5, we crafted a script in the programming language Python. As can be seen in Figure 5, the generated topology is composed of three layers: one controller, one OpenFlow Switch, three access points, and ten stations.

The following command was used to generate ping traffic between stations Sta1 and Sta10:

sta1 ping -c1 sta10

The stations that are located at the source and the destination are not connected to the same access point. As a result, their routes must pass through a minimum of one core switch and two access points; the flow path lengths involve a total of four hops. For the purpose of determining the flow setup latency of tree topology, the round-trip time of ping requests was measured.

Selecting relatively simple topologies when using Ryu to build a Wi-Fi network is easier to configure and manage with a limited number of switches and access points. Network administrators avoid potential configuration errors or issues, making the network more reliable and easier to troubleshoot. It is cost-effective and more scalable. Network administrators can add new switches and access points as needed without having to reconfigure the entire network. It is more efficient, as with fewer switches and access points, network traffic can be streamlined, reducing the likelihood of network congestion or performance issues. There may be situations where a more complex topology is necessary; however, selecting a simple topology can provide several benefits, including ease of management, cost-effectiveness, scalability, and efficiency when building a Wi-Fi network using Ryu.

5. Experiment, Results and Discussion

We used the Hazara University smart network infrastructure as a real-world scenario to validate our findings. The Hazara University smart network is set up as follows:

It has 406 Access Points, 402 layer 2 switches, 24 switches, 1 router, and 470 megabytes of bandwidth. More than 6000 users have registered on Smart Net. The network can support a maximum of 4500 concurrent users, but often only 1200 users are active at once.

- In order to assess performance and scalability, we experimented with Ryu SDN controllers and a real-world Wi-Fi network situation in this work.
- Using the Mininet-Wi-Fi network emulator, which is a fork of Mininet, we emulated a Wi-Fi network [5].

5.1. Experiments and Results

Mininet-Wi-Fi was employed to create the topologies described in the previous section on various network sizes by connecting to a different controller each time. In this section, we discuss the evaluation results and the performance measures we used to make comparisons.

RQ# 01: The output of Mininet-Wi-Fi scripts and commands created for RQ#1 is described in this section. In the complex uneven topology, scripts with 1 access point and 10, 30, 50, or 100 stations were created and executed to test the effectiveness. Scripts with 2, 3, and 4 numbers of access points were run in a similar manner to assess the performance, and the results are presented in Figures 6–9. Using the iperf program, we determined the flow bandwidth and jitter for the TCP and UDP protocols.

5.1.1. TCP Graph

Figure 6 illustrates the creation of a TCP bandwidth flow graph for 1 access point and 10, 30, 50, and 100 stations using the iperf log files. The *Y*-axis displays bandwidth flow in kbits/s, and the *X*-axis displays time in seconds. Similar graphs for 2, 3, and 4 access points with 10, 30, 50, and 100 stations are shown in Figures 7–9.

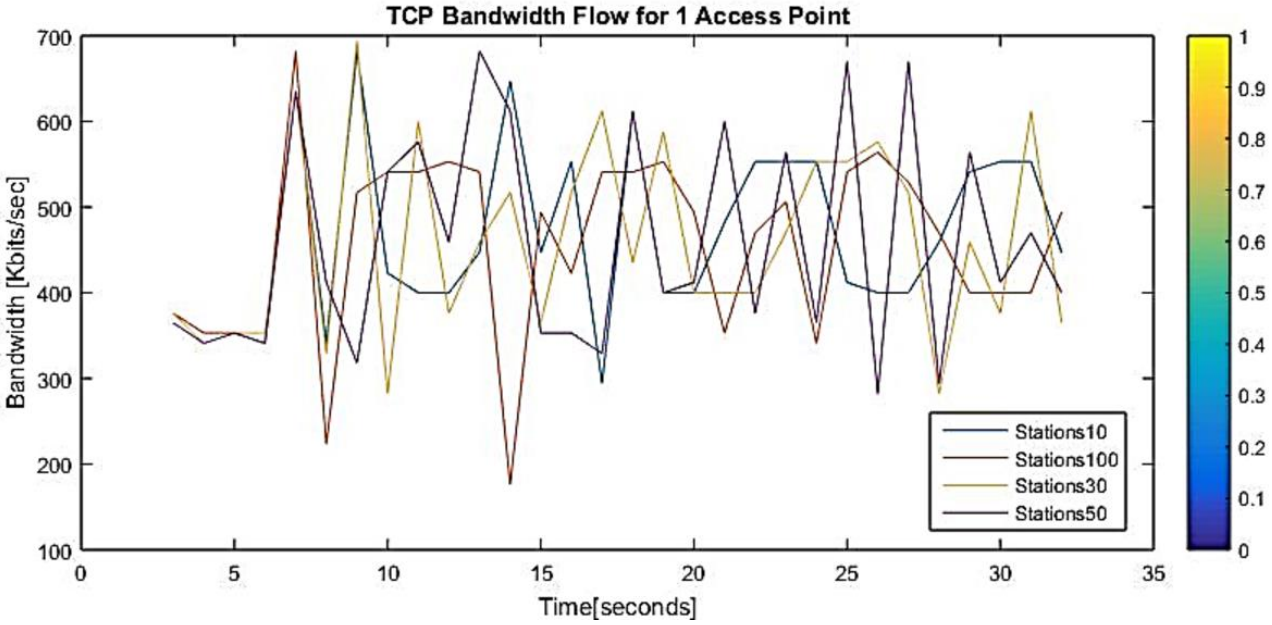

Figure 6. TCP bandwidth for 1 access point.

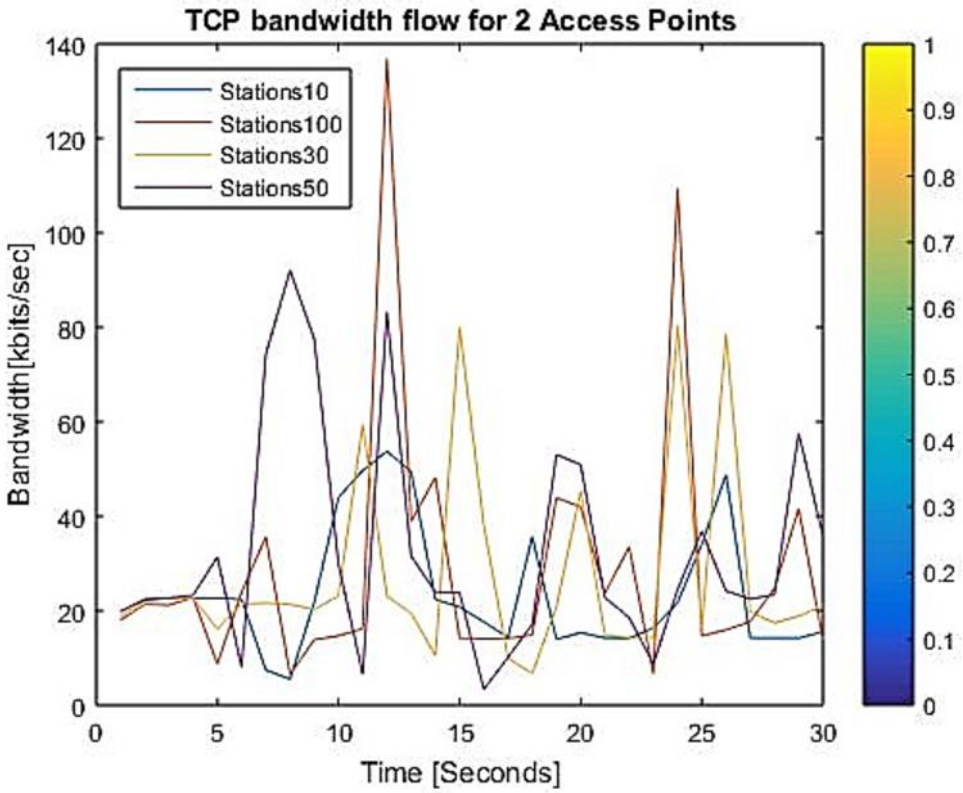

Figure 7. TCP bandwidth for 2 access points.

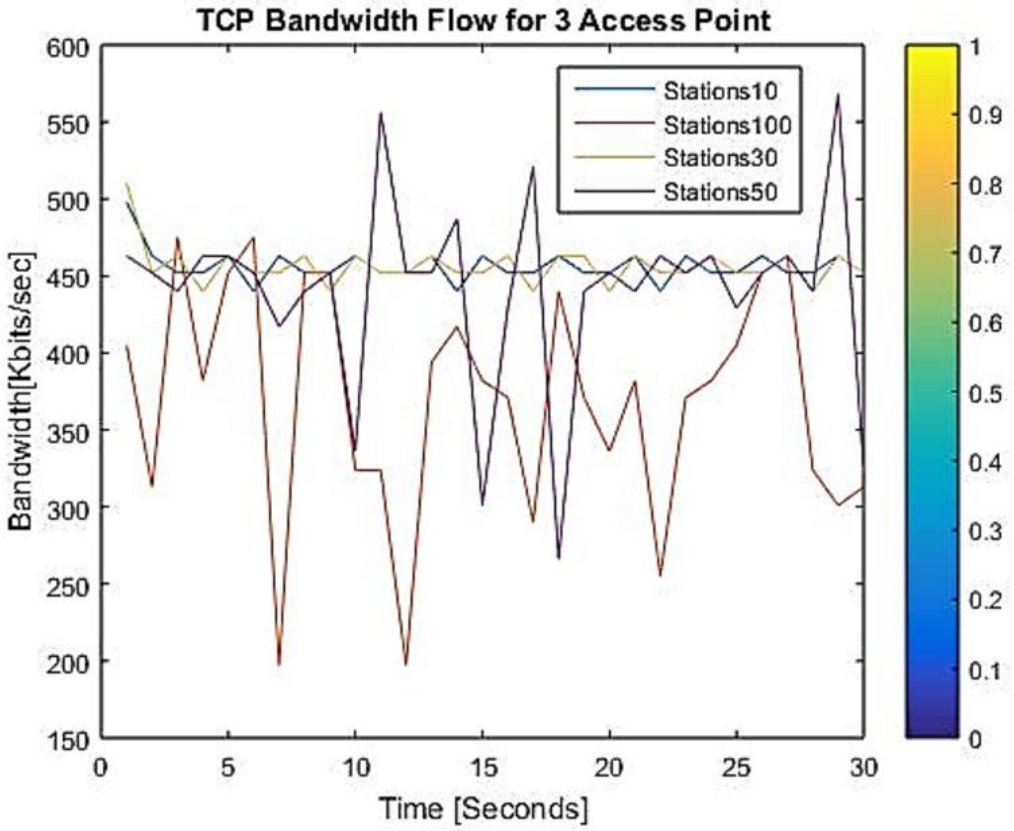

Figure 8. TCP bandwidth for 3 access points.

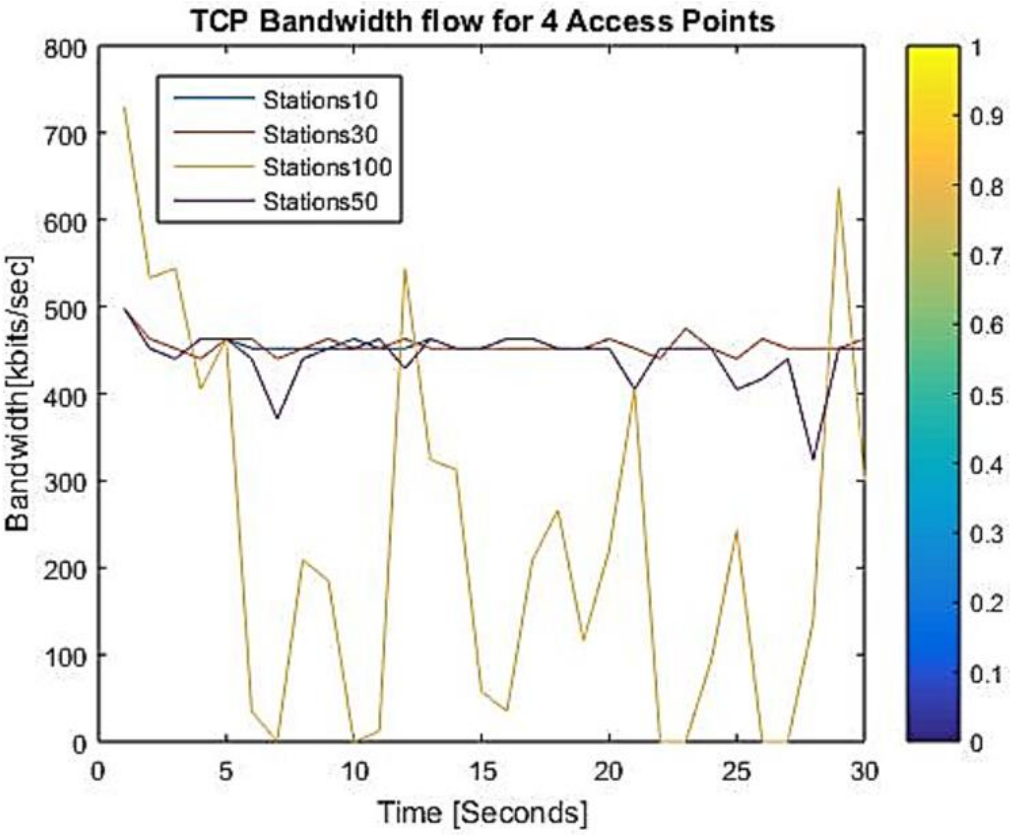

Figure 9. TCP bandwidth for 4 access points.

Result comparison for TCP bandwidth: Figures 6–9 indicate that the bandwidth varies between 440 and 498 kb/s for 1 access point and 10, 30, 50, and 100 stations, whereas it varies between 255 and 498 kb/s for 100 stations. As a result, we can see that up to 50 stations the bandwidth does not change significantly. However, bandwidth diminishes after 100 stations. The bandwidth flow for two access points with 10 to 30 stations remains constant and ranges between 440 and 498 kb/s; beyond 50 stations, the bandwidth flow starts to decline and ranges between 382 and 510, and the bandwidth for 100 stations is 209–544. For three access points, the bandwidth flow values for 10, 30, 50, and 100 stations are 440–498 kb/s, 440–510 kb/s, 266–568 kb/s, and 197–475 kb/s, respectively.

The bandwidth flow values for 10, 30, 50, and 100 stations for 4 access points are 440–498 kb/s, 440–498 kb/s, 324–498 kb/s, and 0–730 kb/s, respectively. We may conclude that SDN networks scale well because there is little difference between the bandwidth flow of the emulated networks for 10 and 30 stations. However, the performance is affected by going from 50 to 100 stations. Furthermore, performance is not improved by adding more access points.

5.1.2. UDP Parameters Graph

Figures 10–13 are based on the iperf log files and depict the UDP bandwidth graphs for access points 1, 2, 3, and 4. The *Y*-axis displays bandwidth in kbits/s, and the *X*-axis displays time in seconds.

Figures 14–17 display the UDP jitter graphs; the *X*-axis displays time in seconds and the *Y*-axis displays jitter in milliseconds. The bandwidth for 10, 30, and 50 stations is comparable and spans from 282 to 694 kb/s, however for 100 stations, it drops to 172 to 682 kb/s. As the number of stations grows gradually, i.e., 53.675, 80.528, 92.052, and 136.52 ms, jitter somewhat increases.

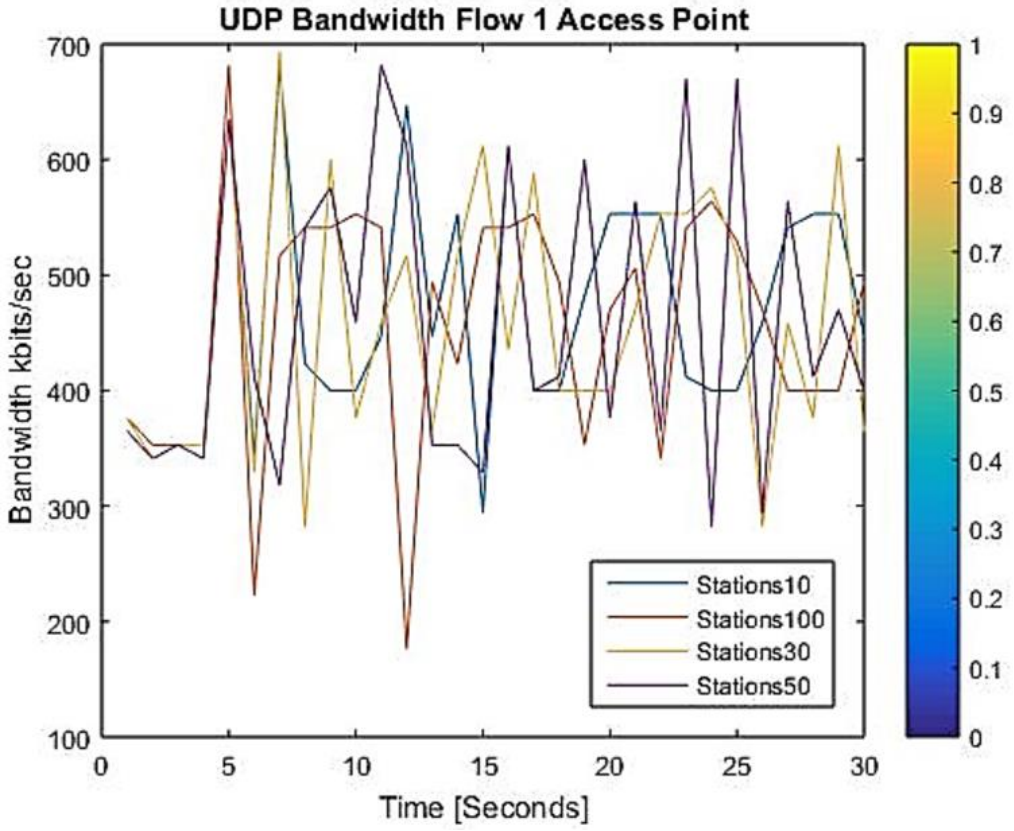

Figure 10. UDP bandwidth for 1 access point.

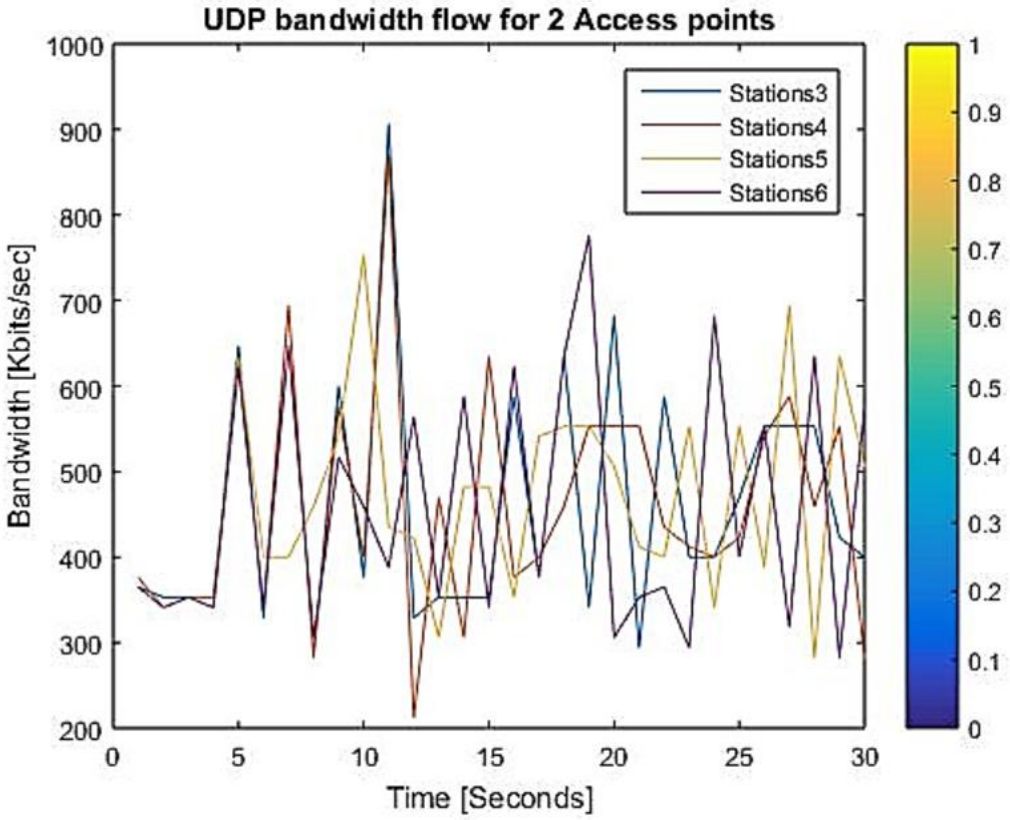

Figure 11. UDP bandwidth for 2 access points.

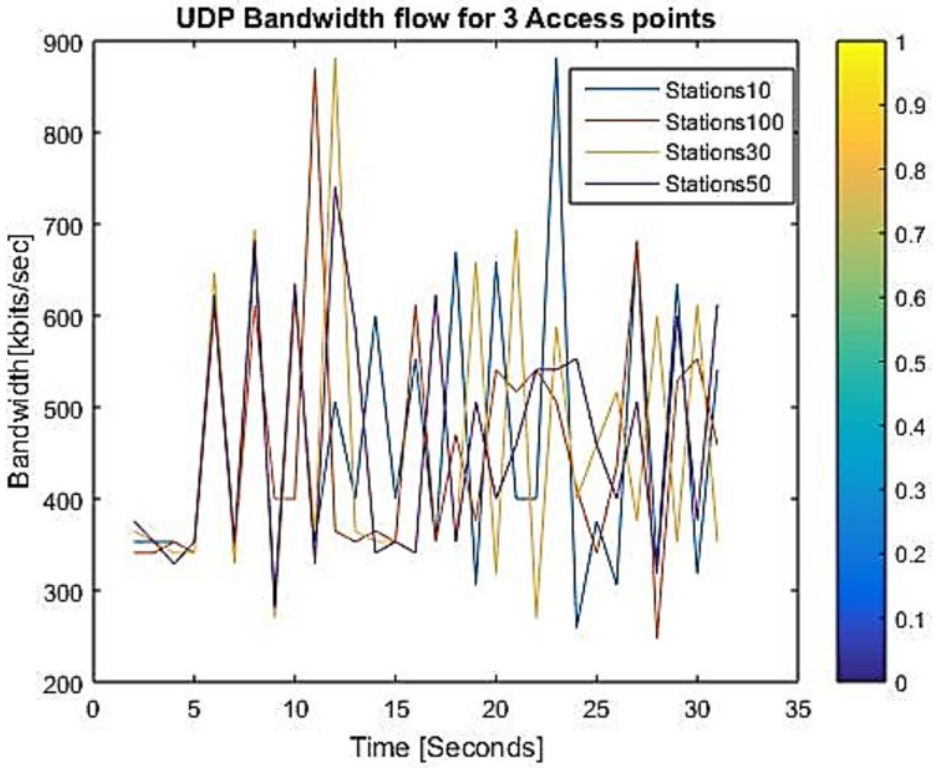

Figure 12. UDP bandwidth for 3 access points.

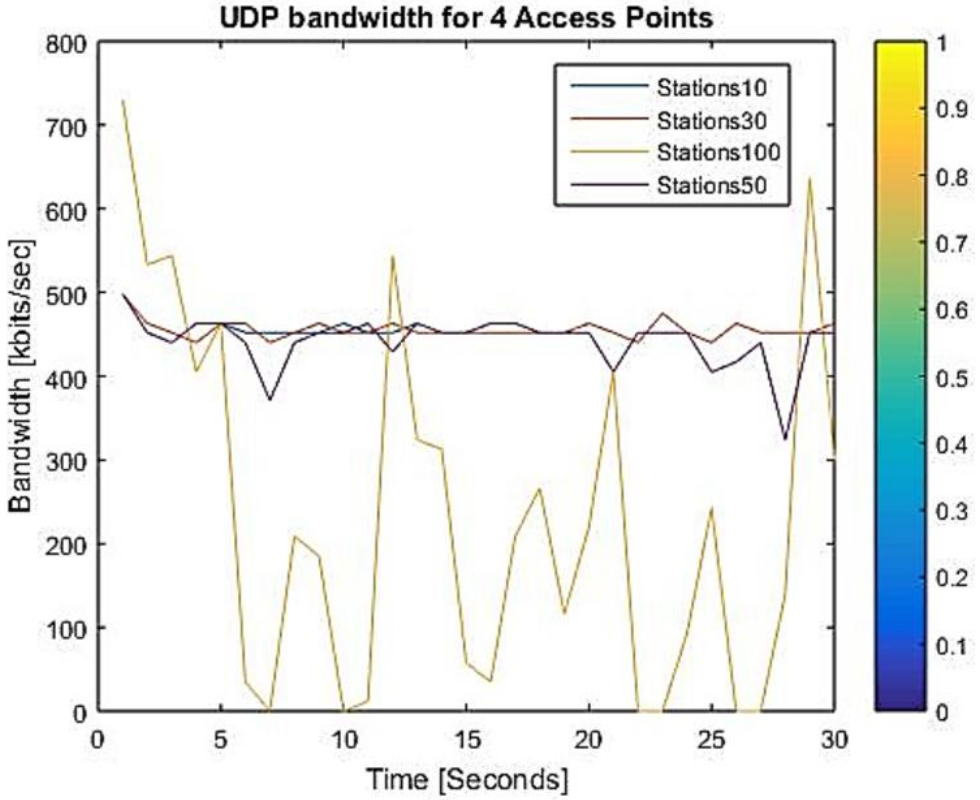

Figure 13. UDP bandwidth for 4 access points.

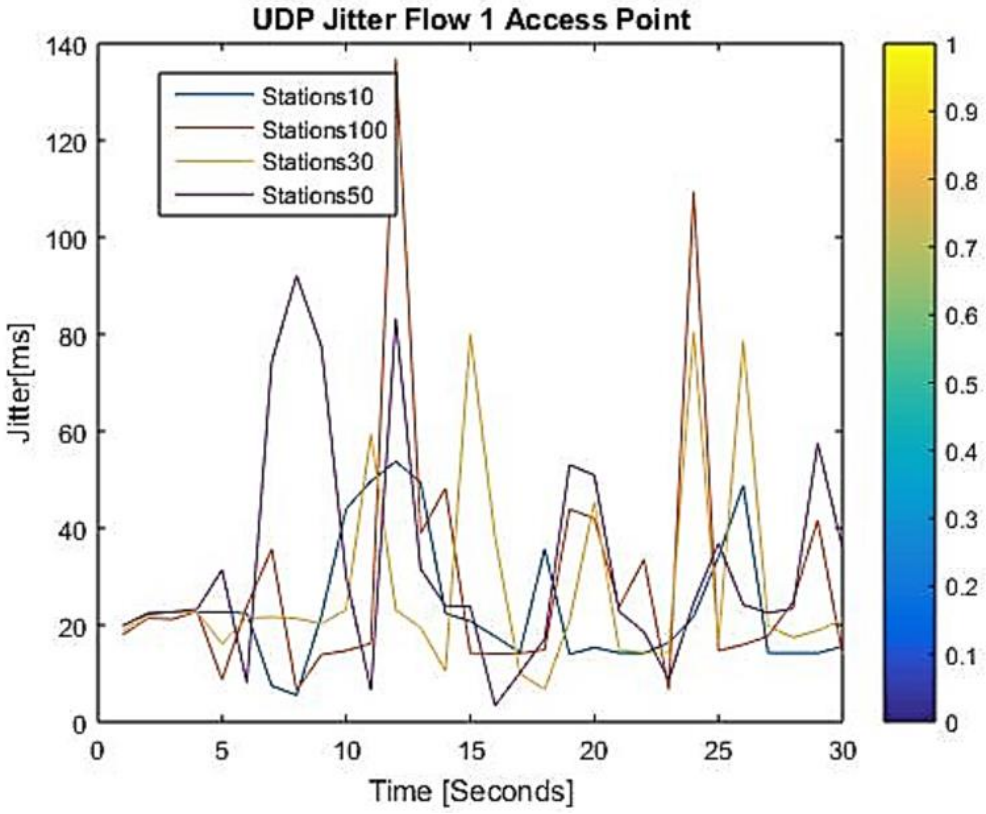

Figure 14. UDP jitter for 1 access point.

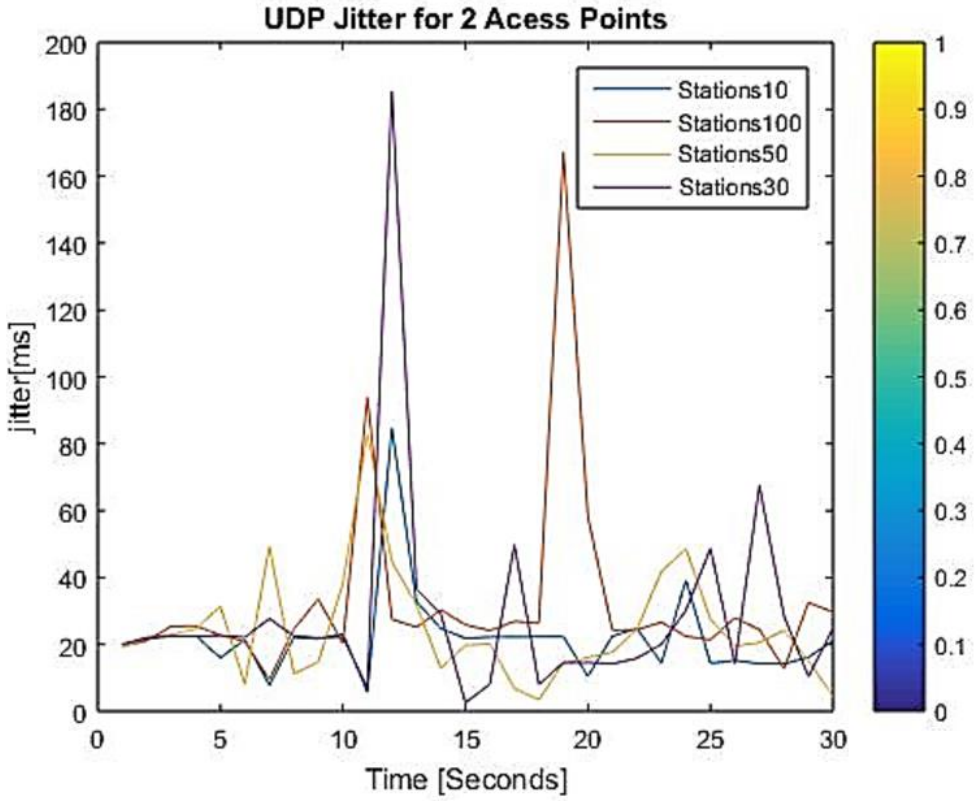

Figure 15. UDP jitter for 2 access points.

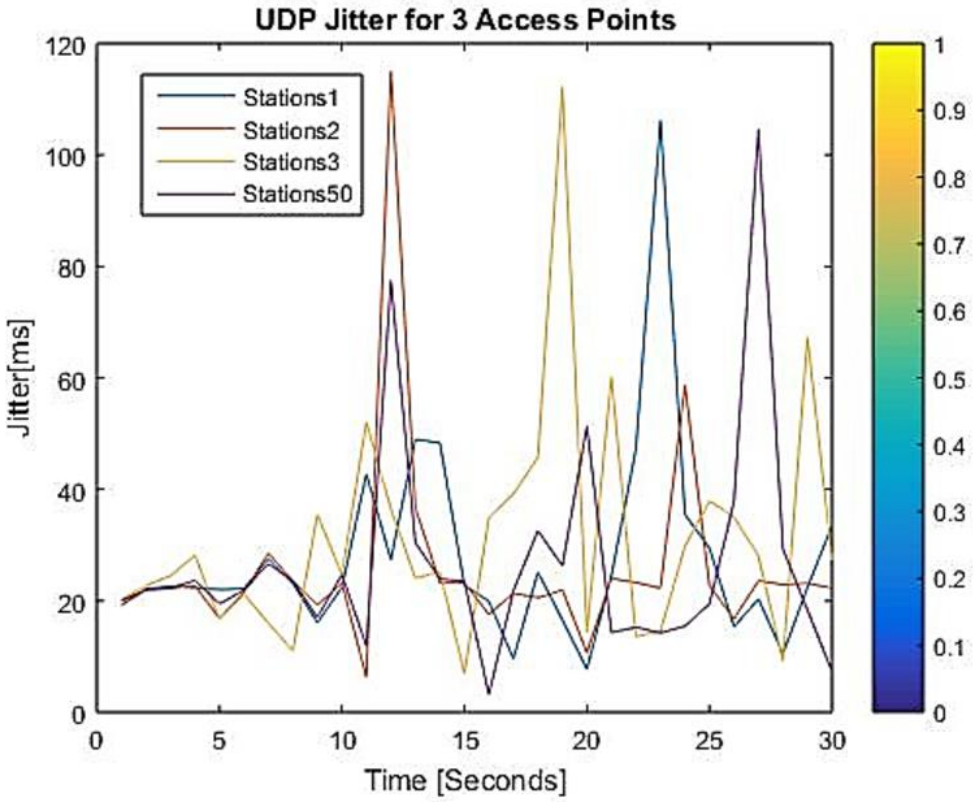

Figure 16. UDP jitter for 3 access points.

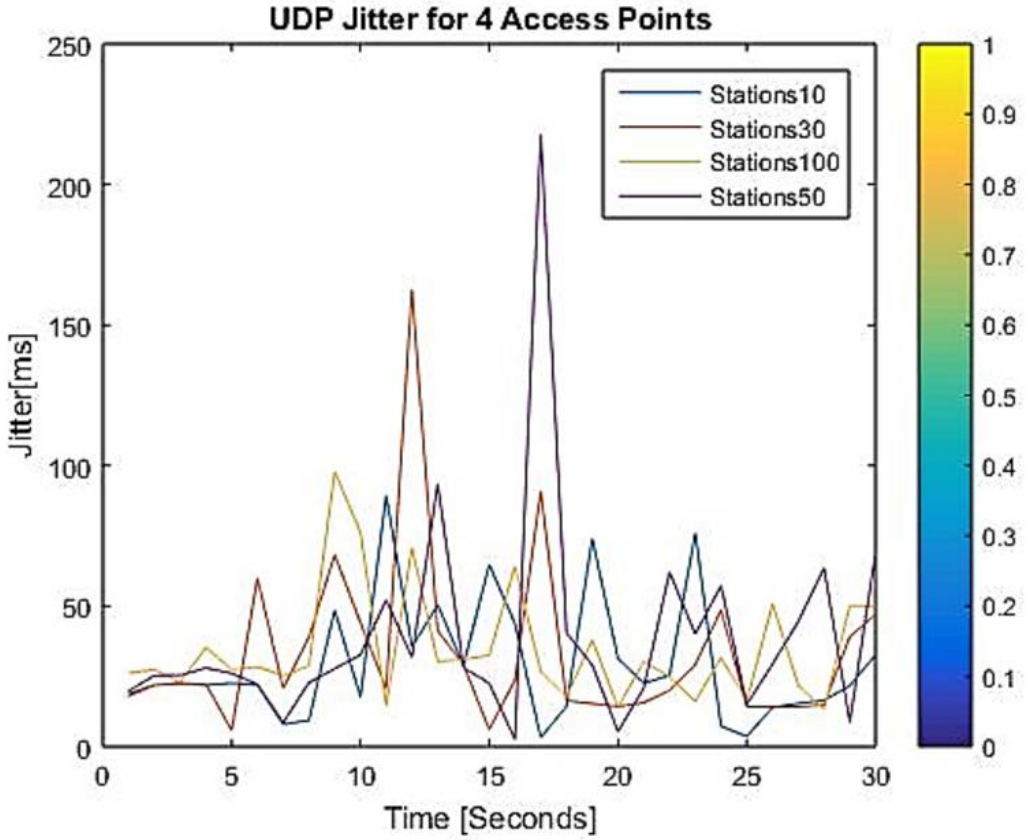

Figure 17. UDP jitter for 4 access points.

RQ#2: The most important factor in determining the flow setup delay of SDN controllers is differences in network architecture. We concentrated our experiments on simple, linear, and tree topologies. Mininet-Wi-Fi topologies can be generated using scripts and commands presented in the previous section. As an example, we used a network with a single access point connected to the Ryu controller and ten stations attached to this access point as the simplest scenario. Stations trade ping flows to calculate the latency of the flow arrangement. The controller configures the uplink flow rule at the access point and passes the packet across the relevant outport of the access point for the first ping request from station one to station two. We calculated the packet's round-trip time and noted it as flow setup latency. By sending ping requests from station one to station three, station ten, and so on, we repeated the process. We then calculated the mean of each flow setup delay. For a straightforward design, the typical flow startup latency is 0.75. Next, as indicated in the previous section, we took into account a linear topology made up of ten access points (AP1, AP2, . . . , AP10), with one station connected to each of them. To assess flow setup latency, we followed the same steps as we did for simple topology. Our estimated average flow setup time is 47.5 ms; the linear architecture has a substantially higher flow setup latency than the basic topology.

Here, we present a report on the setup delay for a complex flow. We took into account a unique tree topology as defined in the preceding section. We took into account a hierarchy that consisted of a controller, a switch, three access points, and three stations connected to each AP. To assess flow setup latency, we followed the same steps as we did for simple topology. The average flow setup latency, we calculated, is 50.54 ms, which is much more than simple topology and hardly more than linear topology. We display the measurement results for various network topologies in Figure 18.

RQ#3: We then took into account a more complicated situation. We took into account a unique tree topology, as defined in the preceding section. We took into account a hierarchy that consisted of a controller, a switch, three access points, and three stations connected to

each AP. We ran the RQ#3 script, increasing the number of switches with each run. Stations trade ping flows to calculate the latency of the flow arrangement. The controller receives a packet for the initial ping request from station one to station two.

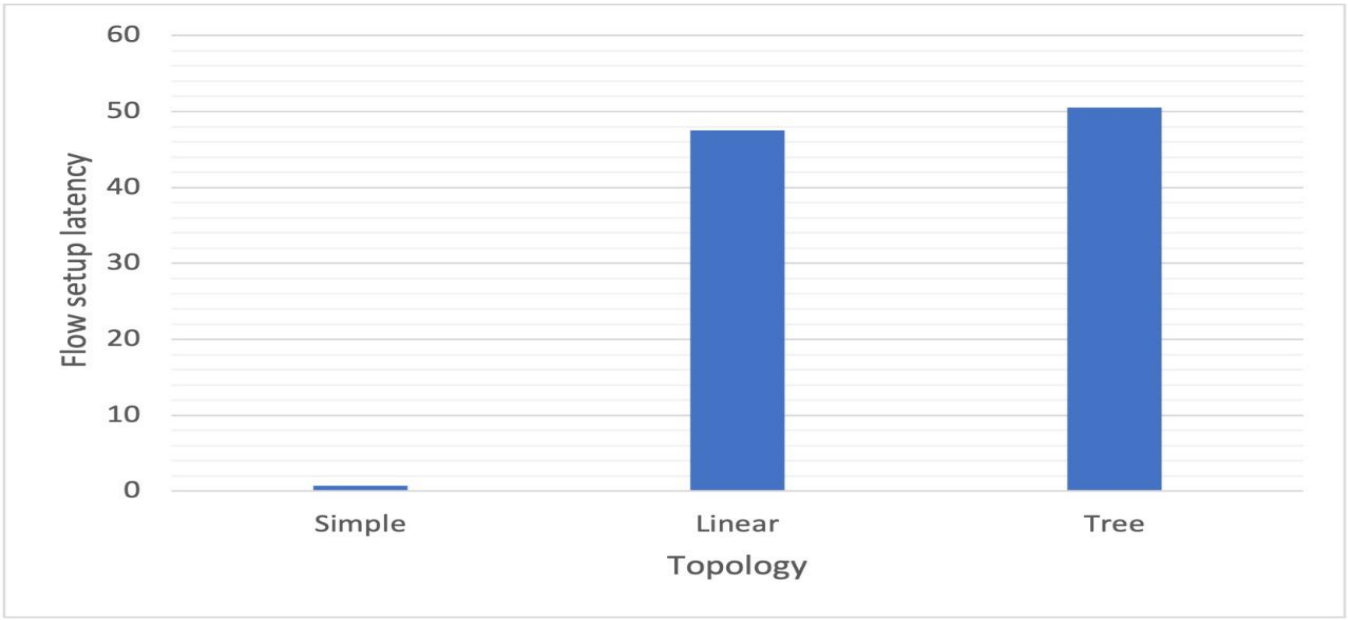

Figure 18. Graph showing flow setup latency with three topologies.

We calculated the packet's round-trip time and noted it as flow setup latency. By sending ping requests from station one to station three, station ten, and so on, we repeated the process. We then calculated the mean of each flow setup delay. The flow setup latencies for each execution are shown in Figure 19. As we raised the number of switches, we saw that the flow setup latencies drastically increased.

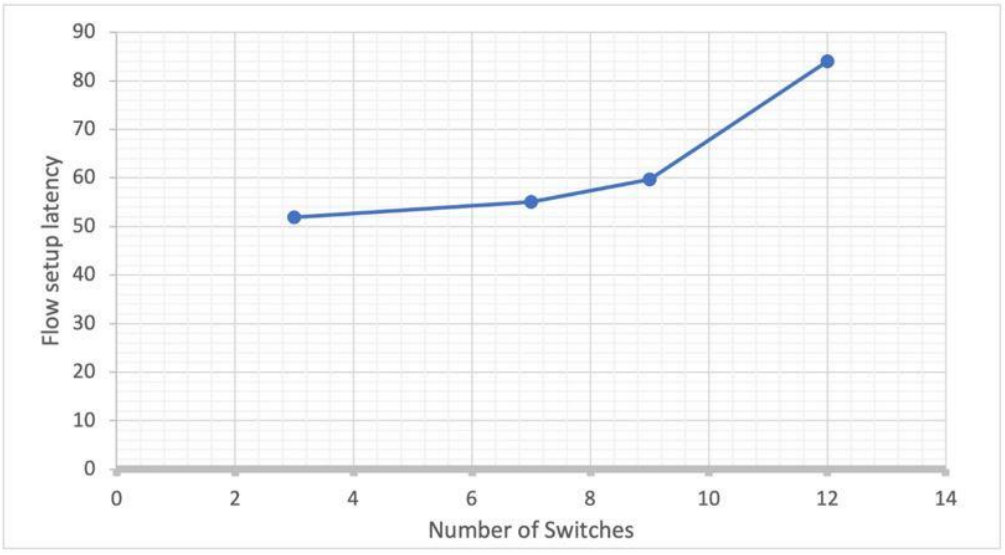

Figure 19. Graph showing flow setup latency.

The performance and scalability analysis of SDN-based large-scale Wi-Fi networks is a novel approach to manage and control large-scale Wi-Fi networks using SDN technologies. Using various performance metrics, this study looks at how well the proposed SDN-based architecture works and how well it can grow. The proposed method can dynamically distribute network resources based on how the network is working in real time. This

lets the network make the best use of its resources, reduces congestion, and improves the network's performance. Using SDN technology makes it possible to control and manage the network from one place. This makes setting up and managing the network easier. However, the proposed study does not focus on mobility because it assumes that the Wi-Fi network is set up in a fixed location, such as the smart campus described in the proposed study. Mobility brings new problems, such as switching between access points and network congestion caused by clients moving around. These problems need more network resources and ways to manage them and can be addressed in future work. Therefore, by focusing on a fixed location deployment, the study can isolate the effects of scalability and performance on the network and provide a more accurate analysis of the proposed approach's effectiveness.

6. Conclusions

We examined the scalability and performance of SDN-based large-scale Wi-Fi networks. SDN is a paradigm that uses programmable interfaces to seamlessly link application provisioning in the cloud with the network. In SDN, the control plane and forwarding plane are independent. SDN is exclusively used in the proposed study's major data centers. However, wireless networks are common and well-liked. Issues are also developing with wireless networks' scalability and performance as their density increases. We require innovative control and administration solutions to address the performance and scalability problems in Wi-Fi networks. Scalability and performance problems in wireless networks are thought to be resolved using SDN techniques. Therefore, we used the TCP and UDP protocols to assess Wi-Fi networks on SDN. Using a testbed comprised of Mininet-Wi-Fi and a Ryu Controller, we investigated Wi-Fi over SDN and assessed its functionality and scalability. In a large-scale scenario, we focused on SDN Wi-Fi network controllers. The SDN Wi-Fi controllers were created with the ability to work with SDN infrastructure in mind. This study assesses the scalability and performance of an SDN-based Wi-Fi network controller. We gauge network performance using variables such as bandwidth, round-trip time, and jitter. In order to assess the performance, the SDN-based Wi-Fi controller (Ryu) linked to an increasing number of stations (10, 30, 50, and 100) and access points (1, 2, 3, and 4). The network performance of Ryu, an SDN Wi-Fi network controller, is scalable and stable in an SDN environment, according to experimental results utilizing Mininet-Wi-Fi. Additionally, increasing the hop count lengthens the distance travelled by TCP transmissions. A total of 50 concurrent users can be supported by a single access point at once.

Future performance analysis of genuine large-scale Low-Power Wide-Area Networks (LPWAN) in SDN networks for the Internet of Things will be included in the planned study's extension. Most SDN studies that have been proposed in the literature do not take SDN Wi-Fi networks into account. However, the ubiquitous use of (LPWAN) networks and their significance for the Internet of Things needs their integration with SDN. We assumed that many Wi-Fi network performance difficulties might be overcome by merging LPWAN networks with SDN controllers.

Author Contributions: Conceptualization, M.A., A.I.J., M.A.A. and R.S.; Data curation, O.I.A., Z.A. and R.S.; Formal analysis, M.A., A.I.J., O.I.A., Z.A., R.M.G. and M.A.A.; Funding acquisition, R.M.G.; Investigation, A.I.J., O.I.A., R.M.G. and M.A.A.; Methodology, M.A., Z.A., M.A.A. and R.S.; Project administration, Z.A. and R.M.G.; Resources, A.I.J., R.M.G. and R.S.; Software, M.A., A.I.J., O.I.A. and R.S.; Supervision, A.I.J. and Z.A.; Validation, A.I.J.; Visualization, O.I.A., Z.A., R.M.G., M.A.A. and R.S.; Writing—original draft, M.A., Z.A. and R.S.; Writing—review and editing, A.I.J., O.I.A., R.M.G. and M.A.A. All authors have read and agreed to the published version of the manuscript.

Funding: Princess Nourah bint Abdulrahman University Researchers Supporting Project number (PNURSP2023R138), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia.

Data Availability Statement: No data were used to support this study. However, any query about the research conducted in this paper is highly appreciated and can be asked from the corresponding authors upon request.

Acknowledgments: The authors appreciate the support from Princess Nourah bint Abdulrahman University Researchers Supporting Project number (PNURSP2023R138), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia. The authors also thank the support from Omar Imhemed Alramli for English proofreading of the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Isong, B.; Molose, R.R.S.; Abu-Mahfouz, A.M.; Dladlu, N. Comprehensive Review of SDN Controller Placement Strategies. *IEEE Access* **2020**, *8*, 170070–170092. [CrossRef]
2. Liu, L.; Jiang, Y.; Shen, G.; Li, Q.; Lin, D.; Li, L.; Wang, Y. An SDN-based Hybrid Strategy for Load Balancing in Data Center Networks. In Proceedings of the 2019 IEEE Symposium on Computers and Communications (ISCC), Barcelona, Spain, 29 June–3 July 2019; pp. 1–6. [CrossRef]
3. Shirmarz, A.; Ghaffari, A. Performance issues and solutions in SDN-based data center: A survey. *J. Supercomput.* **2020**, *76*, 7545–7593. [CrossRef]
4. Abdollahi, S.; Deldari, A.; Asadi, H.; Montazerolghaem, A.; Mazinani, S.M. Flow-Aware Forwarding in SDN Datacenters Using a Knapsack-PSO-Based Solution. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 2902–2914. [CrossRef]
5. Goswami, B.; Asadollahi, S.S. Enhancement of LAN Infrastructure Performance for Data Center in Presence of Network Security. In *Next-Generation Networks*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 419–432. [CrossRef]
6. Das, S.; Goswami, B.; Asadollahi, S. Investigating Software-Defined Network and Networks-Function Virtualization for Emergent Network-oriented Services. *Int. J. Innov. Res. Comput. Commun. Eng.* **2017**, *5*, 201–205.
7. Asadollahi, S. Revolution in Existing Network under the Influence of Software Defined Network. In Proceedings of the 11th INDIACom, New Delhi, India, 1–3 March 2017; pp. 1012–1017.
8. Lin, S.; Che, N.; Yu, F.; Jiang, S. Fairness and Load Balancing in SDWN Using Handoff-Delay-Based Association Control and Load Monitoring. *IEEE Access* **2019**, *7*, 136934–136950. [CrossRef]
9. Ouamri, M.A.; Barb, G.; Singh, D.; Alexa, F. Load Balancing Optimization in Software-Defined Wide Area Networking (SD-WAN) using Deep Reinforcement Learning. In Proceedings of the 2022 International Symposium on Electronics and Telecommunications (ISETC), Bandung, Indonesia, 10–11 November 2022; pp. 1–6. [CrossRef]
10. Gonsai, A.M. Experimental Based Performance Testing of Different TCP Protocol Variants in comparison of RCP + over Hybrid Network Scenario. In Proceedings of the 2014 IFIP Networking Conference, Trondheim, Norway, 2–4 June 2014; Volume 3, pp. 31–37.
11. Islam, S.; Khan, A.I.; Shorno, S.T.; Sarker, S.; Siddik, A. Performance Evaluation of SDN Controllers in Wireless Network. In Proceedings of the 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT), Dhaka, Bangladesh, 3–5 May 2019; pp. 1–5. [CrossRef]
12. Hassan, A.H.M.; Alhassan, A.M.; Izzeldean, F. Performance Evaluation of SDN Controllers in Ofnet Emulation Environment. In Proceedings of the 2019 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE), North Khartoum, Sudan, 21–23 September 2019. [CrossRef]
13. Drutskoy, D.; Keller, E.; Rexford, J. Scalable Network Virtualization in Software-Defined Networks. *IEEE Internet Comput.* **2012**, *17*, 20–27. [CrossRef]
14. Yang, G.; Yu, B.-Y.; Jin, H.; Yoo, C. Libera for Programmable Network Virtualization. *IEEE Commun. Mag.* **2020**, *58*, 38–44. [CrossRef]
15. Bhuvaneswaran, V.; Basil, A.; Tassinari, M.; Manral, V.; Banks, S. *Terminology for Benchmarking Software-Defined Networking (SDN) Controller Performance*; RFC Editors: Los Angeles, CA, USA, 20 October 2018. [CrossRef]
16. Bholebawa, I.Z.; Dalal, U.D. Performance Analysis of SDN/OpenFlow Controllers: POX Versus Floodlight. *Wirel. Pers. Commun.* **2017**, *98*, 1679–1699. [CrossRef]
17. Shamim, S.M. Performance Analysis of Different Openflow based Controller Over Software Defined Networking. *Glob. J. Comput. Sci. Technol. C* **2018**, *18*, 11–15.
18. Fancy, C.; Pushpalatha, M. Performance evaluation of SDN controllers POX and floodlight in mininet emulation environment. In Proceedings of the 2017 International Conference on Intelligent Sustainable Systems (ICISS), Palladam, India, 7–8 December 2017; pp. 695–699. [CrossRef]
19. Mamushiane, L.; Lysko, A.; Dlamini, S. A comparative evaluation of the performance of popular SDN controllers. *IFIP Wirel. Days* **2018**, *2018*, 54–59. [CrossRef]
20. Rastogi, A.; Bais, A. Comparative analysis of software defined networking (SDN) controllers—In terms of traffic handling capabilities. In Proceedings of the 2016 19th International Multi-Topic Conference (INMIC), Islamabad, Pakistan, 5–6 December 2016; pp. 1–6. [CrossRef]
21. Kumar, A.; Goswami, B.; Augustine, P. Experimenting with resilience and scalability of wifi mininet on small to large SDN networks. *Int. J. Recent Technol. Eng.* **2019**, *7*, 201–207.
22. Taha, M. An efficient software defined network controller based routing adaptation for enhancing QoE of multimedia streaming service. *Multimed. Tools Appl.* **2023**. [CrossRef]

23. Zhu, L.; Karim, M.; Sharif, K.; Xu, C.; Li, F.; Du, X.; Guizani, M. SDN Controllers. *ACM Comput. Surv.* **2020**, *53*, 1–40. [CrossRef]
24. Bhardwaj, S.; Panda, S.N. Performance Evaluation Using RYU SDN Controller in Software-Defined Networking Environment. *Wirel. Pers. Commun.* **2021**, *122*, 701–723. [CrossRef]
25. Smida, K.; Tounsi, H.; Frikha, M.; Song, Y.-Q. Efficient SDN Controller for Safety Applications in SDN-Based Vehicular Networks: POX, Floodlight, ONOS or OpenDaylight? In Proceedings of the 2020 IEEE Eighth International Conference on Communications and Networking (ComNet), Hammamet, Tunisia, 27–30 October 2020; pp. 1–6. [CrossRef]
26. Hu, J.; Lin, C.; Li, X.; Huang, J. Scalability of control planes for Software defined networks: Modeling and evaluation. In Proceedings of the 2014 IEEE 22nd International Symposium of Quality of Service (IWQoS), Hong Kong, China, 26–27 May 2014; pp. 147–152. [CrossRef]
27. Karakus, M.; Durresi, A. A Scalability Metric for Control Planes in Software Defined Networks (SDNs). In Proceedings of the 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA), Crans-Montana, Switzerland, 23–25 March 2016; pp. 282–289. [CrossRef]
28. Bianco, A.; Giaccone, P.; Mahmood, A.; Ullio, M.; Vercellone, V. Evaluating the SDN control traffic in large ISP networks. In Proceedings of the 2015 IEEE International Conference on Communications (ICC), London, UK, 8–12 June 2015; pp. 5248–5253. [CrossRef]
29. Yang, G.; Shin, C.; Yoo, Y.; Yoo, C. A Case for SDN-based Network Virtualization. In Proceedings of the 2021 29th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Houston, TX, USA, 3–5 November 2021; pp. 1–8. [CrossRef]
30. Yu, B.; Yang, G.; Yoo, C. Comprehensive Prediction Models of Control Traffic for SDN Controllers. In Proceedings of the 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft), Montreal, QC, Canada, 25–29 June 2018; pp. 262–266. [CrossRef]
31. Eljack, A.H.; Hassan, A.H.M.; Elamin, H.H. Performance Analysis of ONOS and Floodlight SDN Controllers based on TCP and UDP Traffic. In Proceedings of the 2019 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE), North Khartoum, Sudan, 21–23 September 2019. [CrossRef]
32. Muqaddas, A.S.; Bianco, A.; Giaccone, P.; Maier, G. Inter-controller traffic in ONOS clusters for SDN networks. In Proceedings of the 2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, Malaysia, 22–27 May 2016; pp. 1–6. [CrossRef]
33. Bianco, A.; Giaccone, P.; Mashayekhi, R.; Ullio, M.; Vercellone, V. Scalability of ONOS reactive forwarding applications in ISP networks. *Comput. Commun.* **2017**, *102*, 130–138. [CrossRef]
34. Secci, S.; Diamanti, A.; Sanchez, J.M.V.; Bah, M.T.; Vizarreta, P.; Machuca, C.M.; Scott-Hayward, S.; Smith, D. Security and Performance Comparison of ONOS and ODL Controllers. 2019. Available online: https://hal.science/hal-03188550 (accessed on 15 August 2021).
35. Gude, N.; Koponen, T.; Pettit, J.; Pfaff, B.; Casado, M.; McKeown, N.; Shenker, S. NOX: Towards an operating system for networks. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 105–110. [CrossRef]
36. Noman, H.M.; Jasim, M.N. POX Controller and Open Flow Performance Evaluation in Software Defined Networks (SDN) Using Mininet Emulator. *IOP Conf. Series Mater. Sci. Eng.* **2020**, *881*, 012102. [CrossRef]
37. Chouhan, R.K.; Atulkar, M.; Nagwani, N.K. Performance Comparison of Ryu and Floodlight Controllers in Different SDN Topologies. In Proceedings of the 2019 1st International Conference on Advanced Technologies in Intelligent Control, Environment, Computing & Communication Engineering (ICATIECE), Bangalore, India, 19–20 March 2019; pp. 188–191. [CrossRef]
38. Xiong, B.; Yang, K.; Zhao, J.; Li, W.; Li, K. Performance evaluation of OpenFlow-based software-defined networks based on queueing model. *Comput. Netw.* **2016**, *102*, 172–185. [CrossRef]
39. McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; Turner, J. OpenFlow. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 69–74. [CrossRef]
40. Gupta, N.; Maashi, M.S.; Tanwar, S.; Badotra, S.; Aljebreen, M.; Bharany, S. A Comparative Study of Software Defined Networking Controllers Using Mininet. *Electronics* **2022**, *11*, 2715. [CrossRef]
41. Tadros, C.N.; Rizk, M.R.M.; Mokhtar, B.M. Software Defined Network-Based Management for Enhanced 5G Network Services. *IEEE Access* **2020**, *8*, 53997–54008. [CrossRef]
42. Jawaharan, R.; Mohan, P.M.; Das, T.; Gurusamy, M. Empirical Evaluation of SDN Controllers Using Mininet/Wireshark and Comparison with Cbench. In Proceedings of the 2018 27th International Conference on Computer Communication and Networks (ICCCN), Hangzhou, China, 30 July–2 August 2018; pp. 1–2. [CrossRef]
43. Zhu, L.; Karim, M.M.; Sharif, K.; Li, F.; Du, X.; Guizani, M. SDN Controllers: Benchmarking & Performance Evaluation. *arXiv* **2019**, arXiv:1902.04491.