Review

A Qualitative and Comparative Performance Assessment of Logically Centralized SDN Controllers by Mininet Emulator

Mohammad Nowsin Amin Sheikh *, I Shyan Hwang *, Muhammad Saibtain Raza, Mohammad Syuhaimi Ab-Rahman

Posted Date: 30 January 2024

doi: 10.20944/preprints202401.2116.v1

Keywords: SDN, Distributed OpenFlow-enabled Controllers, Logically Centralized Controllers, System Performance.

Review

A Qualitative and Comparative Performance Assessment of Logically Centralized SDN Controllers by Mininet Emulator

Mohammad Nowsin Amin Sheikh [1], I Shyan Hwang [1,*], Muhammad Saibtain Raza [1] and Mohammed Syuhaimi Ab-Rahman [2]

[1] Department of Computer Science and Engineering, Yuan Ze University, Taoyuan 32003, Taiwan
[2] Electrical and Electronic Engineering Department, Universiti Kebangsaan Malaysia, Bangi Selangor 43600, Malaysia
* Correspondence: ishwang@saturn.yzu.edu.tw

Abstract: Software-defined networking (SDN) divides a traditional network into a programmable data plane and a centralized control plane. This technology is expected to transform traditional network companies that provide flexible and intelligent network operations. The core component utilized to oversee every data plane action is the controller in the control plane. In order to achieve optimal performances, the controller's performance and capabilities are therefore crucial. There are lots of controller research available in existing literature. Nevertheless, no qualitative comparison study of OpenFlow-enabled distributed but logically centralized controllers exists for them. This article also includes a quantitative investigation of the performance of several distributed but logically centralized SDN controllers in custom network scenarios using Mininet, as well as a thorough qualitative comparison of them. More precisely, we give a qualitative evaluation of their attributes and classify and categorize 13 distributed but logically centralized SDN controllers according to their capabilities. Additionally, we offer a comprehensive SDN emulation tool called Mininet-based SDN controller performance assessment in this study. Using six performance metrics—bandwidth, round- trip time, delay, jitter, packet loss, and throughput—this work also assesses five distributed but logically centralized controllers against two custom network scenarios (uniform and non-uniform host distribution). Our investigation shows that the OpenDayLight controller performs the best across all performance criteria, with the exception of delay, packet loss and round trip time, where the Ryu controller excels. In the entire experiment, the HyperFlow and ONOS controller performs the worst in terms of all performance metrics. Finally, we discuss detailed research findings on the performance. These experimental results provide a decision-making guideline when selecting a controller.

Keywords: SDN; distributed openflow-enabled controllers; logically centralized controllers; system performance

1. Introduction

A corporate network is made up of two or more computers that share data, resources, and storage in order to exchange knowledge and save money. Typically, a dedicated de- vice is used to conduct the majority of traditional network activities. A dedicated device for application delivery is referred to as one or more switches, routers, and controllers. The majority of these device features are implemented in specific hardware. ASICs, or application-specific integrated circuits, are frequently used for this. The majority of net- works in use today are still conventional networks. While the amount of everyday data transfer is increasing and bringing more devices, these networks still have some limits. The additional device has to rewire while joining a network. Many of these pieces of equipment are managed by individuals, and as networks develop, there are more parts to keep up with. The management of these large network infrastructures is more challenging. As a result, a fresh approach to network administration is required in order to get beyond these established network limitations [1].

 doi:10.20944/preprints202401.2116.v1

The technology known as "software-defined networking" (SDN) is expected to trans- form current network companies. Numerous business advantages are offered, including improved design reliability, flexibility, and stability. SDN systems eliminate disruptions in traditional networks and simplify network architecture by separating network control from forwarding services. SDN enables programmatic network setup to enhance network administration and satisfy our requirements. The separation of the control plane and data plane, flow-based forwarding choices, and the definition of a flow as a series of packets from one are among the primary characteristics of the SDN architecture [2].

SDN does have certain drawbacks, though. The SDN protocol and controller require significant network infrastructure upgrades. Consequently, the entire network will have been reconstructed. The price of reconfiguration rises as a result. It is crucial to train employees. It is important to make new management tools available and teach everyone how to utilize them. The architecture of centralized SDN controllers prioritizes safety. SDN controllers are alone in charge of all transmission operations since they are frequently seen as single points of failure. If the switch-to-controller or SDN controller connections fail, the entire network will crash. A single control issue, such as a single point of failure, restricted control resources, etc., can be resolved if there are numerous controllers [3].

The fundamental principle of SDN enhances the network in several aspects, including more effective resource use, improved network administration, lower costs, innovation with fresh development, and many more. An effective controller is crucial to managing all these changes and enhancing resource usage for increased network performance. In addition, the controller is in charge of tracking and analyzing data flow in real time. The massive rise of distributed processing-based real-time applications has created a large need for high-performance controllers in networking businesses, data centers, academia, and research. Thus, it is essential to look at the controller's performance in order to provide effective traffic routing, which will increase resource usage for improved performance of the network [4].

The efficiency of SDN controllers has been the subject of several research. There is still a chasm between them, though. There is no in-depth classification scheme for SDN control plane architecture. Perhaps the only categorization criterion that can be used is the deployment architecture. Based on this, Nunes et al. [5] has proposed two significant kinds of control plane systems with multiple controllers. Using Control Plane Architecture as our foundation, we offer an in-depth classification of software-defined networking in this research which is shown in Figure 1. The issues with distributed but logically centralized SDN-based controllers haven't been thoroughly researched in the past. It is quite difficult to predict which controller will operate best in a certain sort of network from the standpoint of actual implementation. Therefore, it is crucial to compare these controllers both qualitatively and quantitatively. As far as we are aware, no such work has been done that compares and assesses the attributes of the distributed but logically centralized SDN- based controllers. A Mininet may simulate several types of network components, including connections, layer-2 switches, layer-3 routers, and hosts. It is based on a single Linux kernel and makes use of virtualization to simulate a whole network on a single machine. As far as we are aware, no study has been done specifically to compare performance in a Mininet setting. The investigation of OpenFlow-based controllers' performance has been done before. Works [18–31] give some quantitative comparisons, although most of them either have a single application or a simple environment where several trials may be conducted. We have selected a custom scenario for this research that utilizes Mininet capabilities. Using the efficient network simulator Mininet for custom network environment, this study provides a thorough analysis of five distributed but logically centralized controllers in terms of bandwidth, round-trip time, delay, jitter, packet loss, and throughput. We have contributed the following to this paper:

- We present a more in-depth classification of SDN control plane architecture, shown in Figure 1;
- We give a qualitative evaluation of their attributes and classify and categorize 13 distributed but logically centralized SDN controllers according to their capabilities;
- A detailed survey of SDN controller performance based on Mininet is made;

 doi:10.20944/preprints202401.2116.v1

- Using six performance metrics—bandwidth, round-trip time, delay, jitter, packet loss, and throughput—this work also assesses five distributed but logically centralized controllers against two custom network scenarios (uniform and non-uniform host distribution).

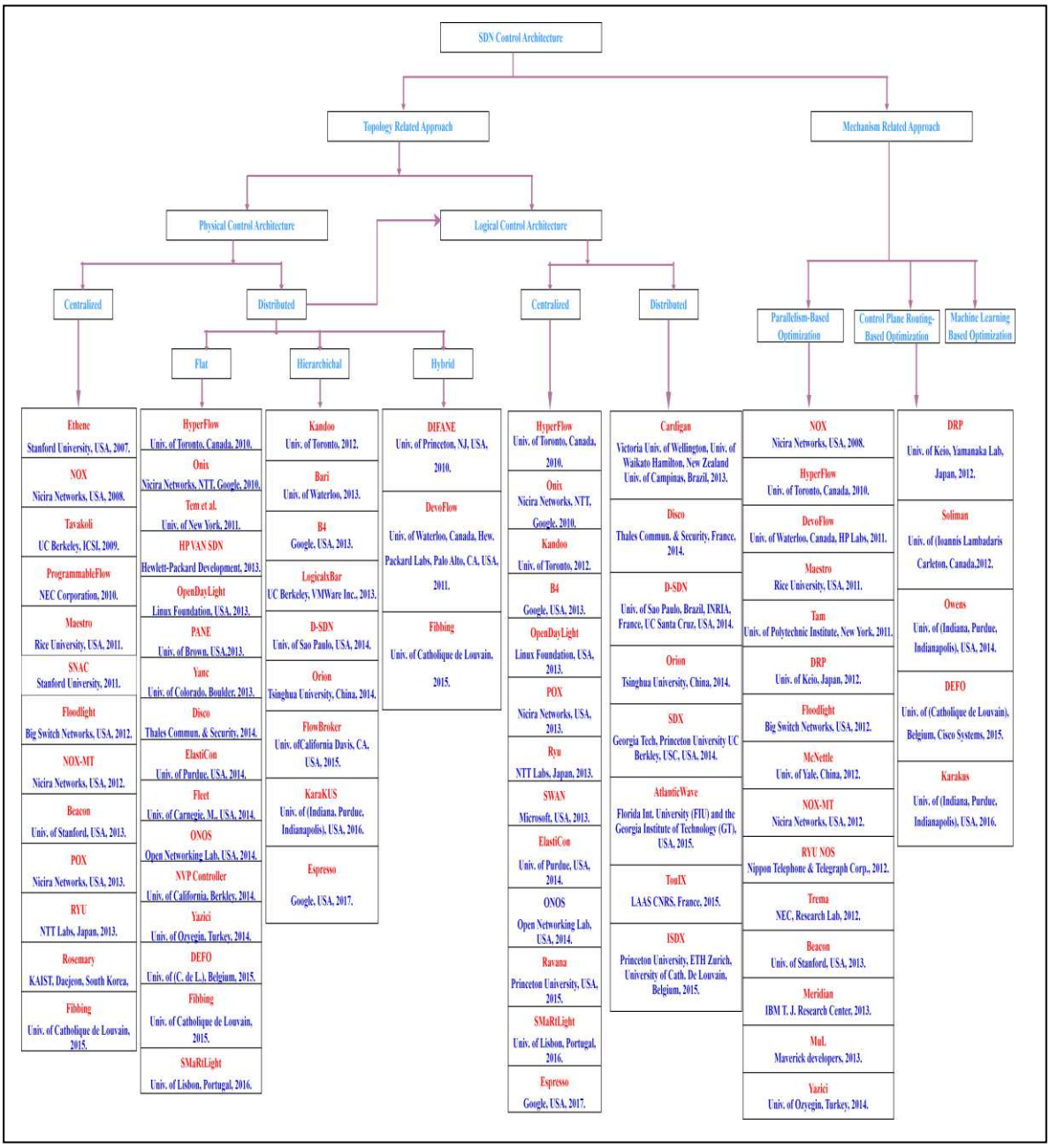


Figure 1. Control Plane Architecture Classification of Software Defined Networking (SDN)[3,6–12].

The remainder of the research paper is structured as follows: Section 2 presents the relevant research and how our work is different from other existing works followed by detailed qualitative evaluation of distributed but logically centralized controllers with design choices and selection criteria along with Mininet-based performance assessment study in Section 3. The qualitative and quantitative description and evaluation of selected 5 controllers, topology description with setup of the experiment and the resources required to evaluate the performance of the controllers are briefly discussed in Section 4. Section 5 presents a performance analysis. Section 6 goes into extensive detail about the outcomes of experiments and studies with discussion of logically centralized controller's advantages and the most likely challenges that can encounter for distributed controllers. Section 7 discusses the conclusion and future work.

 doi:10.20944/preprints202401.2116.v1

2. Related Work

Some relevant works are presented in this area, which is divided into three sections: papers on various control architectures, papers on benchmarking performance assessment, and papers on Mininet performance assessment of SDN controllers.

The SDN control plane is currently created using either single controller designs or distributed/multiple controller architectures [3,6–10]. In their thorough investigation of the scalability problems with SDN-based control planes, Karakus and Durresi [9] provided a classification and taxonomy of the state-of-the-art which is considered the very first taxonomy of control plane. They have simply produced a physical categorization in their work. There were two methods used for categorization: mechanism-related and topological. The topology of various architectures, their relationships, and scalability issues with the design of centralized and distributed controllers were the topics of the topology-related methods. The mechanism-related included many techniques, including parallel-based and routing scheme-based optimization, to improve controllers and their scalability issues. Only machine learning-based optimization was added by Abuarqoub et al., [11] to the mechanism-based approach. If not, the classifications are almost identical. However, both classification lacks the further logical categorization of distributed and centralized SDN control planes. Additionally, a number of distinct controller types are missing from their taxonomy. In addition to the physical categorization, distributed SDN control systems may be divided into conceptually centralized and logically distributed architectures based on how information is shared among controller instances [12] whereas. However, the entire control plane architecture was not provided by this effort, and their taxonomy lacks specific controller types. Additionally, Bannour et al., [10] included the physical classification of control plane architecture along with its various types in their study. However, hybrid orientation, along with logical categorization and overall control plane design, is absent from the subdivision of physically distributed SDN control plane. Isong et al., [13] did not address hybrid orientation, instead explicitly classifying the control plane into physical and logical components. Only in situations when the mechanism-based approach is completely absent did he view the control plane as a topology-related approach. In our work, topology-related and mechanism-related control designs are separated apart. Two categories—physically or logically centralized and physically or logically distributed—are further distinguished amongst these topology-related control structures. Moreover, each control architecture may have a flat, hierarchical, or hybrid orientation. Control plane routing-based optimization and parallelism-based optimization are two categories of mechanism-related approaches. Every category has a unique controller kind.

Several studies have been done in the aim of comparing SDN controllers [2,10,14–37]. The authors of article [10] examined 15 distributed SDN controllers while taking into account the following factors: (1) Scalability; (2) Reliability; (3) Consistency; (4) Interoperability; (5) Monitoring; and (6) Security. All of the aforementioned research have taken into account either open source or widely used SDN-based controllers. Distributed controllers have only been examined by them. However, they did not evaluate and compare the characteristics of the distributed but logically centralized SDN-based controllers. In this study, we have provided a qualitative assessment of their characteristics and, based on their capacities, we classify and categorize thirteen (13) distributed but logically centralized SDN controllers. One of the earliest studies to compare SDN controllers was [25], which only looked at controller performance while taking into account a small number of controllers (NOX-MT, Beacon, and Maestro). But over time, new controllers like POX, Ryu, FloodLight, and OpenDayLight have taken the place of these ones. In [26], a thorough analysis of SDN OpenFlow Controllers was carried out. The efficacy of the commonly used SDN controllers, NOX, POX, Beacon, FloodLight, MuL, Maestro, and Ryu, is compared. The hcprobe tool was utilized by the authors. This comparison needs to be expanded to take into account newly designed controllers as well as additional controller functions. A set of requirements—TLS support, virtualization, open source, interfaces, GUI, RESTful API, productivity, documentation, modularity, platform support, age, OpenFlow support, and OpenStack Neutron support—provided by the study carried out in [27] serve as the foundation for the comparison of the controllers. Analytic Hierarchy Process (AHP), a monotonic interpolation/extrapolation mechanism that transfers the values of the attributes to a value

 doi:10.20944/preprints202401.2116.v1

on a pre-defined scale, was used to modify the Multi-Criteria Decision Making (MCDM) technique for the comparison. Five controllers (POX, Ryu, Trema, FloodLight, and OpenDayLight) were tested using the modified AHP, and "Ryu" was determined to be the best controller based on their needs. Based on how simple it was to: 1) examine the network (discovery), 2) add a new network function (setup), 3) change an existing function (change), and 4) remove a function (removal), the authors of [28] evaluated the qualitative performance of two open source controllers. However, only two SDN controllers (ONOS and OpenDayLight) and one network function (port-/traffic-mirroring) were included in the study. Proactive and reactive operational styles are discussed separately in [29]. Because the rules in the proactive mode are loaded into the switch at startup rather than every time the switch gets a packet for which there is no matching rule in its flow table, the proactive mode performs better than the reactive one. Even though this comparison highlights a significant aspect of the performance comparison, it is insufficient to determine which featured controller is the best. More research is conducted in [30], which offers more factors to take into account while creating a new controller. There are two different designs that are taken into consideration: shared queue with adaptive batching and static partitioning with static batching. The Beacon system, utilizing static batching, demonstrated the best performance in the Maestro, NOX-MT, and FloodLight tests. However, the optimal latency records are shown by Maestro, which employs adaptive batching architecture. Therefore, the behavior of the necessary controller, which is connected to its application domain, determines the architecture to be used. The portability and performance of the controller are shown to be impacted by the programming language selection in [31]. Because Java is cross-platform compatible and allows multithreading, the authors contend that it is the best option. Python has problems with multithreading at the performance level, whereas C++ has memory management and other constraints. The runtime platform determines which net languages work with (Linux compatibility is not supported). As a result, they demonstrate that out of numerous controllers (NOX, POX, Maestro, FloodLight, Ryu), the Java-based Beacon performs the best. Nonetheless, the fact that these several languages are still in use today indicates that each language retains some traits that the others have not taken up. The authors [32], emphasized the problem of software aging. The primary problem under investigation is memory leak, and in order to maintain the study's objectivity, two Java-based controllers—FloodLight and Beacon—were compared. The outcomes demonstrated that Beacon performs better and uses less memory than FloodLight. The authors of [33] and [34] examined 12 whereas authors of [35] examined 8 popular open source controllers taking into account a number of factors, including programming language, graphical user interface, documentation, modularity, distributed/centralized, platform support, southbound and northbound APIs, partners, multithreading support, open stack support, and application area. Furthermore, the comparison takes into account a wide range of other SDN-specific qualitative factors in addition to quantitative metrics like throughput and latency. The study [36] examined a number of open-source controllers, including Ryu, POX, OpenDayLight (ODL), and Open Network Operative System (ONOS), by qualitatively assessing both their performance and features. The authors used the Cbench program to benchmark the controllers under various operating situations and quantitatively assess a number of characteristics. The authors of article [37] compared the features and performances of four well-known SDN controllers. The authors demonstrated that Ryu has a good amount of features, making it perfect for small-scale SDN installations, and that OpenDayLight and ONOS are the controllers with the most feature richness.

A hardware-based testbed or simulation/emulation can be used to assess or benchmark a controller's performance. Hardware testbeds are more expensive for the research community even if they offer measurements that are more in line with real values in a production setting. Thus, assessments based on simulation or emulation are typical. Frameworks like OFLOPS [38], OFLOPS-Turbo [39], Cbench [25], PktBlaster [40], and OFCBenchmark [41] have been suggested to encourage the study of various performance characteristics of OpenFlow devices. In order to test new applications and construct OpenFlow-based networks on a single system, a number of simulation and emulation tools have also been developed. Tools that may be used to deploy a virtual network and estimate performance metrics for different network topologies and sizes is the Mininet [17], NS3

 doi:10.20944/preprints202401.2116.v1

[42] network simulator. Papers [2,14] provide a comparative assessment of previous benchmarking research as well as an in-depth look at the capabilities of benchmarking tools. A summary of SDN-based research utilizing Mininet is provided in Paper [17], yet there is no survey of SDN controller performance evaluation using Mininet. We have conducted a thorough review of the literature on performance evaluation in this study, shown in Table 2. Additionally, this work evaluates five distributed but logically centralized controllers using six performance metrics: bandwidth, round-trip time, delay, jitter, packet loss, and throughput. We created two custom topologies with hosts dispersed in a uniform and non-uniform way using the Mininet simulation environment.

3. Controller Classification and Design Choices

We have conducted a thorough search of proposals in both the academic and commercial domains in order to compare various SDN controllers. Here, we first outline potential controller categorization criteria, then move on to a comparison analysis.

3.1. Selection Criteria

Controllers behave in essentially the same way. After analyzing every controller, we come to the conclusion that the majority of them lack a categorization foundation for their roles, duties, and way of functioning. The deployment architecture may be the only categorization criterion that can be applied [14]. Since the primary goal of SDN was to consolidate the control plane, the majority of controllers only used one controller. Scalability issues and a single point of failure were nevertheless brought about by this. Multiple controllers can operate in a flat or hierarchical structure inside a domain thanks to the distributed design.

3.2. Qualitative Comparison

A thorough overview of the many characteristics of the distributed but logically centralized controllers is provided in Table 1. We will not discuss about each controller separately. Instead, we showcase the characteristics and design options of controllers. The characteristics and design options we have followed is taken by [14].

Table 1. Distributed but Logically Centralized SDN Controller's Feature Comparison Table [14].

Name	P. Language	N. API	S. API	E. API	Supported Platform	Interface	License	Multi-threading	Modul-arity	Consis-tency	Documentation
HyperFlow	C++	REST	OpenFlow 1.0	Publish and subscribe messages	Linux	Web UI	Proprietary	Yes	Fair	No	Limited
Onix	C++	Onix API	OpenFlow 1.0, OVSDB	Zookeeper	Linux	CLI	Apache License 2.0	Yes	Good	No	Limited
Kandoo	C, C++, Python	Java RPC	OpenFlow 1.0-1.2	Messaging Channel	Linux	CLI	Proprietary	Yes	High	No	Limited
B4	C++	BGP	OpenFlow (Version unknown)	Not Specified	Linux	Not Specified	Proprietary	Not Specified	Not Specified	Not Specified	Limited
OpenDay-Light	Java	REST, RESTCONF, XMPP, NETCONF	OpenFlow 1.0, 1.3	Akka, Raft	Linux, MacOS, Windows	CLI, Web UI	EPL 1.0	Yes	High	Yes	Good
POX	Python	ad-hoc	OpenFlow 1.0	Python Script	Linux, MacOS, Windows	CLI, GUI	Apache 2.0	No	Low	No	Limited

 doi:10.20944/preprints202401.2116.v1

Ryu	Python	REST	OpenFlow 1.0, 1.5	Python Script	Linux, MacOS	CLI	Apache 2.0	Yes	Fair	Yes	Good
SWAN	Python	REST	OpenFlow (Version Unknown)	Python Script	Linux, MacOS, Windows	Not Specified	Proprietary	Not Specified	Not Specified	Not Specified	Limited
ElastiCon	Java	Not mentioned	OpenFlow (Version Unknown)	Not Specified	Not mentioned	Not Specified	Proprietary	Not Specified	Not Specified	Not Specified	Limited
ONOS	Java	REST, Neutron	OpenFlow 1.0, 1.3	Raft	Linux, MacOS, Windows	CLI, Web UI	Apache 2.0	Yes	High	Yes	Good
Ravana	Python	OpenFlow	OpenFlow with extensions	Not Specified	Linux	Not Specified	Proprietary	Not Specified	Not Specified	Not Specified	Limited
SMaRtLight	Java	REST	OpenFlow 1.3	BFT-SMaRt	Linux	CLI	Proprietary	Not Specified	Not Specified	No	Limited
Espresso	Java	REST	OpenFlow (Version Unknown)	Not Specified	Linux	Not Specified	Proprietary	Not Specified	Not Specified	Not Specified	Limited

*P.Language= Programming Language, N. API= North Bound API, E. API= East Bound API, OF= OpenFlow, OVSDB= Open vSwitch Database, WUI= Web User Interface, CLI= Command Line Interface.

Programming Language: A variety of programming languages, including C, C++, Java, Java Script, Python, Ruby, Haskell, Go, and Erlang, have been used to write controllers. There are instances where a single language is used to create the complete controller. While the core and modules of many other controllers employ various languages in order to provide effective memory allocation, be executable on different platforms, or—most importantly—achieve superior performance under specific circumstances.

Programmable Interface (API): Generally speaking, Northbound API (NBI) enables the controller to support applications like as intrusion detection, load balancing, network virtualization, flow forwarding, and topology monitoring that rely on network events produced by data plane devices. Conversely, low-level APIs such as Southbound API (SBI) are in charge of facilitating communication between a controller and switches or routers that have SDN enabled. Furthermore, in a dispersed or hierarchical environment, several controllers from various domains build peering relationships with one another through the usage of east-west API (EWBI). Not every controller offers every API, and only a small number of them have been tailored for a particular purpose.

Platform and Interface: These characteristics explain how a controller is implemented to work with a particular operating system. Linux distributions serve as the foundation for the majority of controllers. Furthermore, certain controllers give administrators access to graphical or web-based interfaces for configuring and viewing statistics data.

Threading and Modularity: Lightweight SDN installations are better suited for a single-threaded controller. Multi-threaded controllers, on the other hand, are appropriate for commercial applications including optical networks, SDN-WAN, and 5G. However, the versatility of a controller makes it possible to integrate many features and applications. In a dispersed setting, a controller with high modularity can execute tasks more quickly.

License, Availability, and Documentation: Open Source licensing applies to the majority of the controllers covered in this article. Some, on the other hand, are only accessible by special request or for research purposes due to a proprietary license. Many of these controllers do not receive regular updates since it is difficult for the developers to maintain them on a regular basis. However, the source code is accessible online, enabling anybody to modify it further in accordance with the specifications. We discovered that most of them lack adequate documentation when we accessed them online. Conversely, those that undergo frequent updates come with community-based

 doi:10.20944/preprints202401.2116.v1

assistance as well as comprehensive and up-to-date documentation for every version that is accessible.

3.3. Mininet and Mininet Based Study

A team of experts from Stanford University created Mininet as a tool for research and instruction. With the aid of a controller, Mininet may do testing and mimic SDN networks. We utilize Mininet as a simulation tool, which is a network emulator. It enables the building of networks with different topologies made up of several virtual hosts, connections, and switches. With the help of this program, we can quickly and simply access any network component via the Mininet CLI, create the network to your exact specifications, share it with others, and ultimately develop it with actual hardware. Before implementing our network in real life, we may test and refine it in a virtual environment by using Mininet. A switch and hosts linked to the switch and to each other make up the default Mininet topology. The switch is coupled to an OpenFlow controller. Mininet hosts have the ability to operate on several Linux CLIs. For example, they may simply retrieve the bandwidth between the user and the server by using the iperf command. Although Mininet only offers a few topologies—tree, linear, and single topologies by default—it is possible to design any desired topology by creating a Python script [15,17,45,53,66].

Numerous studies have been conducted utilizing Mininet to assess SDN controller performance. However, a study of previous work on Mininet-based performance evaluation is lacking. Table 2 displays an extensive overview of the current Mininet-based efforts that will be covered in this part.

Table 2. Existing Performance Analysis Study Using Mininet.

Ref	Controller's Evaluated	Topology Used	Evaluation Metrics	Lessons Learned
[4]	Ryu	Single	Bandwidth, Throughput, RTT, Transmission of Data Packet	The Ryu SDN controller is regarded as one of the most effective traffic engineering controllers.
[15]	Ryu, POX, FloodLight, ODL, ONOS	Single, Linear, Tree(Google & Facebook)	Latency, Throughput, Delay, Bandwidth	ODL did better than ONOS among the selected distributed controllers, whereas Ryu did best among the selected centralized controllers.
[17]	ODL, Ryu	Custom	Latency	ODL performs best.
[19]	ODL, Ryu	Internet and Bridge	Topology Discovery Time, Delay, Throughput, CPU Utilization	ODL outperforms Ryu for both the Internet and Brite topologies.
[29]	FloodLight, NOX, POX, Trema	-	Proactive, Reactive	While the reactive technique lowers controller performance but needs less configuration, the proactive controller receives fewer request messages.
[43]	POX	-	Bandwidth	Prior to self-learning, the execution switch component has more bandwidth between hosts.
[44]	Ryu	Linear	Latency	The controller receives more requests as the network grows and is unable to respond to them all in a timely manner.
[45]	FloodLight	Linear, Tree	Time to Create and Destroy the Virtual Networks, Memory Usage	When the number of virtual networks increases, the Mininet takes longer to construct and delete virtual networks and uses more memory.
[46]	ONOS	Custom	Latency, Topology Discovery Time	Although just a small number of ONOS apps have been put into use, more work still has to be done.
[47]	POX, Ryu, ONOS, ODL	Tree	RTT, Bandwidth	While ONOS performs better in Switch mode, Hub mode performance is nearly same.
[48]	POX	Linear	RTT, Delay, Bandwidth, Throughput, Mean Data Rate	Hub components outperform Switch components in terms of performance.
[49]	POX, FloodLight	Tree	Scalability	The impact of a simulation environment on the time needed to construct a topology is noteworthy.
[50]	FloodLight, ODL	Single, Linear, Tree	Latency and Packet Loss	In terms of latency, ODL performs better than FloodLight in networks with low load as well as for tree topologies in networks with medium load. In terms of packet loss for tree topologies and latency for linear topologies, FloodLight can beat ODL in densely trafficked networks.
[51]	FloodLight, Beacon, Open-MUL, Open-IRIS	TCP, UDP and ICMP Traffic	Time of the First Packet, RTT, Transfer Time, Packet Loss,	Using QoS in the OF network improves FloodLight controller performance.
[52]	POX, Ryu and Pyretic	Star	RTT, Latency, Throughput	Ryu outperforms Pyretic and POX in terms of speed.
[53]	OF reference Controller	Single, Linear, Tree	Bandwidth Utilization, Packet Transmission Rate, Round-Trip Propagation Delay, Throughput	The performance of a single topology network is superior than that of a linear topology network, where an increase in hosts would inevitably result in a decrease in network performance. This is not the case with a tree topology network, where load is spread.
[54]	POX and Ryu	Combination of (Linear & Tree), DCN Tree, Mesh	Bit-rate, Delay, Packet-rate, and Jitter	POX performs better in layer 1 switching scenarios due to its superior traffic management capabilities. When it comes to layer 2 switching, Ryu produced far higher performance outcomes.

 doi:10.20944/preprints202401.2116.v1

[55]	ODL and ONOS	Tree	Cluster Failure Recovery Time	In terms of GUI, clusters, link-up, switch-up, and throughput, ONOS performs well. For stability and topology finding, ODL performs better.
[56]	OF- reference Controller	Custom	Bandwidth, Throughput, Jitter, Packet Loss	A centrally controlled OF network operates similarly to a regular network, except that the data plane and control plane are seperated.
[57]	POX, FloodLight	Star, Linear, Tree	Delay, Throughput	The FloodLight Controller performs better than the POX.
[58]	FloodLight	Mesh	Throughput and Latency	The network is subject to load as the number of nodes linked to switches rises.
[59]	FloodLight, ODL	Tree, Single, Linear, Torus	Throughput, Data Transfer Rate and Latency	ODL performs better in a single topology while FloodLight works better in linear, tree, and torus topologies.
[60]	OVS- Controller, POX, FloodLight, ODL	Single, Linear, Tree	Latency, Bandwidth Utilization, Jitter, Packet Loss	In linear topologies, FloodLight Controller regularly delivers severe data loss, whereas ODL Controller is unable to manage the load provided by it.
[61]	OF reference controller	Custom	Throughput, RTT, Delay, Jitter	Delay and Jitter analysis guarantees improved traffic flow, RTT analysis shows a convenient and reliable network, and high network throughput are QoS parameters.
[62]	Libfluid, ONOS, ODL, POX and Ryu	Linear	Throughput, Delay	As the number of switches and hosts rose, throughput values declined and latency values climbed.
[63]	POX, FloodLight	Single, Linear, Tree, Custom	Throughput and Round-Trip Delay	In comparison to the POX controller, the FloodLight controller offers more effective performance. Controllers with fewer features are more appropriate for activities involving configuration. For activities that are performance-based, feature-based controllers work well.
[64]	POX, Ryu	Single, Linear, Tree, Dumbbel, DCN, SAT	Throughput, Latency	Ryu has superior performance.
[65]	Ryu	Mesh	Throughput	Ryu is an extremely resource-demanding controller that maximizes CPU and RAM use, which causes performance to deteriorate as number of nodes rises.
[66]	ONOS, Open MUL, POX	Linear	Latency, Throughput, Topology Discovery Time	The performance evaluation is considerably underestimated by Cbench.
[67]	ONOS	Mesh	Throughput	When there are varying numbers of nodes, ONOS acts steadily and can partially withstand a network explosion.
[68]	NOX	Custom	Throughput, Response Time	Compared to ROIA and Multiple Packet Schedular, NOX performs better.
[69]	NOX, FloodLight	Custom	Throughput, Response Time	Internal controller NOX is inferior to the FloodLight controller.
[70]	Ryu, POX and Pyretic	Tree	RTT	Compared to Ryu and POX controllers, Pyretic controller performs better with Software Defined Networking.
[71]	POX, FloodLight, ODL	Tree, Mesh	RTT, Throughput	POX outperforms FloodLight and ODL in terms of RTT and throughput.
[72]	ODL, Ryu	Tree	Throughput, Switchover-Time	Compared to Mininet simulation, utilizing a hardware test-bed experiment offers a greater and more consistent throughput.
[73]	Ryu	Tree	Throughput, Switchover-time	Number of flow entries within the data plane can be reduced by using MFT.
[74]	Ryu, FloodLight, ODL, ONOS	Linear, Tree, Mesh	Delay, Throughput	In terms of throughput and latency, the FloodLight controller performs better than Ryu, ODL and ONOS.
[75]	Beacon	Mesh	Throughput	When the number of nodes increases, throughput decreases.
[76]	ODL, ONOS	Tree	Burst Rate, Throughput, RTT and Bandwidth	The ODL controller outperforms the ONOS in terms of bandwidth, round trip time (RTT), throughput, and burst rate.
[77]	NOX, POX, ONOS, Ryu	Custom	Throughput, Response Time	In terms of response time, POX is superior, whereas in terms of throughput, ONOS is superior.
[78]	Ryu, POX, ONOS, FloodLight	Linear	Delay, Jitter and Throughput.	FloodLight performs the best.
[79]	ODL, Ryu	Linear	Throughput, Delay, Packet Loss, Resource Utilization	Resource Utilization tests revealed that the ODL controller performed better than the Ryu controller.
[80]	POX	Linear	Bandwidth, CPU Load, Packet Loss, Latency, Throughput	When 20 to 30 switches are used, there is an effective bandwidth usage when the bandwidth begins at 100 Mbps and increases progressively to 500 Mbps.
[81]	FloodLight, Ryu	Single, Minimal, Linear, Tree, Reverse, Custom	Bandwidth, Latency	The FloodLight controller performs better than the Ryu controller because it has a larger bandwidth and less latency.
[82]	Ryu	Single	Bandwidth, Throughput, RTT, Jitter, Packet Loss	Ryu is a great option for research and small commercial applications because it is written in Python.
[83]	ONOS, FloodLight, Ryu	Leaf Spine DCN	Throughput, Topology Discovery Time	The ONOS controller that performed the poorest in network topology discovery time and the best in the flow test.
[84]	FloodLight, POX, NOX	Linear, Tree, Custom	Throughput, Bandwidth, Packet Loss, Latency, Topology Discovery Time	FloodLight controller is the fastest in all topologies.
[85]	POX, Ryu	Mesh	Throughput, Delay, Jitter, Packet Loss	In terms of throughput, packet loss, delay, and jitter, the Ryu controller performs better than the POX controller.
[86]	Ryu, ODL, FloodLight, Beacon, IRIS, ONOS, POX	Tree	Throughput, Jitter, Latency, and Stability	An increase of hosts or switches has an impact on performance.
[87]	FloodLight	Single, Linear, Tree	Throughput, RTT	By rerouting traffic to alternate connection pathways with a greater RTT and lower throughput, SDN can handle link failure circumstances.
[88]	NOX, ONOS, FloodLight, ODL, POX, Ryu	Custom	Throughput, Response Time	Increasing number of operations impact throughput and response time.
[89]	POX	Tree, Bus, Star	Bandwidth Utilization, Jitter and Packet Loss	The number of open switches has a significant role in the star topology, which has the maximum TCP bandwidth and the lowest UDP loss.
[90]	ODL	Custom	Delay, Throughput, Jitter, Packet Drop, Bitrate, Bytes Received.	Multi-controller networks are more dependable and achieve high availability.

 doi:10.20944/preprints202401.2116.v1

[91]	POX, Ryu	Custom	Jitter, Packet Loss, Throughput, Packet Delivery	The POX controller offers superior throughput results. The Ryu controller functions better in terms of packet delivery ratio, jitter, and packet loss.
[92]	POX, FloodLight	Single, Linear, Tree, Custom	Throughput, Round-Trip Delay	FloodLight controller performs better than POX.

A number of papers demonstrated how to create SDN architecture for network traffic analysis using the open-source Ryu SDN controller. The suggested study assessed how well SDN architecture performed for performance metrics including bandwidth, throughput, jitter, packet loss roundtrip time, switchover time, etc. based on various network topologies. Because Ryu is written in Python, the studies revealed that it is an excellent choice for research and small commercial applications [4,44,65,73,82]. Conversely, the OpenFlow interface is implemented in different network simulation scenario with open source controller POX for network traffic analysis in papers [43,48,80,89]. Analysis and evaluation were done on things like packet loss, CPU load, bandwidth utilization etc. According to those studies, efficient use of bandwidth occurs when it begins at 100 Mbps and increases gradually to 500 Mbps, particularly when there are 20 to 30 switches in use. The major reason for the given POX controller's low reliability was dropped packets. A number of researchers have explored the creation and execution of the desired SDN environment scenario utilizing FloodLight [45,58,87], OpenDayLight [90], ONOS [46,67], NOX [68] and Beacon controllers [75]. Using Mininet Simulator, analysis and assessment were performed on several aspects such as throughput, latency, packet loss, jitter, bandwidth usage, etc. We used Mininet to investigate a number of SDN controllers in our preliminary research [68,69,77,88]. We took into consideration a custom topology network situation. Response time and throughput were the performance parameters we looked at. Researchers have made comparisons between the POX and Ryu controllers in several research [54,64,85,91]. Because of its greater traffic management capabilities, POX performs better in layer 1 switching circumstances. Regarding layer 2 switching, Ryu yielded significantly better performance results. The authors of papers [52,70] contrasted the Ryu and POX controllers with Pyretic, another Python-based controller. According to these research, Pyretic controller works better in tree topology while Ryu performs better in star topology. However, in comparison to other Java-based controllers like FloodLight or ODL, the performance of the Java-based controllers was better than that of the POX or Ryu controllers. Not every topology has the same circumstances. Python-based controllers also function better when there are fewer nodes or operations, whereas Java-based controllers perform worse when there are more nodes or operations. Due to their built-in Java files, Java-based controllers have the drawback of requiring a significant amount of memory space to operate [17,19,49,57,63,72,78,79,81,84,92]. In papers [53,56,61], the authors examined the performance study of an OpenFlow network with single, linear, and tree network topologies; however, the authors employed a Mininet reference controller as a point of comparison rather than the POX or Ryu controller. In order to undertake the performance study, all network topologies were compared based on the following metrics: maximum throughput achieved, round-trip propagation time between end nodes, packet transmission rate, and capacity usage. All OpenFlow-enabled topologies were designed with Mininet, a prototype network emulator. The effectiveness of proactive and reactive paradigms in many well-known controllers was examined in [29]. Both an emulator (Cbench and Mininet) and a real environment were used to assess performance. The reactive and proactive modes of operation of several SDN controllers (NOX, POX, Trema, and FloodLight) were compared by Fernandez et al. The results for every controller that was analyzed indicated that proactive mode is when the controller performs at its best. Using a Mininet emulator, Stancu et al. evaluated the four SDN controllers—POX, Ryu, ONOS, and OpenDayLight [47]. The controllers were told to function as a basic L2 learning switch and hub. A tree topology was employed for comparison, and two tests were run in each phase: an iperf command and a ping command between the two end hosts. In a number of studies, researchers have compared the FloodLight and ODL controllers to other controllers [15,50,51,59,60,71,74,86]. According to the research, FloodLight functions better in linear, tree, and torus topologies, but ODL performs better in a single topology. Furthermore, ODL outperforms FloodLight in terms of latency for tree topologies in networks with medium load as well as low load networks. In networks with high traffic volumes, FloodLight can outperform ODL in

 doi:10.20944/preprints202401.2116.v1

terms of packet loss for tree topologies and latency for linear topologies. Researchers have contrasted the ONOS controller with other controllers in a variety of studies [55,62,66,76,83]. The research indicates that ONOS performs well in terms of GUI, clusters, link-up, switch-up, and throughput. However, latency values increased and throughput values decreased as the number of switches and hosts increased. Additionally, the ONOS controller performs best in the flow test and lowest in the network topology discovery time.

In this study, we have provided a Mininet-based SDN controller performance evaluation research, a complete SDN emulation tool. This paper evaluates two custom network scenarios where the hosts are dispersed in a uniform and non-uniform way using the Mininet simulation environment and five distributed but conceptually centralized controllers using six performance metrics: bandwidth, round-trip time, delay, jitter, packet loss, and throughput.

4. Topology Description and Quantitative Evaluation of Controllers

The performance of five distinct distributed yet logically centralized controllers is covered in this section. To the best of our knowledge, this problem has not been addressed in any prior research. The controllers Ryu, ODL, ONOS, POX, and HyperFlow are examined. Out of the thirteen controllers that were previously mentioned, these were chosen because (1) the source code for the controller is available, (2) they are compatible with the latest versions of Linux, (3) they can be interfaced with Mininet tools, and (4) the community finds them interesting.

4.1. Selected Controllers and their Qualitative Evaluation

We will give a brief description of the chosen controllers in this part, and Table 3 will show the qualitative comparison of the controllers.

HyperFlow: The HyperFlow application was developed to provide a logically centralized multi-controller architecture at the top of the NOX controller. The three levels of the HyperFlow network are forwarding, control, and application. The control layer has several NOX controllers. A switch links the forwarding layer to the nearby control device. In the case of a problem (failure), a switch could be allocated to another controller. To transport data across the controller, HyperFlow employs a publish/subscribe messaging architecture [91].

Table 2. Qualitative Analysis of the Selected Controllers.

Name / Features	HyperFlow	OpenDayLight	POX	Ryu	ONOS
Programming Language	C++	Java	Python	Python	Java
North Bound API	REST	REST, RESTCONF, XMPP, NETCONF	Adhoc	REST	REST, Neutron
South Bound API	OpenFlow 1.0	OpenFlow 1.0, 1.3	OpenFlow 1.0	OpenFlow 1.0, 1.5.	OpenFlow 1.0, 1.3
East Bound API	Publish and subscribe messages	Akka, Raft	Python Script	Python Script	Raft
Supported Platform	Linux	Linux, MacOS, Windows	Linux, MacOS, Windows	Linux, MacOS	Linux, MacOS, Windows
Interface	Web UI	CLI, Web UI	CLI, GUI	CLI	CLI, Web UI
License	Proprietary	EPL 1.0	Apache 2.0	Apache 2.0	Apache 2.0
Multi-threading	Yes	Yes	No	Yes	Yes
Modularity	Fair	High	Low	Fair	High
Consistency	No	Yes	No	Yes	Yes
Documentation	Limited	Good	Limited	Good	Good
Application Area	Data Centre, SD-WAN, IoT, Cloud Networking	Data Centre, Enterprise Network, Research & Education	Research, Education and Learning, SDN Application Development	Campus, Research, SDN application development, NFV, Network monitoring & security	Data centre, Carrier-Grade Network, Research, SDN/NFV integration

 doi:10.20944/preprints202401.2116.v1

ONOS: The community behind the ONOS (Open Network Operating System) project is open source. The goal of this project is to create an SDN operating system [46]. The Java packages for the ONOS project are loaded into the Karaf OSGi container.

Ryu: An open SDN controller called Ryu Controller was created to improve network agility. It is a fully Python-written, component-based software defined networking framework. In Japanese, the term "Ryu" signifies "flow". NTTT, or Nippon Telegraph and Telephone Corporation, provides assistance for the Ryu Controller. Ryu controller supports a number of protocols, including NETCONF, OF-config, OpenFlow, and others [4,65].

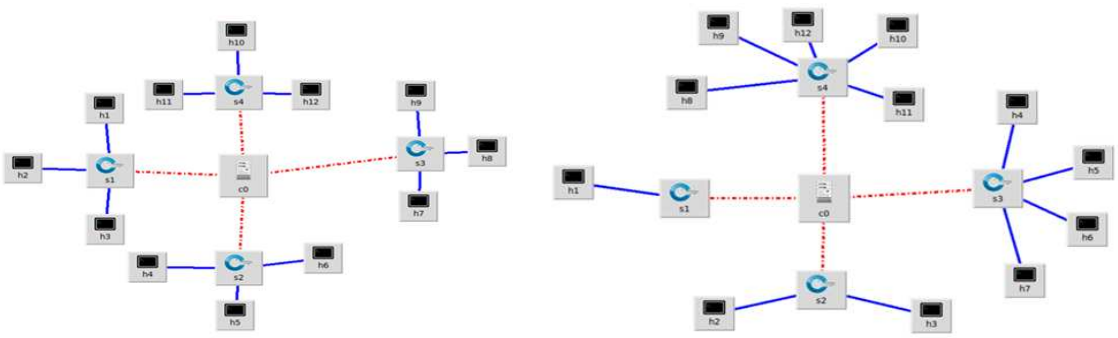
POX: The younger sibling of NOX is a networking software platform called POX (Pythonic Network Operating System). Python is the programming language used in the development of POX [12]. When creating networking software, it can be useful. POX is compatible with several operating systems, including Linux, Mac OS, and Windows. It is compatible with Python 2.7 and lower versions. POX connects with OpenFlow 1.0 switches and has specific support for Open vSwitch/Nicira extensions [43,57,64,71,89,92].

OpenDayLight: A collaborative open source project developed in Java, The OpenDayLight (ODL) Project is hosted by The Linux Foundation [11]. A bidirectional REST and OSGi framework may be programmed using OpenDayLight, and it also supports many non-OpenFlow southbound protocols [50,71,76,79]. There is a wiki specifically for developers, along with many email lists and a source code repository for controller releases, for those who are interested.

4.2. Topology Description

Network topology changes are the most crucial testing characteristic in this experimental research that categorizes all controller tests. The effectiveness of network is significantly influenced by its topology. An efficient network layout may greatly boost performance by lowering energy consumption and boosting data transmission speeds. Any organization's actual network is not restricted to simple topologies like linear or tree. The real network topologies of these firms are complex and resemble hierarchical or graph connections more. The depths of the switches and nodes, as well as the fan out, differ. With custom topology, users may create whatever topology they choose. Here, it is clear that each switch's fan out varies, which makes it impossible to describe it during topology creation for topologies like single, linear, tree, etc. These factors led us to use customized topology for our experiment. The customized architecture is depicted in Figure 2, where four switches and twelve hosts are used for the experiment. Figure 2(a) represents a uniform distribution of hosts to the switch, whereas Figure 2(b) shows a non-uniform distribution of hosts to the switch.

Twelve hosts and four OpenFlow-capable switches are utilized in both topologies of this experiment. Each switch has three hosts connected to it in a uniform host distribution. In contrast to a non-uniform host distribution, switch 1 connects one host, switch 2 connects two hosts, switch 3 connects four hosts, and Switch 4 connects five hosts. In every experiment, controller performance is compared using the same topologies and environment.



(**a**) **Uniform Host Distribution.** (**b**) **Non-Uniform Host Distribution.**

Figure 2. Custom Topology.

 doi:10.20944/preprints202401.2116.v1

4.3. Experiment Setup

The test bed consists of a single PC with an Intel Core i7 CPU operating at 2.90GHz and 8GB of RAM. Windows 10 is the host operating system while Ubuntu 22.10 LTS is the guest operating system on the machine. On a 100Mbit network, congested windows of up to 32Mbytes are employed.

In addition to connecting to the controller and creating network connectivity, Mininet also creates switches, hosts, and linkages. Following a successful connection establishment using Mininet, we used Ubuntu xterm to retrieve client-server interfaces and ran the iperf test to create traffic between hosts in order to assess important factors including bandwidth, throughput, round trip time, delay, jitter, and packet loss.

5. Performance Analysis

This section presents the implementation and usage inside a framework for testing and evaluating experimental attempts to assess distributed (logically centralized) SDN networking experience using five distinct controllers (OpenDayLight, ONOS, POX, Ryu, and HyperFlow).

5.1. Bandwidth

The amount of network bandwidth in Gb/s that has been used the most is shown by the bandwidth in Figure 3. According to our investigation, the bandwidth for OpenDayLight, POX, Ryu, ONOS, and HyperFlow controllers in a uniform host distribution is 62.09, 59.1 and 59.2, 58.4 and 58.8 Gb/s, respectively; the value for a non-uniform host distribution is 61.8, 59, 59.3, 56.8 and 57.8 Gb/s. In both situations, the OpenDayLight controller performs the best and the Ryu controller the worst.

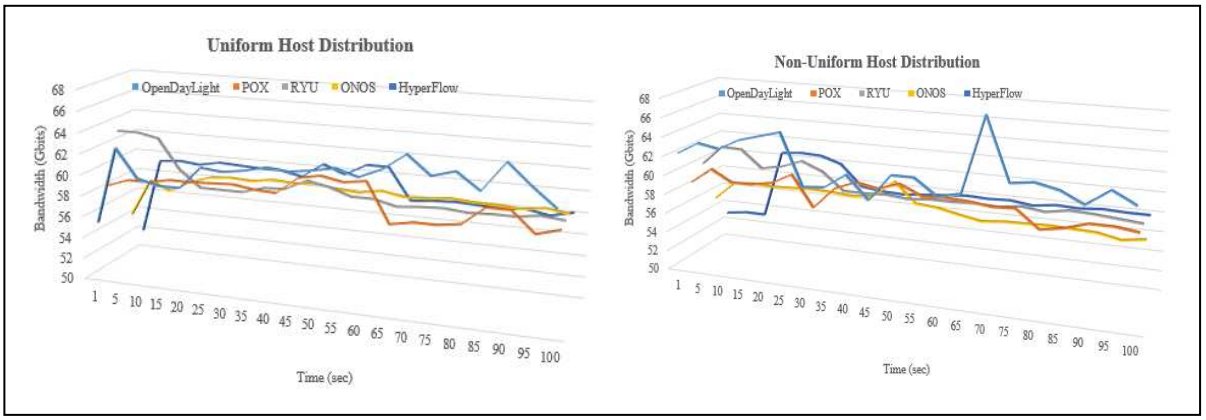


Figure 3. Bandwidth.

5.2. Round Trip Time

The round trip time is the entire period of time that is it takes for a data packet to travel from where it started to its endpoint and for its confirmation to be received at the origin. In Figure 4, the round trip time between a network and server may be found using the ping command. It is measured in milliseconds. OpenDayLight, POX, Ryu, ONOS, and HyperFlow controller have average round trip times for uniform host distribution of 0.1014, 0.1244, 0.0927, 0.1403 and 0.3298 ms, respectively. For non-uniform host distribution, the round trip times are 0.1028, 0.1388, 0.0945, 0.1228, and 0.3416 ms, respectively. The POX controller has the fastest average round trip time in both scenarios, whereas the HyperFlow controller has the slowest average.

 doi:10.20944/preprints202401.2116.v1

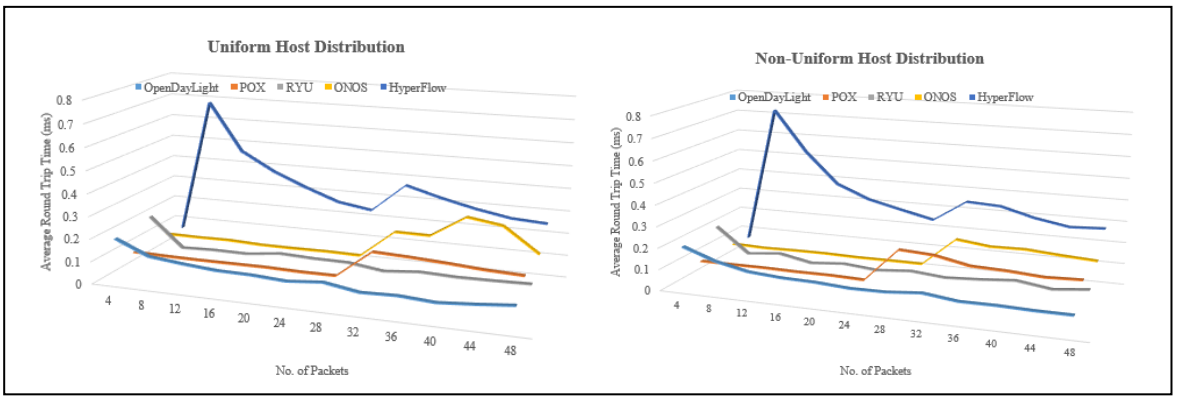


Figure 4. Round Trip Time.

5.3. Delay

The amount of time it requires for a data packet to go from a particular location to another is known as latency or delay. Milliseconds are used to measure it. According to Figure 5, the OpenDayLight, ONOS, Ryu, POX, and HyperFlow controllers' respective delays or latency for uniform host distribution are 0.0702, 0.0790, 0.0668, 0.0764, and 0.0760 ms; for non-uniform host distribution, the delays are 0.0689, 0.0816, 0.0653, 0.0719, and 0.0785ms. Both times, POX controller and OpenDayLight controller had extremely similar values, with POX controller being marginally superior to OpenDayLight controller. For both situations, ONOS has experienced the highest delay.

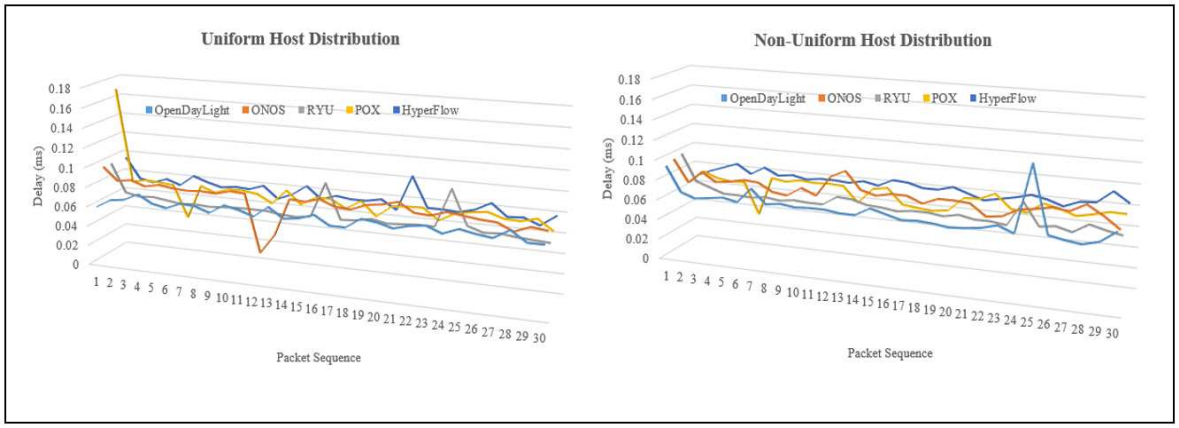


Figure 5. Delay.

5.4. Jitter

Delays in the normal interval between data packet sequences would be the simplest cause of jitter. Jitter is the variation in the data packet delay periods that are received. It is easiest to identify jitter by PINGing a faraway device with a series of packets and then determining the average time difference between each return packet sequence. There is just one type of measurement, time-based and based on milliseconds. The jitter for the OpenDayLight, Ryu, POX, HyperFlow and ONOS controllers is shown in Figure 6 and is, respectively, 0.0055, 0.0072, 0.0082, 0.0132, and 0.0099ms for uniform host distribution and 0.0046, 0.0053, 0.0075, 0.0147 and 0.0118ms for non-uniform host distribution. In all scenarios, Ryu controller has the worst jitter whereas OpenDayLight surpasses all other controllers.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 January 2024 doi:10.20944/preprints202401.2116.v1

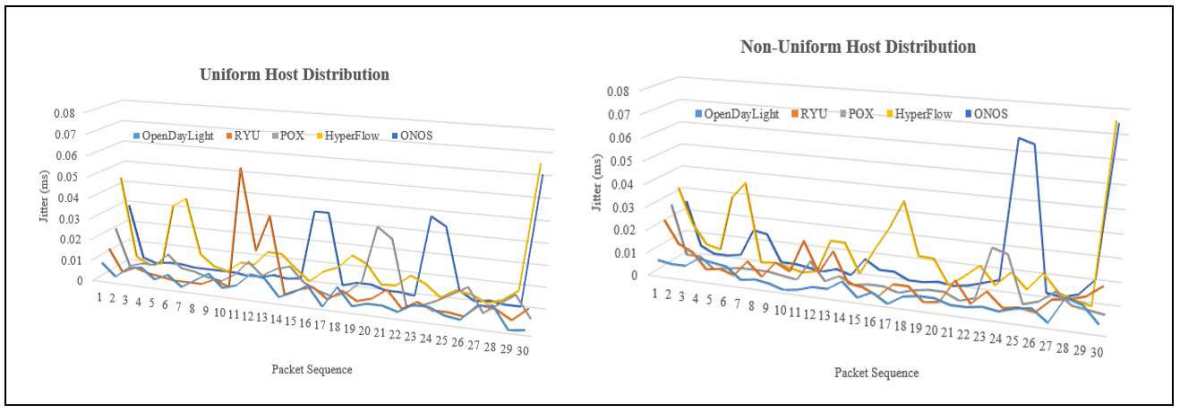


Figure 6. Jitter.

5.5. Packet Loss

Packet loss can be determined and expressed as a percentage based on the difference between the received packets and the transmitted packets. Figure 7 illustrates packet loss for Ryu, ONOS, POX, OpenDayLight, and HyperFlow controllers for uniform host distribution. For non-uniform host distribution, the corresponding values are 0.065, 14.5, 20.5, 12.5, and 28.5 percentages where for uniform distribution, the values are 0.05, 12.5, 20.5, 8.5, and 22.5 percentages. Ryu experiences the least packet loss in both circumstances, whereas HyperFlow experiences the most.

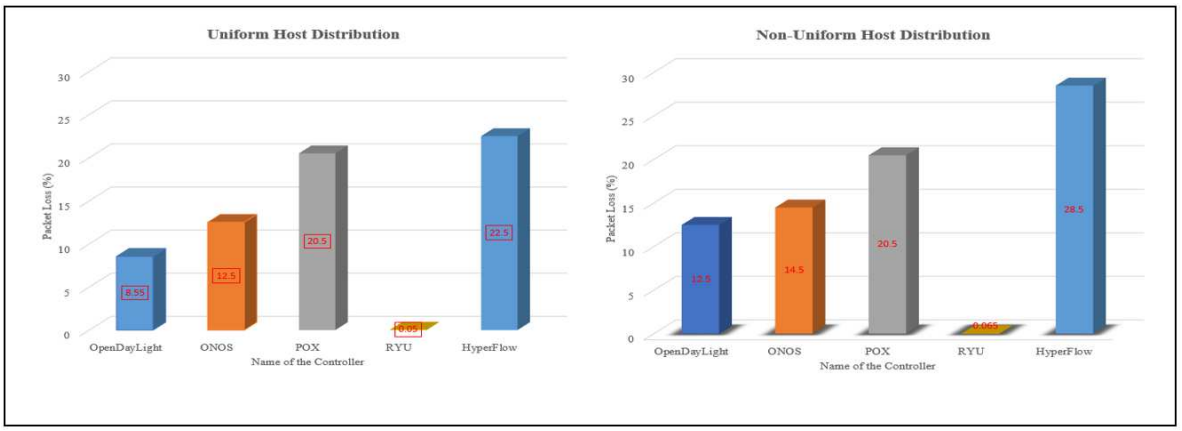


Figure 7. Packet Loss.

5.6. Throughput

Throughput is the maximum quantity of data that may be sent from a source to a destination in a predetermined period of time. Throughput, or payload over time, quantifies the largest burst of data transfer, to put it another way. GB/s are used to measure it. In our experiment, the term "time" refers to a duration chosen by the examiner (100 seconds). Our research shows that the throughput for the OpenDayLight, ONOS, POX, Ryu, and HyperFlow controllers in a uniform host distribution is 7.23, 6.88, 6.89, 6.62, and 6.77 GB/s, respectively, while the values for a non-uniform host distribution are 7.2, 6.87, 6.9, 6.59, and 6.83 GBytes/sec are shown in Figure 8. In both situations, Ryu has the lowest throughput while OpenDayLight has the greatest.

 doi:10.20944/preprints202401.2116.v1

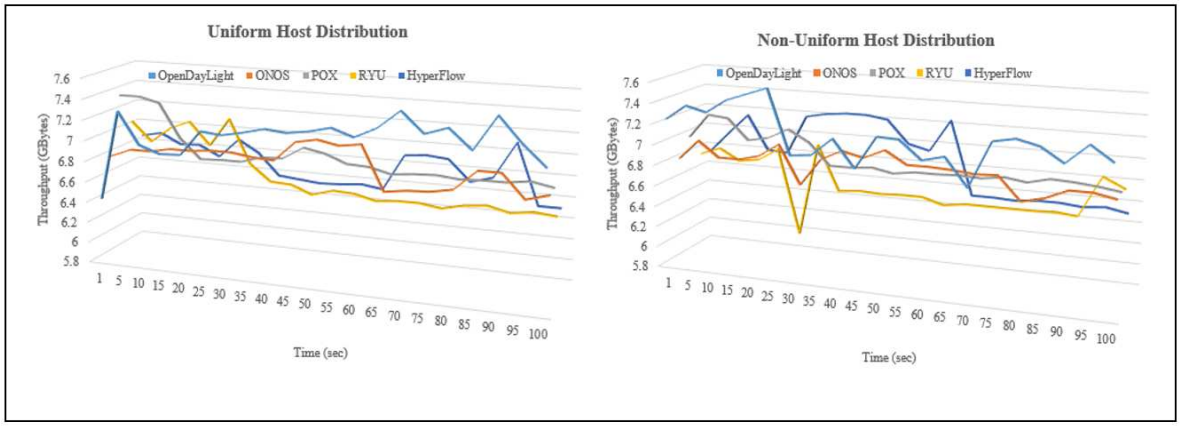


Figure 8. Throughput.

6. Discussion

This part covers the performance study of the different distributed (but logically centralized) controllers discussed in the preceding section. Bandwidth, throughput, round trip time, delay, jitter, and packet loss are the six metrics used for analysis. According to the thorough analysis, the OpenDayLight controller performs better than the Ryu controller in all performance parameters except than delay, packet loss and round trip time, which are the Ryu controller's strong points. In terms of all performance metrics, the ONOS and HyperFlow controller performs the worst during the whole trial.

The OpenDayLight controller surpasses the POX controller by 1.00% in terms of bandwidth consideration in a uniform host distribution, the Ryu controller by 0.97%, the ONOS controller by 1.24%, and the HyperFlow controller by 1.11%, according to the findings of our experiment. The OpenDayLight controller, on the other hand, surpasses the POX controller, the Ryu controller, the ONOS controller, and the HyperFlow controller by 0.95%, 0.85%, 1.69%, and 1.36%, respectively in non-uniform host distribution. With a uniform host distribution for delay considerations, Ryu controller outperforms in comparison to the OpenDayLight controller by 0.91%, the ONOS controller by 3.30%, the POX controller by 2.61%, and the HyperFlow controller by 2.50%. However, in terms of non-uniform host distribution, the Ryu controller outperforms the OpenDayLight controller, the ONOS controller, the POX controller, and the HyperFlow controller by 0.36%, 1.63%, 0.67%, and 1.32%, respectively. Regarding jitter consideration in a uniform host distribution, the OpenDayLight controller outperforms the Ryu controller by 3.87%, the POX controller by 6.26%, the HyperFlow controller by 17.57%, and the ONOS controller by 10.15%. On the other side, the OpenDayLight controller outperforms the Ryu controller, the POX controller, the HyperFlow controller, and the ONOS controller by 1.57%, 6.49%, 22.92%, and 16.30%, respectively, for non-uniform host distribution. The Ryu controller outperforms the OpenDayLight controller by 1.11%, the POX controller by 4.03%, the ONOS controller by 6.04%, and the HyperFlow controller by 24.02% when round trip time factors are taken into account. However, the Ryu controller surpasses the OpenDayLight controller, the POX controller, the ONOS controller, and the HyperFlow controller by 1.03%, 5.52%, 3.52%, and 30.86%, respectively, in terms of non-uniform host distribution. The OpenDayLight controller surpasses the ONOS, POX, Ryu, and HyperFlow controllers in terms of throughput consideration in a uniform host distribution by 1.02%, 0.99%, 1.77%, and 1.34%, respectively. In contrast, the OpenDayLight controller performs better than the ONOS controller, the POX controller, the Ryu controller, and the HyperFlow controller for non-uniform host distribution by 0.96%, 0.87%, 1.77%, and 1.08%, respectively. The Ryu controller beats the ONOS controller by 12.45%, the POX controller by 20.45%, the OpenDayLight controller by 8.45%, and the HyperFlow controller by 22.45% with regard to packet loss consideration in a uniform host distribution. On the other hand, with non-uniform host distribution, the Ryu controller performs better than the ONOS controller, the POX controller, the OpenDayLight controller, and the HyperFlow controller by 14.435%, 20.435%, 12.435%, and 28.435%, respectively. The summery is shown in Table 3.

 doi:10.20944/preprints202401.2116.v1

Table 3. Feature based Performance Comparison with respect to other controllers in percentage.

	Bandwidth								Jitter								Throughput							
	Uniform				Non-uniform				Uniform				Non-uniform				Uniform				Non-uniform			
OpenDayLight	POX	Ryu	ONOS	HyperFlow	POX	Ryu	ONOS	HyperFlow	POX	Ryu	ONOS	HyperFlow	POX	Ryu	ONOS	HyperFlow	POX	Ryu	ONOS	HyperFlow	POX	Ryu	ONOS	HyperFlow
	1.00	0.97	1.24	1.11	0.95	0.85	1.69	1.36	6.26	3.87	10.15	17.57	6.49	1.57	16.30	22.92	0.99	1.77	1.02	1.34	0.87	1.77	0.96	1.08
	Delay								**Round Trip Time**								**Packet Loss**							
	Uniform				Non-uniform				Uniform				Non-uniform				Uniform				Non-uniform			
Ryu	ODL	ONOS	POX	HyperFlow	ODL	ONOS	POX	HyperFlow	ODL	ONOS	POX	HyperFlow	ODL	ONOS	POX	HyperFlow	ODL	ONOS	POX	HyperFlow	ODL	ONOS	POX	HyperFlow
	0.91	3.30	2.61	2.50	0.36	1.63	0.67	1.32	1.11	6.04	4.03	24.02	1.03	3.52	5.52	30.86	8.45	12.4	20.4	22.4	12.4	14.4	20.4	28.4

A variety of factors can have an impact on the SDN controller's performance. When the number of actions or switches varies or rises, as well as the amount of stress that is applied, it heavily depends on the selection of controllers. We recommend using the OpenDayLight or Ryu controller when selecting the best distributed (logically centralized) controller. The Ryu controller was recommended by the authors of papers [4,52,64,82,85,91] for their research because of its simplicity of use and Python-based scripting. Additionally, using OpenDayLight controller has been recommended by the authors of papers [17,19,55,76,79]. However, paper [15] is the most pertinent work that fits in with our findings. When choosing distributed controllers, ODL should be used, whereas Ryu should be used for centralized controller selection.

6.1. Topology Description

Some advantages of distributed (logically centralized) controllers include the following:

- **Higher-level policies:** Rather than using network IDs, language used to describe policies is based on principles.
- **Paths should be determined by policy:** Based on policy, the controller should choose the pathways.
- **Fine-grained control:** The data plane keeps a per-flow state while the controller manages the initial packet in a flow

6.2. Challenges of Logically Centralized Controllers

The distributed (logically centralized) SDN controller has problems with interoperability, stability, controller location, and other things. The challenges are discussed below:

Global Consistency: Domain-specific controllers deal with traffic congestion and problems particular to their domains in the distributed SDN control plane. Wherever possible, all other controller instances within a cluster should get critical updates as soon as possible. Maintaining a steady and high-caliber performance over time may be challenging, but it is not impossible [94]. At

 doi:10.20944/preprints202401.2116.v1

greater synchronization and overhead rates, strong consistency ensures that all dispersed controllers have access to the same network data. According to Levin et al. [95], control plane consistency can significantly affect the efficacy of a network. To keep a consistent overall view for all controllers, rational trade-offs between rules enforcement and performance are required.

Reliability: For fault tolerance, centralized SDN management uses a simple master/slave architecture. In order to maintain a consistent, logically centralized global picture, network state information must be divided amongst various controllers under distributed SDN control that communicate specifics. In a distributed SDN control plane, there should be coordinated approaches for resolving issues, obtaining simultaneous updates, and maintaining a constant network state [3]. On large-scale networks, the burden can be spread among the other active controllers using a rapid, self-healing method. The maintenance of a sizable amount of state overhead is required by this method, as is the division of domain state among the participating controllers. In the paper Jyotish et al. [96], a performance metric based on dependability, availability, and security is described.

Automatic Reconfiguration: Mapping between forwarding devices and distributed controllers should be automatic rather than using static settings. Static deployments could cause an uneven load distribution among cluster controllers. In order to monitor and communicate load information with adjacent controller scenarios, applications need to be implemented on all active SDN controllers and switches connected to different SDN controllers. However, this approach could overwhelm the controller with load-sharing data, raising scalability issues [94]. Without a consistent northbound or eastward interface, it is also impossible to communicate between apps and device mobility.

Interoperability: Interoperability is essential for the development and deployment of SDN in next-generation networks. This requires compatibility between various SDN controllers operating in various domains and utilizing various technologies. Interoperability is difficult since each SDN controller has a distinct data model and lacks a standard east-west interface. YANG [10], is an open-source data modeling language that enables the standardization and automation of data representations. This NETCONF-based IETF contribution is anticipated to enable SDN interoperability.

Network segmentation: Depending on the topology, distributed SDN controls may cause latency-sensitive applications (monitoring) or compute-intensive applications (route calculation) to suffer. When latency-sensitive and compute-intensive applications are co-located in the same controller, the authors of Chang et al. [97] found it challenging to achieve quick response and convergence times.

The recommended separation of several applications makes use of the slicing that is now available to lessen inter-controller communication. Functional slicing and communication-aware control applications can help to speed up network convergence and response times. To enhance the performance of network partitioning, the scientific community has to do more research.

Problems with load balancing and controller deployment: The SDN controller's integration with the forwarding apparatus generates queries concerning the best locations for controller deployment and the required number of controllers in the network. Such challenges must be overcome, especially in WANs where propagation delay is a major consideration [3]. In other situations, such data centers or businesses (enterprise), load balancing and fault tolerance receive more attention from researchers. The distributed SDN control architecture is scalable rather than being controlled centrally. However, in order to achieve scalability and maintain good performance, it is necessary to physically install the controller as well as use several SDN controllers [9,14].

Security: Security can be compromised throughout the SDN network due to the dispersed SDN controllers' complete network intelligence. The SDN may lose access to the control plane if the safety problem is not resolved on a big scale. The distributed control strategy reduces the risk of authentication and message integrity issues in comparison to a single central controller. Without adequate authentication, an attacker may easily join their network node, making it behave like other SDN controller instances and damaging the whole network. Information exchange between controllers must be safe in order to give unified views of distributed SDN controllers throughout the

whole network. To solve these problems in a distributed SDN controller environment, new methodologies and security norms are required [21,22].

5. Conclusions

A thorough analysis of SDN controllers, their classification is summarized. Testing of many OpenFlow-bases distributed (but logically centralized) SDN controllers, including OpenDayLight, ONOS, POX, Ryu, and HyperFlow, are all included in this experimental research. The topologies used for the experiments were custom generated for them. The six parameters that are examined are bandwidth, round trip time, delay, jitter, packet loss, and throughput. The amount of load that can affect a controller's performance, the number of activities that increase or decrease, and both have been considered. Our analysis reveals that the OpenDayLight controller outperforms the Ryu controller in all performance metrics except latency, packet loss and round trip time, which are strong suits for the Ryu controller. The ONOS and HyperFlow controllers perform the worst in every performance metric consideration. Based on the aforementioned statistics, we advise using OpenDayLight or Ryu controllers as the best performing distributed (logically centralized) controllers. For greater experience, OpenDayLight can be used else Ryu controller should be used where everything is Python language dependent. More research is necessary to conclusively support our hypothesis. To assess the efficacy of more distributed (logically centralized) controllers, we plan to apply diverse loads and scenarios in future. In future, we want to create an SD-IoT architecture and evaluate various DDoS attacks prior to the network. After that, we'll isolate the data from the traffic and attack and analyze it using machine learning or deep learning models.

Author Contributions: Conceptualization, M.N.A.S, I.-S.H; methodology, M.N.A.S and M.S.R; software, M.N.A.S and M.S.R; validation, M.N.A.S, M.S.R; formal analysis, M.N.A.S and M.S.R; investigation, M.N.A.S; resources, M.N.A.S.; data curation, M.N.A.S; writing—original draft preparation, M.N.A.S and M.S.R; writing—review and editing, M.S.A.R; visualization, M.N.A.S and M.S.R; supervision, I.-S.H. and M.S.A.R; project administration, I.-S.H. and M.S.A.R; funding acquisition, I.-S.H. and M.S.A.R. All authors have read and agreed to the published version of the manuscript.

Funding: This project was supported by NSTC 112-2221-E-155-009, Taiwan.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Acknowledgments: We wish to acknowledge the anonymous referees who gave precious suggestions to improve the work.

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

References

1. Herrera, J.G.; Botero, J.F. Resource allocation in NFV: A comprehensive survey. IEEE Transactions on Network and Service Management 2016, 13, 518–532. doi: 10.1109/TNSM.2016.2598420.
2. Zhu, L., Karim, M.M., Sharif, K., Li, F., Du, X. and Guizani, M. SDN controllers: Benchmarking & performance evaluation. arXiv preprint, 2019, pp. 1-14, *arXiv:1902.04491*.
3. Zhang, Y.; Cui, L.; Wang, W.; Zhang, Y. A survey on software defined networking with multiple controllers. Journal of Network and Computer Applications 2018, 103, 101–118. doi: https://doi.org/10.1016/j.jnca.2017.11.015.
4. Bhardwaj, S.; Panda, S.N. Performance evaluation using Ryu SDN controller in software- defined networking environment. Wireless Personal Communications 2022, 122, 701–723. doi: https://doi.org/10.1007/s11277-021-08920-3.
5. Santos, M.A.; Nunes, B.A.; Obraczka, K.; Turletti, T.; De Oliveira, B.T.; Margi, C.B. Decentralizing SDN's control plane. In Proceedings of the 39th Annual IEEE Conference on Local Computer Networks, 2014, pp. 402–405. doi: 10.1109/LCN.2014.6925802.
6. Khorramizadeh, M.; Ahmadi, V. Capacity and load-aware software-defined network controller placement in heterogeneous environments. Computer Communications 2018, 129, 226–247. doi: https://doi.org/10.1016/j.comcom.2018.07.037.
7. Sahoo, K.S.; Puthal, D.; Obaidat, M.S.; Sarkar, A.; Mishra, S.K.; Sahoo, B. On the placement of controllers in software-defined-WAN using meta-heuristic approach. Journal of Systems and Software 2018, 145, 180–194. doi: https://doi.org/10.1016/j.jss.2018.05.032.

8. Jalili, A.; Keshtgari, M.; Akbari, R.; Javidan, R. Multi criteria analysis of controller placement problem in software defined networks. Computer Communications 2019, 133, 115–128. doi: https://doi.org/10.1016/j.comcom.2018.08.003.
9. Karakus, M.; Durresi, A. A survey: Control plane scalability issues and approaches in software- defined networking (SDN). Computer Networks 2017, 112, 279–293. doi: https://doi.org/10.1016/j.comnet.2016.11.017.
10. Bannour, F.; Souihi, S.; Mellouk, A. Distributed SDN control: Survey, taxonomy, and challenges. IEEE Communications Surveys & Tutorials 2017, 20, 333–354. doi: 10.1109/COMST.2017.2782482.
11. Abuarqoub, A. A review of the control plane scalability approaches in software defined networking. Future Internet 2020, 12, 49. doi: https://doi.org/10.3390/fi12030049.
12. Benamrane, F.; Benaini, R.; et al. Performances of OpenFlow-based software-defined networks: an overview. Journal of Networks 2015, 10, 329. doi: 0002059787; 10.4304/jnw.10.6.329-337.
13. 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. doi: 10.1109/ACCESS.2020.3023974.
14. Zhu, L.; Karim, M.M.; Sharif, K.; Xu, C.; Li, F.; Du, X.; Guizani, M. SDN controllers: A comprehensive analysis and performance evaluation study. ACM Computing Surveys (CSUR) 2020, 53, 1–40. doi: https://doi.org/10.1145/3421764.
15. Mostafavi, S.; Hakami, V.; Paydar, F. Performance evaluation of software-defined networking controllers: a comparative study. Computer and Knowledge Engineering 2020, 2, 63–73.
16. Elmoslemany, M.M.; Eldien, A.S.T.; Selim, M.M. Performance Analysis in Software Defined Network Controllers. In Proceedings of the 2020 15th International Conference on Computer Engineering and Systems (ICCES), 2020, pp. 1–6.
17. 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.
18. Keshari, S.K.; Kansal, V.; Kumar, S. A systematic review of quality of services (QoS) in software defined networking (SDN). Wireless Personal Communications 2021, 116, 2593–2614.
19. Ali, J.; Roh, B.h.; Lee, S. QoS improvement with an optimum controller selection for software- defined networks. Plos one 2019, 14, e0217631.
20. Shirvar, A.; Goswami, B. Performance comparison of software-defined network controllers. In Proceedings of the 2021 International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT), 2021, pp. 1–13.
21. Bhat, D.; Rao, K.; Latha, N. A survey on software-defined networking concepts and architecture. International Journal of Science, Technology & Management. Volume 2015.
22. Xia, W.; Wen, Y.; Foh, C.H.; Niyato, D.; Xie, H. A survey on software-defined networking. IEEE Communications Surveys & Tutorials 2014, 17, 27–51.
23. Amin, R.; Reisslein, M.; Shah, N. Hybrid SDN networks: A survey of existing approaches. IEEE Communications Surveys & Tutorials 2018, 20, 3259–3306.
24. Islam, S.; Khan, M.A.I.; Shorno, S.T.; Sarker, S.; Siddik, M.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), 2019, pp. 1–5.
25. Tootoonchian, A.; Gorbunov, S.; Ganjali, Y.; Casado, M.; Sherwood, R. On Controller Performance in Software-Defined Networks. In Proceedings of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE 12), 2012, pp. 1–6.
26. Shalimov, A.; Zuikov, D.; Zimarina, D.; Pashkov, V.; Smeliansky, R. Advanced study of SDN/OpenFlow controllers. In Proceedings of the Proceedings of the 9th central & eastern european software engineering conference in russia, 2013, pp. 1–6.
27. Khondoker, R.; Zaalouk, A.; Marx, R.; Bayarou, K. Feature-based comparison and selection of Software Defined Networking (SDN) controllers. In Proceedings of the 2014 world congress on computer applications and information systems (WCCAIS), 2014, pp. 1–7.
28. Bondkovskii, A.; Keeney, J.; van der Meer, S.; Weber, S. Qualitative comparison of open-source sdn controllers. In Proceedings of the NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium, 2016, pp. 889–894.
29. Fernandez, M.P. Comparing openflow controller paradigms scalability: Reactive and proactive. In Proceedings of the 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), 2013, pp. 1009–1016.
30. Shah, S.A.; Faiz, J.; Farooq, M.; Shafi, A.; Mehdi, S.A. An architectural evaluation of SDN controllers. In Proceedings of the 2013 IEEE international conference on communications (ICC), 2013, pp. 3504–3508.
31. Erickson, D. The beacon openflow controller. In Proceedings of the Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, 2013, pp. 13–18.
32. Alencar, F.; Santos, M.; Santana, M.; Fernandes, S. How Software Aging affects SDN: A view on the controllers. In Proceedings of the 2014 Global Information Infrastructure and Networking Symposium (GIIS), 2014, pp. 1–6.

33. Salman, O.; Elhajj, I.H.; Kayssi, A.; Chehab, A. SDN controllers: A comparative study. In Proceedings of the 2016 18th mediterranean electrotechnical conference (MELECON), 2016, pp. 1–6.
34. Paliwal, M.; Shrimankar, D.; Tembhurne, O. Controllers in SDN: A review report. IEEE Access 2018, 6, pp. 36256–36270.
35. Sakellaropoulou, D. A qualitative study of SDN controllers. Athens University of Economics and Business 2017.
36. Bispo, P.; Corujo, D.; Aguiar, R.L. A qualitative and quantitative assessment of SDN controllers. In Proceedings of the 2017 International Young Engineers Forum (YEF-ECE), 2017, pp. 6–11.
37. Mamushiane, L.; Lysko, A.; Dlamini, S. A comparative evaluation of the performance of popular SDN controllers. In Proceedings of the 2018 Wireless Days (WD), 2018, pp. 54–59.
38. Rotsos, C.; Sarrar, N.; Uhlig, S.; Sherwood, R.; Moore, A.W. OFLOPS: An open framework for OpenFlow switch evaluation. In Proceedings of the Passive and Active Measurement: 13th International Conference, PAM 2012, Vienna, Austria, March 12-14th, 2012. Proceedings 13. Springer, 2012, pp. 85–95.
39. Rotsos, C.; Antichi, G.; Bruyere, M.; Owezarski, P.; Moore, A.W. An open testing framework for next-generation OpenFlow switches. In Proceedings of the 2014 Third European Workshop on Software Defined Networks, 2014, pp. 127–128.
40. Veryx Technologies, ''PktBlaster SDN controller test.'' [Online]. Available: *http://sdn.veryxtech.com/*. Hossa
41. Jarschel, M.; Lehrieder, F.; Magyari, Z.; Pries, R. A flexible OpenFlow-controller benchmark. In Proceedings of the 2012 European Workshop on Software Defined Networking, 2012, pp. 48–53.
42. Carneiro, G. J., "NS3."
43. Prete, L.R.; Shinoda, A.A.; Schweitzer, C.M.; De Oliveira, R.L.S. Simulation in an SDN network scenario using the POX Controller. In Proceedings of the IEEE Colombian Conference on Communications and Computing (COLCOM), 2014, pp. 1–6.
44. Benamrane, F.; Mamoun, M.B.; Benaini, R. Short: A case study of the performance of an openflow controller. In Proceedings of the International Conference on Networked Systems. Springer, 2014, pp. 330–334.
45. De Oliveira, R.L.S.; Schweitzer, C.M.; Shinoda, A.A.; Prete, L.R. Using Mininet for emulation and prototyping software-defined networks. In Proceedings of the IEEE Colombian conference on communications and computing (COLCOM), 2014, pp. 1–6.
46. Berde, P.; Gerola, M.; Hart, J.; Higuchi, Y.; Kobayashi, M.; Koide, T.; Lantz, B.; O'Connor, B.; Radoslavov, P.; Snow, W.; et al. ONOS: towards an open, distributed SDN OS. In Proceedings of the Proceedings of the third workshop on Hot topics in software defined networking, 2014, pp. 1–6.
47. Stancu, A.L.; Halunga, S.; Vulpe, A.; Suciu, G.; Fratu, O.; Popovici, E.C. A comparison between several software defined networking controllers. In Proceedings of the 2015 12th international conference on telecommunication in modern satellite, cable and broadcasting services (TELSIKS), 2015, pp. 223–226.
48. Ramadona, S.; Hidayatulloh, B.A.; Siswanto, D.F.; Syambas, N. The simulation of SDN network using POX controller: Case in Politeknik Caltex Riau. In Proceedings of the 2015 9th International Conference on Telecommunication Systems Services and Applications (TSSA), 2015, pp. 1–6.
49. Keti, F.; Askar, S. Emulation of software defined networks using Mininet in different simulation environments. In Proceedings of the 2015 6th International Conference on Intelligent Systems, Modelling and Simulation, 2015, pp. 205–210.
50. Rowshanrad, S.; Abdi, V.; Keshtgari, M. Performance evaluation of SDN controllers: FloodLight and OpenDayLight. IIUM Engineering Journal 2016, 17, 47–57.
51. Jasim, A.; Hamid, D. Enhancing the performance of OpenFlow network by using QoS. International Journal of Scientific & Engineering Research (IJSER) 2016, 7, 950–955.
52. Kaur, K.; Kaur, S.; Gupta, V. Performance analysis of python based openflow controllers 2016, pp. 1–4.
53. Bholebawa, I.Z.; Dalal, U.D. Design and performance analysis of OpenFlow-enabled network topologies using Mininet. International Journal of Computer and Communication Engineering 2016, 5, 419-429.
54. 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), 2016, pp. 1–6.
55. Yamei, F.; Qing, L.; Qi, H. Research and comparative analysis of performance test on SDN controller. In Proceedings of the 2016 first IEEE international conference on computer communication and the internet (ICCCI), 2016, pp. 207–210.
56. Bholebawa, I.Z.; Jha, R.K.; Dalal, U.D. Performance analysis of proposed OpenFlow-based network architecture using Mininet. Wireless Personal Communications 2016, 86, 943–958.
57. 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), 2017, pp. 695–699.
58. Asadollahi, S.; Goswami, B.; Raoufy, A.S.; Domingos, H.G.J. Scalability of software defined network on FloodLight controller using OFNet. In Proceedings of the 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), 2017, pp. 1–5.

59. J., P.; Raj, P. Topology-based analysis of performance evaluation of centralized vs. distributed SDN controller. In Proceedings of the 2018 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC), 2018, pp. 1–8.
60. Mittal, S. Performance evaluation of openflow SDN controllers. In Proceedings of the Intelligent Systems Design and Applications: 17th International Conference on Intelligent Systems Design and Applications (ISDA 2017) held in Delhi, India, December 14-16, 2017. Springer, 2018, pp. 913–923.
61. Jany, M.H.R.; Islam, N.; Khondoker, R.; Habib, M.A. Performance analysis of OpenFlow based software defined wired and wireless network. In Proceedings of the 2017 20th International Conference of Computer and Information Technology (ICCIT), 2017, pp. 1–6.
62. Abdullah, M.Z.; Al-Awad, N.A.; Hussein, F.W. Performance Comparison and Evaluation of Different Software Defined Networks Controllers. International Journal of Computing and Network Technology 2018, 6, 1–7.
63. Bholebawa, I.Z.; Dalal, U.D. Performance analysis of SDN/OpenFlow controllers: POX versus FloodLight. Wireless Personal Communications 2018, 98, 1679–1699.
64. Ali, J.; Lee, S.; Roh, B.h. Performance analysis of POX and Ryu with different SDN topologies. In Proceedings of the Proceedings of the 1st International Conference on Information Science and Systems, 2018, pp. 244–249.
65. Asadollahi, S.; Goswami, B.; Sameer, M. Ryu controller's scalability experiment on software defined networks. In Proceedings of the 2018 IEEE international conference on current trends in advanced computing (ICCTAC), 2018, pp. 1–5.
66. 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), 2018, pp. 1–2.
67. Sameer, M.; Goswami, B. Experimenting with ONOS scalability on software defined network. Journal of Advanced Research in Dynamical and Control Systems 2018, 10, 1820–1830.
68. Hossain, M.A.; Sheikh, M.N.A.; Rahman, S.S.; Biswas, S.; Arman, M.A.I. Enhancing and measuring the performance in software defined networking. International Journal of Computer Networks & Communications (IJCNC) 2018, 10, 27–39.
69. Sheikh, M.N.A.; Halder, M.; Kabir, S.S.; Mohalder, R.N. Performance Evaluation on Software Defined Networking through External Controller FloodLight and Internal Controller NOX. International Journal of Scientific Engineering Research, 2018, 9, 1753–1758.
70. Shamim, S.; Shisir, S.; Hasan, A.; Hasan, M.; Hossain, A. Performance analysis of different open flow based controller over software defined networking. Global Journal of Computer Science and Technology 2018, 18, 11–15.
71. Altangerel, G.; Chuluuntsetseg, T.; Yamkhin, D. Performance analysis of SDN controllers: POX, FloodLight and OpenDayLight. arXiv preprint arXiv:2112.10387 2021.
72. Baskoro, F.; Hidayat, R.; Wibowo, S.B. Comparing LACP implementation between Ryu and OpenDayLight SDN controller. In Proceedings of the 2019 11th International Conference on Information Technology and Electrical Engineering (ICITEE), 2019, pp. 1–4.
73. Baskoro, F.; Hidayat, R.; Wibowo, S.B. LACP Experiment using Multiple Flow Table in Ryu SDN Controller. In Proceedings of the 2019 2nd International Conference on Applied Information Technology and Innovation (ICAITI), 2019, pp. 51–55. *https://doi.org/10.1109/ICAITI48442.2019.8982149*.
74. Arahunashi, A.K.; Neethu, S.; Ravish Aradhya, H.V. Performance Analysis of Various SDN Controllers in Mininet Emulator. In Proceedings of the 2019 4th International Conference on Recent Trends on Electronics, Information, Communication Technology (RTEICT), 2019, pp. 752–756. *https://doi.org/10.1109/RTEICT46194.2019.9016693*.
75. Manuel, T.; Goswami, B. Experimenting with scalability of beacon controller in software defined network. In Proceedings of the International Journal of Recent Technology and Engineering, 2019, pp. 550–555.
76. Badotra, S., P.S. Evaluation and comparison of OpenDayLight and open networking operating system in software-defined networking. In Proceedings of the Cluster Computing, 2019, pp. 1281–1291. doi: *https://doi.org/https://doi.org/10.1007/s10586-019-02996-0*.
77. Sheikh, Mohammad Nowsin Amin, Halder, Monishanker, Kabir, Sk. Shalaudin, Miah, Md. Wasim, Khatun, Sawrnali. SDN-Based approach to evaluate the best controller: internal controller NOX and external controllers POX, ONOS, Ryu. Global Journal of Computer Science and Technology 2019, 19, 21–32.
78. Islam, S.; Islam Khan, M.A.; Tasnim Shorno, S.; Sarker, S.; Siddik, M.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), 2019, pp. 1–5. *https://doi.org/10.1109/ICASERT.2019.8934553*.
79. Pramudita, A.; Suartana, I. Perbandingan Performa Controller OpenDayLight dan Ryu pada Arsitektur Software Defined Network. In Proceedings of the Journal of Informatics and Computer Science (JINACS), 2020, pp. 174–178.

80. Noman, H.; Jasim, M. POX controller and open flow performance evaluation in software defined networks (sdn) using Mininet emulator. In Proceedings of the In IOP conference series: materials science and engineering, 2020, pp. 1–9.
81. Li, Y.; Guo, X.; Pang, X.; Peng, B.; Li, X.; Zhang, P. Performance Analysis of FloodLight and Ryu SDN Controllers under Mininet Simulator. In Proceedings of the 2020 IEEE/CIC International Conference on Communications in China (ICCC Workshops), 2020, pp. 85–90. doi: https://doi.org/10.1109/ICCCWorkshops49972.2020.9209935.
82. Islam, M.T., I.N.; Refat, M. Node to node performance evaluation through Ryu SDN controller. In Proceedings of the Wireless Personal Communication, 2020, 112, 555–570.
83. Silva, J.B., S.F.N.E.L.M.; Neto, A. Benchmarking of mainstream SDN controllers over open off-the-shelf software-switches. In Proceedings of the Internet Technology Letters, 2020, pp. 1–6.
84. Eltaj, A.; Hassan, A. Performance Evaluation of SDN Controllers: FloodLight, POX and NOX. In Proceedings of the International Journal of Engineering and Applied Sciences (IJEAS), 2020, 7, 16–43.
85. Abidin, N.Z.; Fiade, A.; Arini.; Aripiyanto, S.; Nuryasin.; Handayani, V. Performance Analysis of POX and Ryu Controller on Software Defined Network with Spanning Tree Protocol. In Proceedings of the 2021 9th International Conference on Cyber and IT Service Management (CITSM), 2021, pp. 1–5. doi: https://doi.org/10.1109/CITSM52892.2021.9588867.
86. Lunagariya, D.; Goswami, B. A Comparative Performance Analysis of Stellar SDN Controllers using Emulators. In Proceedings of the 2021 International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT), 2021, pp. 1–9. doi: https://doi.org/10.1109/ICAECT49130.2021.9392391.
87. Daha, M.Y.; Zahid, M.S.M.; Husain, K.; Ousta, F. Performance Evaluation of Software Defined Networks with Single and Multiple Link Failure Scenario under FloodLight Controller. In Proceedings of the 2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), 2021, pp. 959–965. doi: https://doi.org/10.1109/ICCCIS51004.2021.9397125.
88. Sheikh, M.N.A.; Hwang, I.S.; Ganesan, E.; Kharga, R. Performance Assessment for different SDN-Based Controllers. In Proceedings of the 2021 30th Wireless and Optical Communications Conference (WOCC), 2021, pp. 24–25. doi: https://doi.org/10.1109/WOCC53213.2021.9603050.
89. Bhattacharyya, A.; P, P. Performance Evaluation of Various Topological Software Defined Networks Using Mininet Simulator and POX Controller. In Proceedings of the 2022 Second International Conference on Next Generation Intelligent Systems (ICNGIS), 2022, pp. 1–5. doi: https://doi.org/10.1109/ICNGIS54955.2022.10079779.
90. Elmoslemany, M.M., T.E.A.; Selim, M. Performance Analysis in Software Defined Network (SDN) Multi-Controllers. In Proceedings of the Delta University Scientific Journal, 2023, pp. 181–192.
91. Naim, N., I.M.H.M.A.M.K.S.; Khan, A. POX and Ryu Controller Performance Analysis on Software Defined Network. In Proceedings of the EAI Endorsed Transactions on Internet of Things, 2023, p. e5.
92. Bholebawa, I.; Dalal, U. Performance analysis of SDN/OpenFlow controllers: POX versus FloodLight. In Proceedings of the Wireless Personal Communications, 2018, pp. 1679–1699.
93. Ganjali, Y. and Tootoonchian, A. Hyperflow: A distributed control plane for openflow. In Proceedings of the 2010 internet network management conference on Research on enterprise networking (INM/WREN'10), 2010, pp. 1–3.
94. Ahmad, S. and Mir, A.H. SDN interfaces: protocols, taxonomy and challenges. International Journal of Wireless and Microwave Technologies, 2022, 2, 11–32. doi: 10.5815/ijwmt.2022.02.02.
95. Levin, D., Wundsam, A., Heller, B., Handigol, N. and Feldmann, A. Logically Centralized? State Distribution Tradeoffs in Software Defined Networks. In Proceedings of the Workshop on Hot Topics in Software Defined Networks (HotSDN '12) at ACM SIGCOMM 2012, Finland, 2010, pp. 1–6. 2012, doi: 10.1145/2342441.2342443.
96. Jyotish, N.K., Singh, L.K. and Kumar, C. A state-of-the-art review on performance measurement petri net models for safety critical systems of NPP. Annals of Nuclear Energy. 2022, 165, pp. 1–16, doi: https://doi.org/10.1016/j.anucene.2021.108635.
97. Chang, Y., Rezaei, A., Vamanan, B., Hasan, J., Rao, S. and Vijaykumar, T.N. Hydra: Leveraging Functional Slicing for Efficient Distributed SDN Controllers. In Proceedings of the 2017 9th IEEE International Conference on Communication Systems and Networks (COMSNETS), 2017, pp. 251–258, doi: 10.48550/arXiv.1609.07192.