computers

Review

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

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

[1] Department of Computer Science and Engineering, Yuan Ze University, Taoyuan 32003, Taiwan; s1116042@mail.yzu.edu.tw

[2] Electrical and Electronic Engineering Department, Universiti Kebangsaan Malaysia, Bangi 43600, Selangor, Malaysia; syuhaimi@ukm.edu.my

* Correspondence: s1089104@mail.yzu.edu.tw (M.N.A.S.); ishwang@saturn.yzu.edu.tw (I.-S.H.)

Abstract: An alternative networking approach called Software Defined Networking (SDN) enables dynamic, programmatically efficient network construction, hence enhancing network performance. It splits a traditional network into a centralized control plane and a configurable data plane. Because the core component overseeing every data plane action is the controller in the control plane, which may contain one or more controllers and is thought of as the brains of the SDN network, controller functionality and performance are crucial to achieve optimal performances. There is much controller research available in the existing literature. Nevertheless, no qualitative comparison study of OpenFlow-enabled distributed but logically centralized controllers exists. This paper 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 within two custom network scenarios (uniform and non-uniform host distribution). Our analysis reveals that the Ryu controller outperforms the OpenDayLight controller in terms of latency, packet loss, and round-trip time, while the OpenDayLight controller performs well in terms of throughput, bandwidth, and jitter. Throughout the entire experiment, the HyperFlow and ONOS controllers performed worst in all performance metrics. Finally, we discuss detailed research findings on performance. These experimental results provide decision-making guidelines when selecting a controller.

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

check for updates

Citation: Sheikh, M.N.A.; Hwang, I.-S.; Raza, M.S.; Ab-Rahman, M.S. A Qualitative and Comparative Performance Assessment of Logically Centralized SDN Controllers via Mininet Emulator. *Computers* **2024**, *13*, 85. https://doi.org/10.3390/computers13040085

Academic Editor: Kannan Govindarajan

Received: 2 February 2024
Revised: 10 March 2024
Accepted: 15 March 2024
Published: 25 March 2024

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 device is used to conduct the majority of traditional network activities. A dedicated device for application delivery may contain 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 networks in use today are still conventional networks. Typically, a network system's control plane is entirely distributed. The data and control planes of a networking device are separate. The control-forwarding plane's rules determine how the data plane forwards packets. Integrated firmware informs the hardware of a networking device's destination when a packet arrives. It is necessary for the network administrator to manually perform low-level setup on these vendor-specific networking devices via the CLI whenever the forwarding policy is changed, as this requires reconfiguring the nodes with their unique interface. In addition, the absence of an open standard interface limits researchers' capacity to develop and evaluate applications. The scalability of horizontal networks is another challenging issue in the traditional networking paradigm. This makes it challenging to manage a network that is so dynamic and always changing in status. As a result, a fresh approach to network administration is required in order to move beyond these established network limitations [1].

The technology known as "software-defined networking" (SDN) is expected to transform 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 a single one are among the primary characteristics of the SDN architecture [2].

SDN does have certain drawbacks, though. Organizations face a number of difficulties when switching from conventional networks to SDN-based infrastructure. First of all, investing in new hardware, software, and staff training for SDN technologies comes at a high expense. SDN's complexity can also cause operational difficulties and possible interruptions throughout the migration process, given its interaction with current systems and the learning curve administrators must experience. The constraints posed by interoperability arise from the need for enterprises to guarantee compatibility between SDN components and legacy infrastructure, minimize vendor lock-in, and facilitate the smooth integration of SDN with other IT systems. To maximize the long-term rewards of SDN adoption while minimizing disruption and optimizing return on investment, effective planning, stakeholder alignment, and calculated investments are necessary for successfully negotiating these obstacles. SDN controllers alone are in charge of all transmission operations, since they are frequently treated 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].

SDN offers more flexibility and control over network resources, which sets it apart from traditional networks in terms of scalability, agility, and performance. Scalability is improved by SDN via programmable control and centralized administration, which enable administrators to dynamically distribute resources and modify network regulations in response to evolving requirements. Because SDN can automate network provisioning, setup, and optimization, new services and applications may be launched quickly. This is what gives SDN its agility. Additionally, through traffic engineering and QoS regulations, SDN designs frequently demonstrate increased performance by lowering latency, optimizing traffic flows, and enabling more effective use of network capacity. All things considered, SDN's centralized control, programmability, and automation capabilities enable enterprises to grow their networks more successfully, react more quickly to changing requirements, and attain superior performance in comparison to traditional network architectures. In addition, the controller is in charge of tracking and analyzing data flow in real time. The massive rise in 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 a controller's performance in order to provide effective traffic routing, which will increase resource usage for improved network performance [4].

The efficiency of SDN controllers has been the subject of several studies. There is still a gap 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] 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 paper, which is shown in Figure 1. The issues with distributed but logically centralized SDN-based controllers have not 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 compared and assessed the attributes of 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 carried out specifically to compare performance in a Mininet setting. The choice of controller to use for deployment is a crucial consideration when dealing with SDN architecture. Every controller has a functioning domain as well as pros and cons of its own. The necessity of choosing a controller is what motivated this study. This article differs from previous literature in that it concentrates on the control plane features by providing a quick overview and description of general control plane architectural classification, along with particular controller types associated with each category. The benefits and drawbacks of distributed but logically centralized controllers are also covered in this study, allowing network engineers and researchers to make well-informed choices about the development, use, and optimization of distributed control plane architectures.

Investigation of OpenFlow-based controllers' performance has been previously conducted. Several works [6–19] give some quantitative comparisons, although most of them use either a single application or a simple environment where several trials may be conducted. We selected a custom scenario for this study that utilizes Mininet capabilities. Using Mininet as an efficient network simulator for a 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 contribute the following in this paper:

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

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 a Mininet-based performance assessment study in Section 3. Qualitative and quantitative description and evaluation of five selected controllers, topology description with setup of the experiment, and 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 controllers' advantages and the most likely challenges that one could encounter for distributed controllers. Section 7 discusses the conclusions and future work.

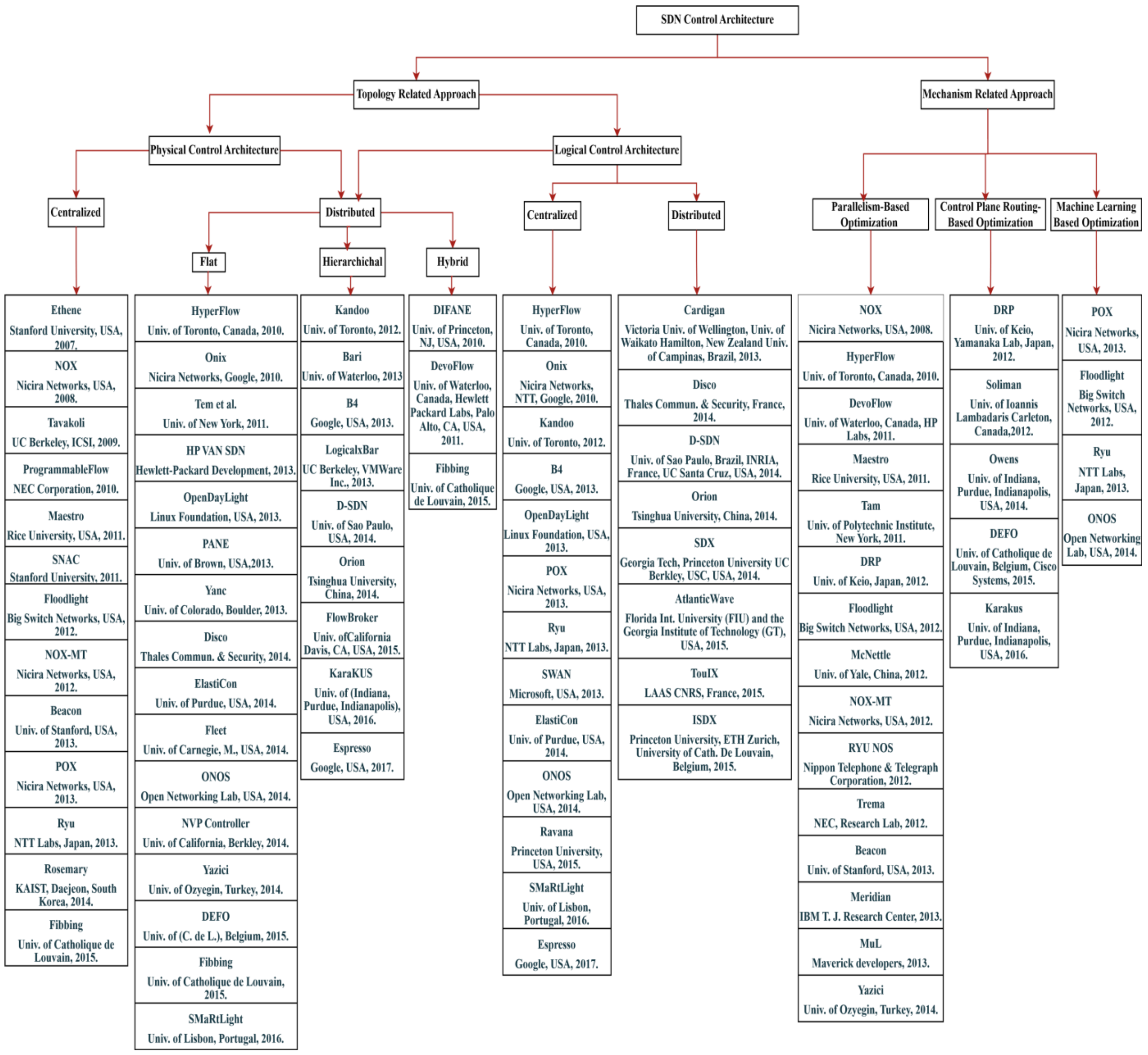

SDN Control Architecture
Topology Related Approach
Mechanism Related Approach
Physical Control Architecture
Logical Control Architecture
Centralized
Distributed
Centralized
Distributed
Parallelism-Based Optimization
Control Plane Routing-Based Optimization
Machine Learning Based Optimization
Flat
Hierarchichal
Hybrid
Ethene Stanford University, USA, 2007.
NOX Nicira Networks, USA, 2008.
Tavakoli UC Berkeley, ICSI, 2009.
ProgrammableFlow NEC Corporation, 2010.
Maestro Rice University, USA, 2011.
SNAC Stanford University, 2011.
Floodlight Big Switch Networks, USA, 2012.
NOX-MT Nicira Networks, USA, 2012.
Beacon Univ. of Stanford, USA, 2013.
POX Nicira Networks, USA, 2013.
Ryu NTT Labs, Japan, 2013.
Rosemary KAIST, Daejeon, South Korea, 2014.
Fibbing Univ. of Catholique de Louvain, 2015.
HyperFlow Univ. of Toronto, Canada, 2010.
Onix Nicira Networks, Google, 2010.
Tem et al. Univ. of New York, 2011.
HP VAN SDN Hewlett-Packard Development, 2013.
OpenDayLight Linux Foundation, USA, 2013.
PANE Univ. of Brown, USA,2013.
Yanc Univ. of Colorado, Boulder, 2013.
Disco Thales Commun. & Security, 2014.
ElastiCon Univ. of Purdue, USA, 2014.
Fleet Univ. of Carnegie, M., USA, 2014.
ONOS Open Networking Lab, USA, 2014.
NVP Controller Univ. of California, Berkley, 2014.
Yazici Univ. of Ozyegin, Turkey, 2014.
DEFO Univ. of (C. de L.), Belgium, 2015.
Fibbing Univ. of Catholique de Louvain, 2015.
SMaRtLight Univ. of Lisbon, Portugal, 2016.
Kandoo Univ. of Toronto, 2012.
Bari Univ. of Waterloo, 2013
B4 Google, USA, 2013.
LogicalxBar UC Berkeley, VMWare Inc., 2013.
D-SDN Univ. of Sao Paulo, USA, 2014.
Orion Tsinghua University, China, 2014.
FlowBroker Univ. ofCalifornia Davis, CA, USA, 2015.
KaraKUS Univ. of (Indiana, Purdue, Indianapolis), USA, 2016.
Espresso Google, USA, 2017.
DIFANE Univ. of Princeton, NJ, USA, 2010.
DevoFlow Univ. of Waterloo, Canada, Hewlett Packard Labs, Palo Alto, CA, USA, 2011.
Fibbing Univ. of Catholique de Louvain, 2015.
HyperFlow Univ. of Toronto, Canada, 2010.
Onix Nicira Networks, NTT, Google, 2010.
Kandoo Univ. of Toronto, 2012.
B4 Google, USA, 2013.
OpenDayLight Linux Foundation, USA, 2013.
POX Nicira Networks, USA, 2013.
Ryu NTT Labs, Japan, 2013.
SWAN Microsoft, USA, 2013.
ElastiCon Univ. of Purdue, USA, 2014.
ONOS Open Networking Lab, USA, 2014.
Ravana Princeton University, USA, 2015.
SMaRtLight Univ. of Lisbon, Portugal, 2016.
Espresso Google, USA, 2017.
Cardigan Victoria Univ. of Wellington, Univ. of Waikato Hamilton, New Zealand Univ. of Campinas, Brazil, 2013.
Disco Thales Commun. & Security, France, 2014.
D-SDN Univ. of Sao Paulo, Brazil, INRIA, France, UC Santa Cruz, USA, 2014.
Orion Tsinghua University, China, 2014.
SDX Georgia Tech, Princeton University UC Berkley, USC, USA, 2014.
AtlanticWave Florida Int. University (FIU) and the Georgia Institute of Technology (GT), USA, 2015.
TouIX LAAS CNRS, France, 2015.
ISDX Princeton University, ETH Zurich, University of Cath. De Louvain, Belgium, 2015.
NOX Nicira Networks, USA, 2008.
HyperFlow Univ. of Toronto, Canada, 2010.
DevoFlow Univ. of Waterloo, Canada, HP Labs, 2011.
Maestro Rice University, USA, 2011.
Tam Univ. of Polytechnic Institute, New York, 2011.
DRP Univ. of Keio, Japan, 2012.
Floodlight Big Switch Networks, USA, 2012.
McNettle Univ. of Yale, China, 2012.
NOX-MT Nicira Networks, USA, 2012.
RYU NOS Nippon Telephone & Telegraph Corporation, 2012.
Trema NEC, Research Lab, 2012.
Beacon Univ. of Stanford, USA, 2013.
Meridian IBM T. J. Research Center, 2013.
MuL Maverick developers, 2013.
Yazici Univ. of Ozyegin, Turkey, 2014.
DRP Univ. of Keio, Yamanaka Lab, Japan, 2012.
Soliman Univ. of Ioannis Lambadaris Carleton, Canada,2012.
Owens Univ. of Indiana, Purdue, Indianapolis, USA, 2014.
DEFO Univ. of Catholique de Louvain, Belgium, Cisco Systems, 2015.
Karakus Univ. of Indiana, Purdue, Indianapolis, USA, 2016.
POX Nicira Networks, USA, 2013.
Floodlight Big Switch Networks, USA, 2012.
Ryu NTT Labs, Japan, 2013.
ONOS Open Networking Lab, USA, 2014.

Figure 1. Control plane architecture classification of software defined networking (SDN) [3,20–26].

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 [20–24]. In their thorough investigation of the scalability problems in SDN-based control planes, Karakus and Durresi [23] provided a classification and taxonomy of the state of the art, which is considered the very first taxonomy of control planes. They 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 in the design of centralized and distributed controllers were the topics for topology-related methods. Mechanism-related methods included many techniques, including parallel-based and routing scheme-based optimization, to improve controllers and their scalability issues. Machine learning-based optimization was added by Abuarqoub et al. [25] to only the mechanism-based approach. In both cases, the classifications were almost identical. However, both classifications lack the further logical categorization of distributed and centralized SDN control planes. Additionally, a number of distinct controller types are missing from this taxonomy. In addition to 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 [27]. However, not all control plane architecture was provided by this effort, and their taxonomy lacks specific controller types. Additionally, Bannour et al. [24] 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 planes. Isong et al. [28] did not address hybrid orientation, instead explicitly classifying control planes into physical and logical components. Only in situations when the mechanism-based approach is completely absent did they view the control plane from a topology-related approach. In our work, topology-related and mechanism-related control designs are separated. 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 the two categories of mechanism-related approaches. Every category has a unique controller kind.

Several studies have been performed with the aim of comparing SDN controllers [2,13–19,24–38]. The authors of paper [24] examined 15 distributed SDN controllers while taking into account the following factors: (1) Scalability; (2) Reliability; (3) Consistency; (4) Interoperability; (5) Monitoring; (6) Security. All of the aforementioned studies took into account either open source or widely used SDN-based controllers. Only distributed controllers were examined by them. However, they did not evaluate and compare the characteristics of distributed but logically centralized SDN-based controllers. In this study, we provide 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 [13], which only looked at controller performance while taking into account a small number of controllers (NOX-MT, Beacon, and Maestro). However, over time, new controllers like POX, Ryu, FloodLight, and OpenDayLight have taken the place of these ones. A thorough analysis of SDN OpenFlow Controllers was carried out by Shalimov et al. The efficacy of the commonly used SDN controllers NOX, POX, Beacon, FloodLight, MuL, Maestro, and Ryu is compared. The hcprobe tool [14] 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 [15] serve as the foundation for comparison of controllers. Analytic Hierarchy Process (AHP), a monotonic interpolation/extrapolation mechanism that transfers the values of attributes to a value 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 [16] 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 [17]. Because rules in proactive mode are loaded into the switch at startup rather than every time the switch receives 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 best.

More research is conducted in [18], 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 [19]. Because Java (Java 1.6) 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 the net for languages to 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 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 of [33] emphasized the problem of software aging. The primary problem under investigation is memory leakage, 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 [34,35] examined twelve whereas authors of [36] examined eight 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.

Study [37] examined a number of open source controllers, including Ryu, POX, OpenDayLight (ODL), and Open Network Operating System (ONOS), by qualitatively assessing both their performance and their features. The authors used the Cbench [25] tool to benchmark the controllers under various operating situations and quantitatively assess a number of characteristics.

The authors of paper [38] 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 [39], OFLOPS-Turbo [40], Cbench [13], PktBlaster [41], and OFCBenchmark [42] 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 in 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 include the Mininet [32] and NS3 [43] network simulators.

Papers [2,29] provide a comparative assessment of previous benchmarking research and an in-depth look at the capabilities of benchmarking tools. A summary of SDN-based research utilizing Mininet is provided in Paper [32], yet there is no survey of SDN controller performance evaluation using Mininet. Table 1 gives a comprehensive summary of all the numerous features of the controllers and Table 2 gives a thorough review of the performance evaluations. 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.

Table 1. Distributed but logically centralized SDN controllers feature comparison table [29].

Software Framework	Language	API Style	OpenFlow Support	Communication Mechanism	Platforms	Interface	Licensing	Multithreading	Modularity Level	Consistency Enforcement	Documentation Depth
HyperFlow	C++	RESTful	1.0	Publish/Subscribe	Linux	Web UI	Custom	Yes	Moderate	Absent	Minimal
Onix	C++	Onix Specific	1.0, OVSDB	Coordination Service	Linux	Command Line	Open Source (Apache 2.0)	Yes	High	Absent	Minimal
Kandoo	C, C++, Python	Java RPC	1.0–1.2	Message Passing	Linux	Command Line	Custom	Yes	Extensive	Absent	Minimal
B4	C++	BGP	Not Specified	Not Specified	Linux	Not Specified	Custom	Not Specified	Not Specified	Not Specified	Minimal
OpenDay-Light	Java	Multiple RESTful Protocols	1.0, 1.3	Distributed Coordination	Cross-Platform	Command Line and Web UI	Open Source (EPL 1.0)	Yes	Extensive	Present	Moderate
POX	Python	Ad-hoc	1.0	Scripting	Cross-Platform	Command Line and GUI	Open Source (Apache 2.0)	No	Low	Absent	Minimal
Ryu	Python	RESTful	1.0–1.5	Scripting	Linux, MacOS	Command Line	Open Source (Apache 2.0)	Yes	Moderate	Present	Moderate
SWAN	Python	RESTful	Not Specified	Scripting	Cross-Platform	Not Specified	Custom	Not Specified	Not Specified	Not Specified	Minimal
ElastiCon	Java	Not Mentioned	Not Specified	Not Specified	Not Specified	Not Specified	Custom	Not Specified	Not Specified	Not Specified	Minimal
ONOS	Java	RESTful and Neutron	1.0, 1.3	Consensus Algorithm	Cross-Platform	Command Line and Web UI	Open Source (Apache 2.0)	Yes	Extensive	Present	Moderate
Ravana	Python	OF Direct	Extensions Supported	Not Specified	Linux	Not Specified	Custom	Not Specified	Not Specified	Not Specified	Minimal
SMaRtLight	Java	RESTful	1.3	Consensus Protocol	Linux	Command Line	Custom	Not Specified	Not Specified	Absent	Minimal
Espresso	Java	RESTful	Not Specified	Not Specified	Linux	Not Specified	Custom	Not Specified	Not Specified	Not Specified	Minimal

Note: 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.

Table 2. Existing performance analysis studies using Mininet.

Ref.	Controller's Evaluated	Topology Used	Evaluation Metrics	Lessons Learned
[4]	Ryu	Single	Bandwidth, Throughput, RTT, Transmission of Data Packet	Ryu is regarded as one of the most effective traffic engineering controllers.
[7]	ODL, Ryu	Internet and Bridge	Topology Discovery Time, Delay, Throughput, CPU Utilization	ODL outperforms Ryu for both the Internet and Bridge topologies.
[17]	FloodLight, NOX, POX, Trema	-	Proactive, Reactive	Reactive technique lowers controller performance, whereas a proactive controller receives fewer request messages.
[30]	Ryu, POX, FloodLight, ODL, ONOS	Single, Linear, Tree (Google and Facebook)	Latency, Throughput, Delay, Bandwidth	ODL performs better among the selected distributed controllers, whereas Ryu performs best among the selected centralized controllers.
[32]	ODL, Ryu	Custom	Latency	ODL performs best.
[44]	POX	-	Bandwidth	Prior to self-learning, the execution switch component has more bandwidth.
[45]	Ryu	Linear	Latency	The controller receives more requests as the network grows.
[46]	FloodLight	Linear, Tree	Time to Create and Destroy the Virtual Networks, Memory Usage	When the number of virtual networks increases, the Mininet takes a longer time and uses more memory.
[47]	ONOS	Custom	Latency, Topology Discovery Time	ONOS needs more work to be done to be accepted by all.
[48]	POX, Ryu, ONOS, ODL	Tree	RTT, Bandwidth	ONOS performs better in Switch mode, Hub mode performance is nearly the same.
[49]	POX	Linear	RTT, Delay, Bandwidth, Throughput, Mean Data Rate	Hub components outperform Switch components in terms of performance.
[50]	POX, FloodLight	Tree	Scalability	Simulation environment vs. time needed to construct a topology is noteworthy.
[51]	FloodLight, ODL	Single, Linear, Tree	Latency and Packet Loss	FloodLight outperforms ODL in terms of packet loss for tree topologies and latency for linear topologies in densely trafficked networks, while ODL demonstrates better latency performance in networks with low load and for tree topologies with medium load.
[52]	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.
[53]	POX, Ryu and Pyretic	Star	RTT, Latency, Throughput	Ryu outperforms Pyretic and POX in terms of speed.
[54]	OF reference Controller	Single, Linear, Tree	Bandwidth Utilization, Packet Transmission Rate, Round-Trip Propagation Delay, Throughput	The scalability issue is mitigated in a tree topology network where the load is distributed across branches.
[55]	POX and Ryu	Combination of (Linear and Tree), DCN Tree, Mesh	Bit-rate, Delay, Packet-rate, and Jitter	POX performs better in layer 1 switching scenarios, whereas Ryu produced far higher performance outcomes in layer 2 switching.
[56]	ODL and ONOS	Tree	Cluster Failure Recovery Time	For GUI, clusters, link-up, switch-up, and throughput, ONOS performs well. For stability and topology discovery, ODL performs better.
[57]	OF- reference Controller	Custom	Bandwidth, Throughput, Jitter, Packet Loss	OF network operates similarly to a regular network, except that the data plane and control plane are separated.

Table 2. *Cont.*

Ref.	Controller's Evaluated	Topology Used	Evaluation Metrics	Lessons Learned
[58]	POX, FloodLight	Star, Linear, Tree	Delay, Throughput	The FloodLight Controller performs better than the POX.
[59]	FloodLight	Mesh	Throughput and Latency	The network is subject to load as the number of nodes linked to switches rises.
[60]	FloodLight, ODL	Tree, Single, Linear, Torus	Throughput, Data Transfer Rate and Latency	FloodLight works better in linear, tree, and torus topologies but not in a single topology.
[61]	OVS- Controller, POX, FloodLight, ODL	Single, Linear, Tree	Latency, Bandwidth Utilization, Jitter, Packet Loss	In linear topologies, FloodLight Controller has severe data loss, whereas ODL Controller is unable to manage the load provided by it.
[62]	OF reference controller	Custom	Throughput, RTT, Delay, Jitter	Delay, Jitter, RTT and throughput are efficient QoS parameters.
[63]	Libfluid, ONOS, ODL, POX and Ryu	Linear	Throughput, Delay	As the number of switches and hosts rose, throughput declined and latency increased.
[64]	POX, FloodLight	Single, Linear, Tree, Custom	Throughput and Round-Trip Delay	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.
[65]	POX, Ryu	Single, Linear, Tree, Dumbbell, DCN, SAT	Throughput, Latency	Ryu has superior performance.
[66]	Ryu	Mesh	Throughput	Ryu is an extremely resource-demanding controller.
[67]	ONOS, Open MUL, POX	Linear	Latency, Throughput, Topology Discovery Time	Performance evaluation is considerably underestimated by Cbench.
[68]	ONOS	Mesh	Throughput	When there are varying numbers of nodes, ONOS acts steadily.
[69]	NOX	Custom	Throughput, Response Time	Compared to ROIA and Multiple Packet Schedular, NOX performs better.
[70]	NOX, FloodLight	Custom	Throughput, Response Time	The internal NOX controller is inferior to the FloodLight controller.
[71]	Ryu, POX and Pyretic	Tree	RTT	The pyretic controller performs better with Software Defined Networking.
[72]	POX, FloodLight, ODL	Tree, Mesh	RTT, Throughput	POX outperforms FloodLight and ODL in terms of RTT and throughput.
[73]	ODL, Ryu	Tree	Throughput, Switchover Time	Compared to Mininet simulation, utilizing a hardware testbed experiment offers greater and more consistent throughput.
[74]	Ryu	Tree	Throughput, Switchover Time	Number of flow entries within the data plane can be reduced by using MFT.
[75]	Ryu, FloodLight, ODL, ONOS	Linear, Tree, Mesh	Delay, Throughput	In terms of throughput and latency, the FloodLight controller performs better.
[76]	Beacon	Mesh	Throughput	When the number of nodes increases, throughput decreases.
[77]	ODL, ONOS	Tree	Burst Rate, Throughput, RTT and Bandwidth	The ODL controller outperforms the ONOS.
[78]	NOX, POX, ONOS, Ryu	Custom	Throughput, Response Time	For response time, POX is superior, whereas for throughput, ONOS is superior.
[78]	Ryu, POX, ONOS, FloodLight	Linear	Delay, Jitter and Throughput	FloodLight performs best.
[79]	ODL, Ryu	Linear	Throughput, Delay, Packet Loss, Resource Utilization	Resource Utilization tests revealed that the ODL controller performed better.
[80]	POX	Linear	Bandwidth, CPU Load, Packet Loss, Latency, Throughput	There is 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.
[82]	Ryu	Single	Bandwidth, Throughput, RTT, Jitter, Packet Loss	Ryu is a great option for research and small commercial applications.
[83]	ONOS, FloodLight, Ryu	Leaf Spine DCN	Throughput, Topology Discovery Time	ONOS controller performs poorest in network topology discovery time and best in flow testing.
[84]	FloodLight, POX, NOX	Linear, Tree, Custom	Throughput, Bandwidth, Packet Loss, Latency, Topology Discovery Time	The FloodLight controller is fastest in all topologies.
[85]	POX, Ryu	Mesh	Throughput, Delay, Jitter, Packet Loss	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 in hosts or switches has an impact on performance.
[87]	FloodLight	Single, Linear, Tree	Throughput, RTT	By rerouting traffic with a greater RTT and lower throughput, SDN can handle link failure circumstances.
[88]	NOX, ONOS, FloodLight, ODL, POX, Ryu	Custom	Throughput, Response Time	An increasing number of operations impacts throughput and response time.
[89]	POX	Tree, Bus, Star	Bandwidth Utilization, Jitter and Packet Loss	The number of open switches plays a significant role in the star topology.
[90]	ODL	Custom	Delay, Throughput, Jitter, Packet Drop, Bitrate, Bytes Received	Multi-controller networks are more dependable and achieve high availability.
[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]	Ryu	bespoke network topology	RTT, bandwidth and throughput	Ryu outperforms other controllers.

3. Controller Classification and Design Choices

We 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 act in a largely uniform manner. Upon examining each controller, we conclude that most of them do not have a classification basis for their responsibilities, functions, or methods of operation. It is possible that the deployment architecture is the sole applicable classification criterion [29]. Most controllers only utilized one controller because the main objective of SDN was to consolidate the control plane. Still, this led to scalability problems and a single point of failure. A distributed design allows several controllers to function inside a domain in a flat or hierarchical structure.

Programming language and API selection can have a big influence on controller performance and network device compatibility. A few examples of variables that impact performance include concurrency support, memory management, and language execution efficiency. High-performance SDN controllers typically favor languages with a reputation for performance, such as C/C++ and Java. Furthermore, since the API's design dictates how controllers communicate with other components and network devices, it is essential to interoperability. Interoperability is enhanced by using standardized APIs, such as RESTful APIs or OpenFlow, which allow controllers and other network devices to communicate with one other independent of vendor or protocol. Additionally, user-friendly and well-documented APIs facilitate integration tasks and promote vendor and developer usage, which improves interoperability and supports ecosystem proliferation. Thus, to achieve maximum controller performance and ensure flawless interoperability with various network devices, choosing the right programming languages and designing the API is essential.

When combined with platform compatibility and modularity, Software-Defined Networking (SDN) offers a great deal of flexibility and capability in an Internet of Things (IoT) environment. Consider a smart city deployment in which several IoT devices are dispersed throughout the urban environment, each with a different set of hardware needs and communication protocols. Administrators may ensure interoperability and effective resource management by integrating IoT devices from many manufacturers and technologies with ease by utilizing an SDN controller with a modular design and platform compatibility. A modular design method makes it simple to adjust to evolving IoT device landscapes, facilitating the quick and inexpensive deployment of new devices and services. Platform compatibility improves overall system performance, scalability, and resilience by facilitating seamless coordination and communication across heterogeneous IoT devices. Thus, by promoting creativity and adaptability in responding to the changing demands of the smart city context, this integrated strategy makes it easier to effectively manage the IoT ecosystem.

3.2. Qualitative Comparison

A thorough overview of the many characteristics of distributed but logically centralized controllers is provided in Table 1. We will not discuss each controller separately. Instead, we showcase the characteristics of and design options for controllers. The characteristics and design options we followed is taken from [29].

Programming Language: A variety of programming languages, including C, C++, Java, Java Script, Python, Ruby, Haskell, Go, and Erlang, were 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 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 peer relationships with one another through the usage of east-west API (EWBI). Not every controller offers every API, and only a small number of APIs 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 statistical 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, SD-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 paper. 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 lacked adequate documentation when we accessed them online. Conversely, those that undergo frequent updates come with community-based 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 perform testing and mimic SDN networks. We utilized Mininet as a simulation tool, specifically as a network emulator. It enables the building of networks with different topologies made up of several virtual hosts, connections, and switches. With this application, we may build the network to our precise specifications, share it with others, and eventually develop it with real hardware. We might quickly and easily access any network component via the Mininet CLI. Before implementing our network in real life, we can test and refine it in a virtual environment by using Mininet. The standard Mininet topology consists of switches and hosts connected to the switches, and 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 offers only a few topologies—tree, linear, and single by default—it is possible to design any desired topology by creating a Python script [30,32].

The capacity of Mininet, a well-known open source network emulator, to build virtual networks on a single computer makes it an invaluable tool for network research and development. Mininet's user-friendly interface makes it simple for users to experiment with different network setups and quickly prototype network topologies. The best option for testing SDN applications is to use it, as it can simulate intricate network behaviors and supports OpenFlow switches. Mininet is not without its limitations, though, including simplified network behavior simulation compared to real-world networks and scalability issues brought on by its reliance on host system resources. Its application in various research and production scenarios is also limited due to its lack of support for several modern networking capabilities and protocols. Even with these drawbacks, Mininet is still a useful tool for teaching, quick prototyping, and preliminary testing of network topologies and applications [46,54,67].

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 current Mininet-based efforts, which will be covered in this part.

Several papers revealed how to use the open source Ryu SDN controller to develop SDN architecture for network traffic analysis. Based on several network topologies, the proposed study evaluated the performance parameters of SDN architecture, such as bandwidth, throughput, jitter, packet loss round-trip time, switchover time, etc. Studies showed that Ryu is a great option for research and small commercial applications because it was developed in Python [4,45,66,74,82,92].

Conversely, the OpenFlow interface was implemented in different network simulation scenarios with the open source controller POX for network traffic analysis in papers [43,48,80,89]. Analysis and evaluation were performed on things like packet loss, CPU load, bandwidth utilization, etc. According to those studies, efficient use of bandwidth occurs beginning at 100 Mbps and increases gradually up 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 [46,59,87], OpenDayLight [90], ONOS [47,68], NOX [69] and Beacon controllers [76].

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 [69,70,78,88]. 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 studies [55,65,85,91]. 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 [53,71] contrasted the Ryu and POX controllers with Pyretic, another Python-based controller. According to these studies, Pyretic controller works better in tree topology while Ryu performs better in star topology. However, when compared to other Java-based controllers like FloodLight and 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 [7,32,50,58,64,73,78,79,81,84].

In papers [54,57,62], 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 [17]. 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 best.

Using a Mininet emulator, Stancu et al. evaluated four SDN controllers—POX, Ryu, ONOS, and OpenDayLight [48]. 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 [30,51,52,60,61,72,75,86]. 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 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 [56,63,67,77,83]. 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 flow testing and worst in network topology discovery time.

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 were examined. Out of the thirteen controllers that were previously mentioned, these were chosen because (1) the source code for the controllers 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 [93].

4.1. Selected Controllers and Their Qualitative Evaluation

We will give a brief description of the chosen controllers in this section; Table 3 shows the qualitative comparison of the controllers.

Table 3. Qualitative analysis of the selected controllers.

Features \ Name	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 and Education	Research, Education and Learning, SDN Application Development	Campus, Research, SDN application development, NFV, Network monitoring and security	Data centre, Carrier-Grade Network, Research, SDN/NFV integration

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 [94].

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 [47]. The Java packages for the ONOS project are loaded into a 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". NTT, or Nippon Telegraph and Telephone Corporation, provides assistance for the Ryu controller. Ryu supports a number of protocols, including NETCONF, OF-config, OpenFlow, and others [4,66].

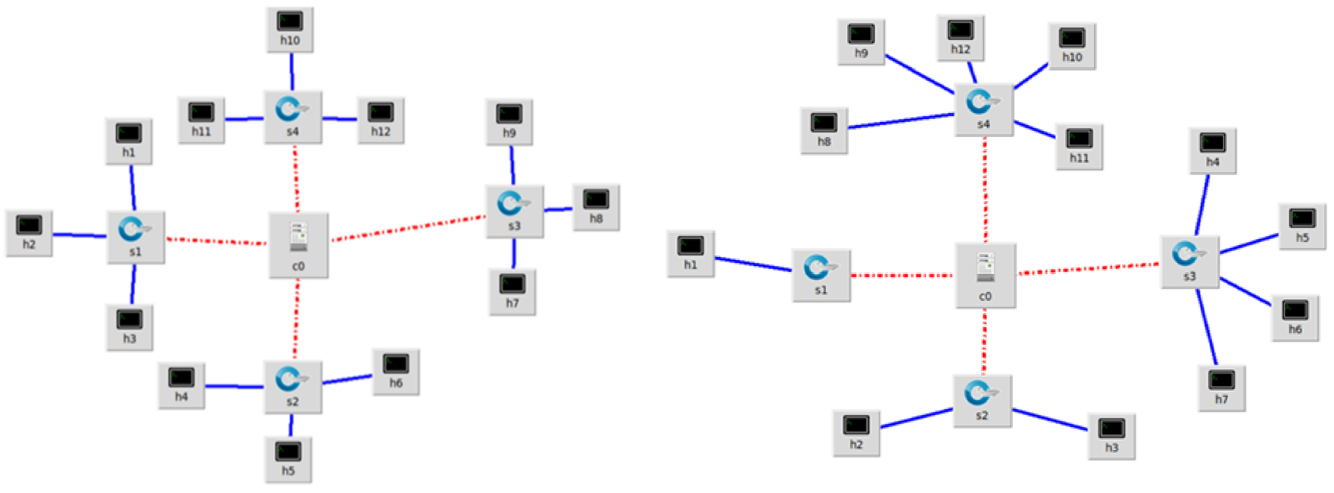
POX: The younger sibling of NOX is a networking software platform called POX (Pythonic Network Operating System). Python is the programming language used in the development of POX [27]. When creating networking software, it can be useful. POX is compatible with several operating systems, including Linux, Mac OS, and Windows. It is compatible with Python 2.7 and lower versions. POX connects to OpenFlow 1.0 switches and has specific support for Open vSwitch/Nicira extensions [44,58,64,65,72,89].

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 [51,72,77,79]. 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

The performance of a network is significantly influenced by its topology. Performance may be greatly enhanced by reducing energy consumption and increasing data transfer rates through careful network topology design and management. An organization's real network is not restricted to simple topologies like linear and tree. With custom topology, users may create whatever topology they choose. Overall, customized topologies provide the adaptability, flexibility, and optimization required to meet the many demands of and difficulties in contemporary network installations. For our experiment, we chose a customized topology with both uniform and non-uniform host distribution. A combination of uniform and non-uniform distribution may be used in many IoT implementations. For instance, a citywide IoT network may have a uniform sensor distribution for some purposes (like environmental monitoring) and a non-uniform distribution for other applications (like intelligent traffic control at high-traffic junctions). 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 2a represents a uniform distribution of hosts to the switch, whereas Figure 2b shows a non-uniform distribution of hosts to the switch. Twelve hosts and four OpenFlow-capable switches were 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 environments.



(**a**) (**b**)

Figure 2. Custom topology. (**a**) Uniform host distribution. (**b**) Non-uniform host distribution.

4.3. Experiment Setup

The test bed consisted of a single PC with an Intel Core i7 CPU operating at 2.90 GHz and 8 GB of RAM. Windows 10 was the host operating system while Ubuntu 22.10 LTS was the guest operating system on the machine. On a 100 Mbit network, congested windows of up to 32 Mbytes were 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 and ping command to create traffic between hosts in order to assess important factors including bandwidth, throughput, round-trip time, delay, jitter, and packet loss.

The iPerf command is used to generate traffic. iPerf is a tool for measuring throughput and creating traffic. To launch an iPerf test, the following commands are used:

$$(\text{iperf} - \text{s} - \text{i}\ 1 - \text{t}\ 100 - \text{w}\ 100\ \text{M})$$

$$(\text{iperf} - \text{c}\ 10.0.0.7 - \text{i}\ 1 - \text{t}\ 100 - \text{w}\ 100\ \text{M})$$

Using the aforementioned commands, one host was utilized as a server and the other as a client. The test was set up in iPerf to run from the client to the server for 100 s (a time interval of 1 s), with a window size of 32 Mbytes at most and as many ICMP packets as the link can handle. We evaluated throughput, packet loss, and bandwidth using the iPerf command. Table 4 displays the features of iPerf. Following the successful session formation by Mininet, we launched two terminals and used iPerf to generate traffic between hosts.

Table 4. iPerf Parameters [84].

Parameters	Description	Value
s	Identify session as server	No value passed
i	Reporting intervals	1 s
t	Time interval	100 s
f	Output format	M
w	Window size	100 M
c	Identify session as client	10.0.0.x server's IP

Subsequently, in the network topology that was planned, the Ping command was used to establish a connection between hosts. Using the ping command, we exchanged 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, and 48 packets between hosts during the test. We then recorded the round-trip time, jitter, and delay. For every test, these measurements and experiments were repeated ten times.

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). Bandwidth, throughput, round-trip time, packet loss, delay, and jitter are critical metrics chosen for controller evaluation due to their direct impact on network performance and user experience. Bandwidth represents the maximum data rate a network can handle, reflecting its capacity and potential throughput. Throughput measures the actual rate of data transfer achieved, providing insights into network efficiency and utilization. Round-trip time indicates the time taken for data to travel from source to destination and back, affecting responsiveness and latency-sensitive applications. Packet loss measures the percentage of lost or discarded packets during transmission, indicating network reliability and congestion issues. Delay and jitter represent variations in packet arrival times and transmission delays, which can degrade the quality of real-time applications such as VoIP or video conferencing. By evaluating controllers based on these metrics, researchers and network operators can assess their ability to optimize network performance, manage congestion, minimize latency, and ensure reliable and efficient data delivery across the network.

5.1. Bandwidth

The maximum amount of network bandwidth in Gb/s that was used is shown by the bandwidth in Figure 3. According to our investigation, the bandwidth for the 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 best and the Ryu controller worst.

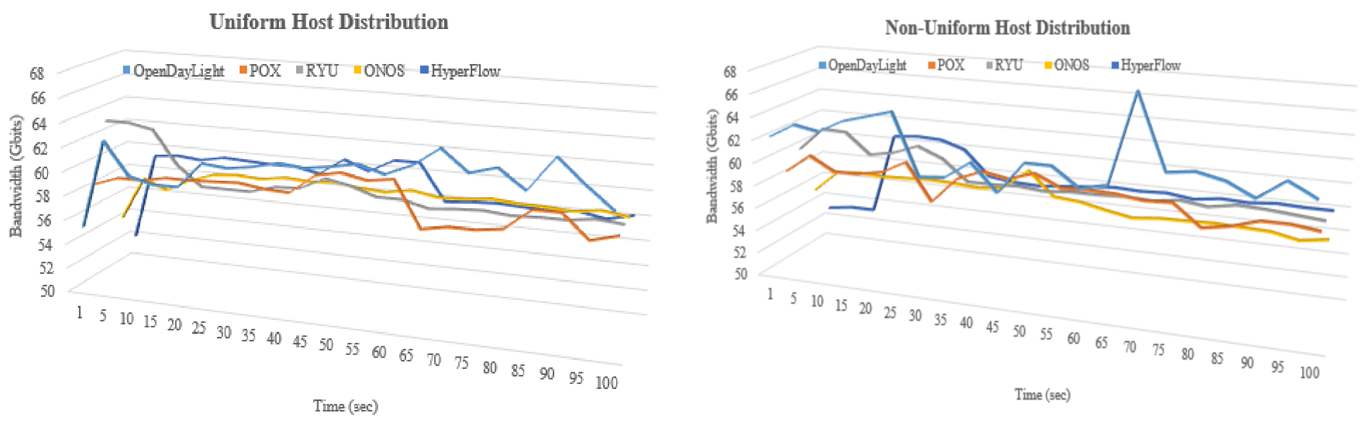


Figure 3. Bandwidth.

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. The OpenDayLight, POX, Ryu, ONOS, and HyperFlow controllers 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.

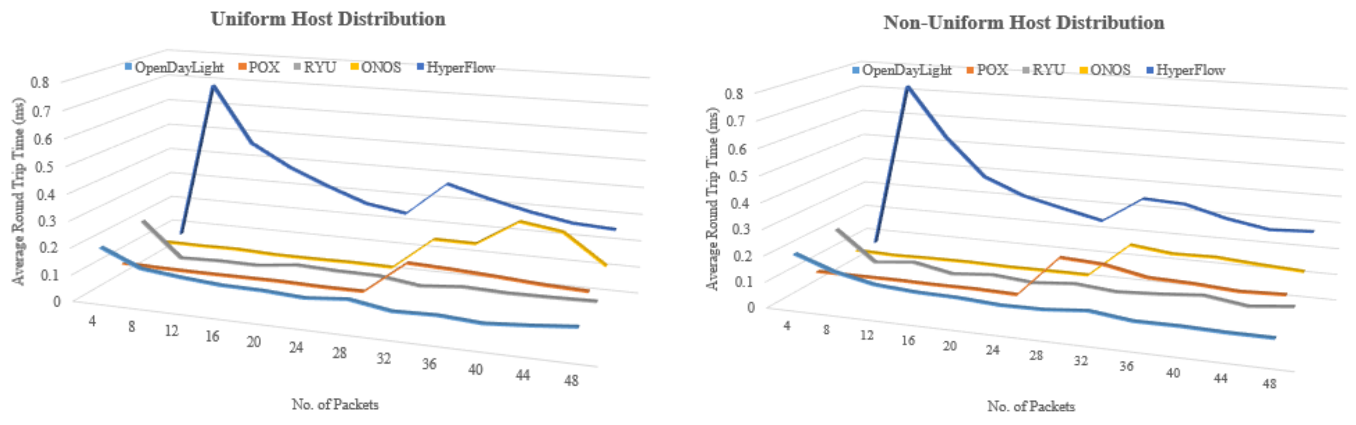
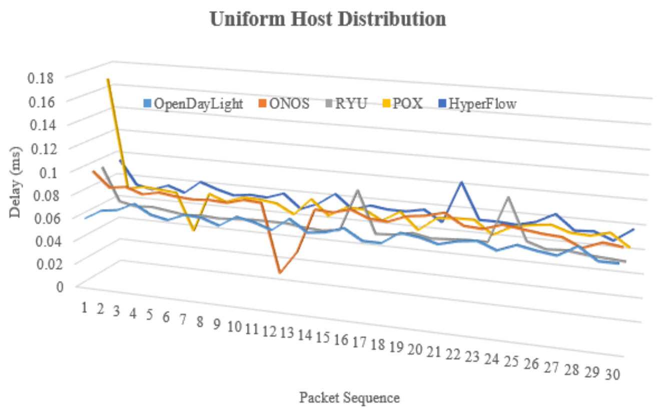


Figure 4. Round-trip time.

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 0.0785 ms. Both times, the POX controller and the OpenDayLight controller had extremely similar values, with POX being marginally superior to OpenDayLight. For both situations, ONOS experienced the highest delay.



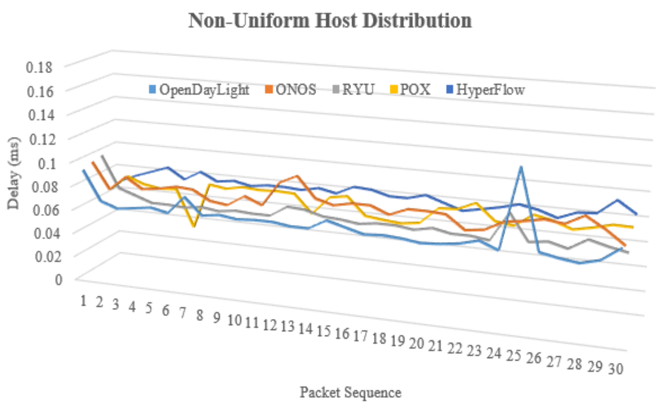
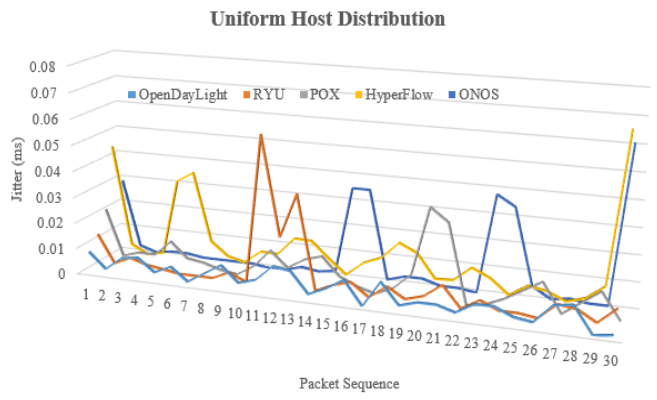


Figure 5. Delay.

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 delay periods between data packets 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 measured in 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 0.0099 ms for uniform host distribution and 0.0046, 0.0053, 0.0075, 0.0147 and 0.0118 ms for non-uniform host distribution. In all scenarios, Ryu has the worst jitter whereas OpenDayLight surpasses all other controllers.



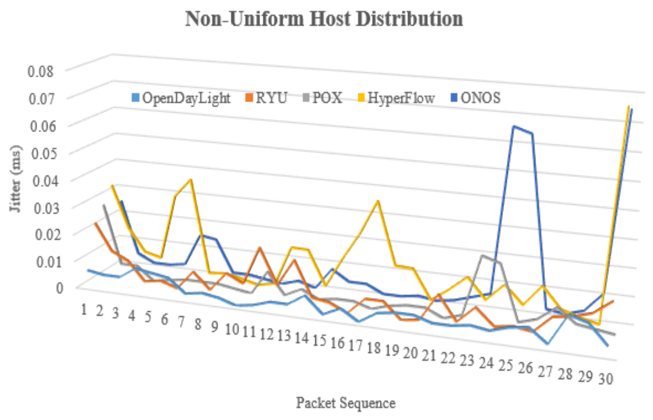


Figure 6. Jitter.

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 the 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 percent, while for uniform distribution, the values are 0.05, 12.5, 20.5, 8.5, and 22.5 percent. Ryu experiences the least packet loss in both circumstances, whereas HyperFlow experiences the most.

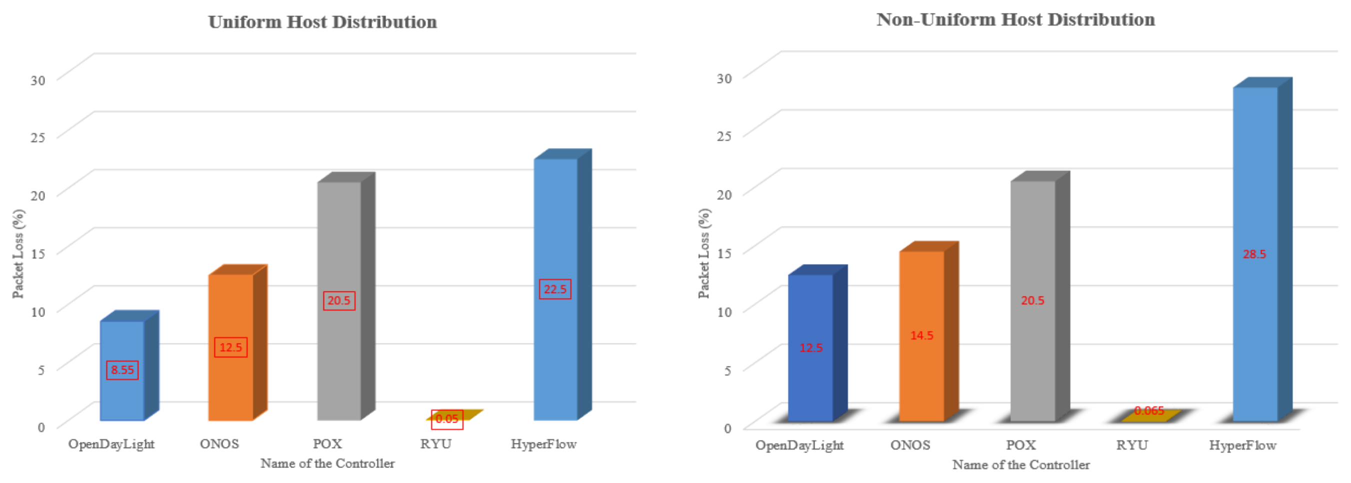


Figure 7. Packet Loss.

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 data transfer burst, 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 s). 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 GB/s, as shown in Figure 8. In both situations, Ryu has the lowest throughput while OpenDayLight has the greatest.

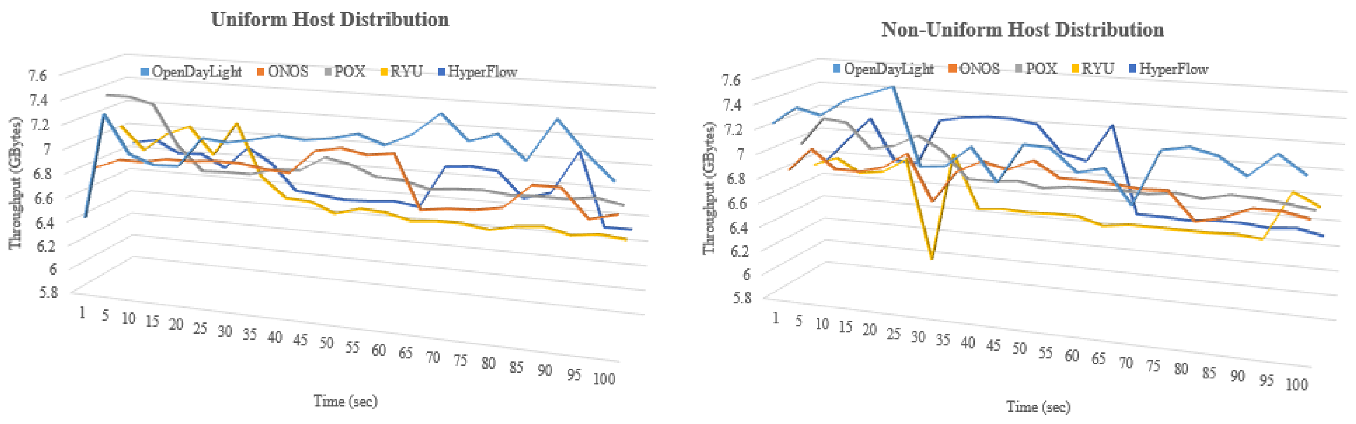


Figure 8. Throughput.

6. Discussion

This section 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 this thorough analysis, the OpenDayLight controller performs better than the Ryu controller in all performance parameters except delay, packet loss and round-trip time, which are the Ryu controller's strong points. In all performance metrics, the ONOS and HyperFlow controller perform worst during the whole trial.

The OpenDayLight controller surpasses the POX controller by 1.00% in bandwidth considered 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. On the other hand, the OpenDayLight controller 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. Considering a uniform host distribution for delay, the 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%. Meanwhile, in a 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 considered 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. For the same factor, 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 a non-uniform host distribution. The OpenDayLight controller surpasses the ONOS, POX, Ryu, and HyperFlow controllers in throughput considered in a uniform host distribution by 1.02%, 0.99%, 1.77%, and 1.34%, respectively. In comparison, 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 considered 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. A summary is shown in Table 5.

Table 5. Feature-based performance comparison with respect to other controllers in percentage.

	Bandwidth								Jitter								Throughput							
	Uniform				Non-uniform				Uniform				Non-uniform				Uniform				Non-uniform			
	POX	Ryu	ONOS	HyperFlow	POX	Ryu	ONOS	HyperFlow	POX	Ryu	ONOS	HyperFlow	POX	Ryu	ONOS	HyperFlow	POX	Ryu	ONOS	HyperFlow	POX	Ryu	ONOS	HyperFlow
OpenDayLight	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			
	ODL	ONOS	POX	HyperFlow	ODL	ONOS	POX	HyperFlow	ODL	ONOS	POX	HyperFlow	ODL	ONOS	POX	HyperFlow	ODL	ONOS	POX	HyperFlow	ODL	ONOS	POX	HyperFlow
Ryu	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	12.45	20.45	22.45	12.44	14.44	20.44	28.44

The studied critical metrics of bandwidth, throughput, round-trip time, packet loss, delay, and jitter are extremely relevant to real-world network deployments. In order to meet the demands of different applications, bandwidth has a direct influence on the quantity of data that can be carried in a given length of time. A network's ability to efficiently use its available bandwidth is reflected in its throughput. In real-time applications like online gaming, financial transactions, and audio and video communication, network latency and dependability are evaluated in part by round-trip time, packet loss, delay, and jitter. In order to guarantee that network performance satisfies the requirements of a variety of applications and user expectations in real-world deployments, it is imperative to monitor metrics such as high packet loss, delay, or jitter. These phenomena can negatively impact user experience and quality of service.

A controller's efficacy and usefulness in practical deployments are directly impacted by how well it performs in a particular set of metrics. For example, slow network performance might be caused by poor throughput or insufficient bandwidth, which makes it more difficult to provide data and services on time. Poor user experiences and interruptions can result from real-time applications like online gaming and video conferencing having high round-trip times, packet loss, delay, or jitter. Reliability is crucial in situations like financial transactions and healthcare applications, where even small packet loss or delay can have a big impact. As a result, a controller's capacity to maximize these metrics directly affects the network's capacity to satisfy the requirements of various applications and user expectations in practical contexts, which in turn determines the general quality and dependability of the system.

The findings from performance analysis are indeed limited to the specific experimental setup and conditions. A controller's performance can vary significantly depending on factors such as network size, topology, traffic patterns, and other environmental variables. Generalizing the findings requires careful consideration of these factors and their potential impacts. The scope of our experiment using custom topologies to evaluate bandwidth, throughput, round-trip time, packet loss, delay, and jitter is to assess the controller's performance under controlled conditions. However, the limitation lies in the potential lack of representativeness of real-world network scenarios. While a custom topology allows for controlled experimentation, it may not fully capture the complexities and dynamics of actual network environments. Therefore, the generalizability of our findings to real-world deployments may be limited, and further validation in diverse network settings is necessary for comprehensive insights.

A variety of factors can have an impact on an SDN controller's performance. When the number of actions or switches varies or rises, as well as the amount of stress that is applied, performance heavily depends on the selection of controller. 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 [4,53,65,82,85,91] in their research because of its simplicity of use and Python-based scripting. Conversely, using OpenDayLight controller was recommended by the authors of papers [7,32,56,77,79]. However, paper [30] 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. Advantages of Logically Centralized Controllers

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

- Higher-level policies: Rather than using network identifiers, language used to describe policies is based on principles.
- Paths should be determined by policy: Depending 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. These 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 receive critical updates as soon as possible. Maintaining steady and high-caliber performance over time may be challenging, but it is not impossible [95]. At greater synchronization and overhead rates, strong consistency ensures that all dispersed controllers have access to the same network data. According to Levin et al. [96], 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]. In 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 the domain state among the participating controllers. In the paper by Jyotish et al. [97], 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 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 [95]. Without a consistent northbound or eastbound interface, it is also impossible to communicate between applications 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 [24] 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 computation-intensive applications (route calculation) to suffer. When latency-sensitive and computation-intensive applications were co-located in the same controller, the authors of Chang et al. [98] found it challenging to achieve quick response and convergence times.

The recommended separation of several applications makes use of 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 undertake more research.

Problems with load balancing and controller deployment: An SDN controller's integration with a 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 (enterprises), load balancing and fault tolerance receive more attention from researchers. 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 [23,29].

Security: Security can be compromised throughout the SDN network due to the dispersed SDN controllers' complete network intelligence. The SDN network may lose access to the control plane if the safety problem is not resolved on a big scale. A 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 [9,10].

7. Conclusions

A thorough analysis of SDN controllers and their classification is summarized in this paper. Testing of many OpenFlow-based distributed (but logically centralized) SDN controllers, including OpenDayLight, ONOS, POX, Ryu, and HyperFlow, was included in the experimental research. The topologies used for the experiments were custom-generated for them. The six parameters that were 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 can increase or decrease, and a combination of both were 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 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, while Ryu should be used when the system 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 against the network. After that, we will 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. and 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. and 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.

Abbreviations

5G	Fifth Generation
ASIC	Application Specific Integrated Circuit
AHP	Analytic Hierarchy Process
API	Application Programming Interface
CLI	Command Line Interface
CPU	Central Processing Unit
Cbench	Connection Benchmark
DCN	Data-Centric Network
EWBI	East West Bound Interface
EPL	Eclipse Public License
GUI	Graphical User Interface
IETF	Internet Engineering Task Force
ICMP	Internet Control Message Protocol
MCDM	Multi Criteria Decision Making
MFT	Mean Free Time
Ms	Milliseconds
Mbps	Megabytes per second
NS3	Network Simulator 3
NTT	Nippon Telegraph and Telephone
NETCONF	Network Configuration Protocol
NOX	Network Operating System
NBI	North Bound Interface
NETCONF	Network Configuration Protocol
NFV	Network Functions Virtualization
ODL	OpenDayLight
OF	OpenFlow
ONOS	Open Networking Operating System
OS	Operating System
OSGi	Open Service Gateway initiative
OF-config	OpenFlow Configuration and Management Protocol
OVSDB	Open vSwitch Database
ONOS	Open Network Operating System
POX	Pythonic Network Operating System
PC	Personal Computer
PING	Packet Internet Groper
QoS	Quality of Service
REST	Representational State Transfer
RESTCONF	RESTful Network Configuration Protocol
RTT	Round-Trip Time
RAM	Random Access Memory
ROIA	Real-Time Online Interactive Applications
SDN	Software Defined Networking
SD-IoT	Software Defined Internet of Thing
SD-WAN	Software Defined Wireless Access Network
SAT	Satellite
SBI	South Bound Interface
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UI	User Interface
WAN	Wireless Access Network
XMPP	Extensible Messaging and Presence Protocol

References

1. Gautam, P.; Jha, S.; Yadav, C.S. Performance Comparison of the Proposed OpenFlow Network with the Pox Controller and the Traditional Network in Software Defined Networks (SDN). In Proceedings of the Artificial Intelligence and Communication Technologies; Hiranwal, S., Mathur, G., Eds.; Routledge: London, UK, 2022; pp. 537–552. [CrossRef]
2. Zhu, L.; Karim, M.M.; Sharif, K.; Li, F.; Du, X.; Guizani, M. SDN controllers: Benchmarking & performance evaluation. *arXiv* **2019**, arXiv:1902.04491.
3. Nisar, K.; Jimson, E.R.; Hijazi, M.H.A.; Welch, I.; Hassan, R.; Aman, A.H.M.; Sodhro, A.H.; Pirbhulal, S.; Khan, S. A survey on the architecture, application, and security of software defined networking: Challenges and open issues. *Internet Things* **2020**, *12*, 100289. [CrossRef]
4. Bhardwaj, S.; Panda, S.N. Performance evaluation using Ryu SDN controller in software- defined networking environment. *Wirel. Pers. Commun.* **2022**, *122*, 701–723. [CrossRef]
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, Edmonton, AB, Canada, 8–11 September 2014; pp. 402–405. [CrossRef]
6. Keshari, S.K.; Kansal, V.; Kumar, S. A systematic review of quality of services (QoS) in software defined networking (SDN). *Wirel. Pers. Commun.* **2021**, *116*, 2593–2614. [CrossRef]
7. Ali, J.; Roh, B.H.; Lee, S. QoS improvement with an optimum controller selection for software-defined networks. *PLoS ONE* **2019**, *14*, e0217631. [CrossRef]
8. 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), Bhilai, India, 19–20 February 2021; pp. 1–13.
9. Bhat, D.; Rao, K.; Latha, N. A survey on software-defined networking concepts and architecture. *Int. J. Sci. Technol. Manag.* **2015**, *4*, 132–141.
10. Xia, W.; Wen, Y.; Foh, C.H.; Niyato, D.; Xie, H. A survey on software-defined networking. *IEEE Commun. Surv. Tutor.* **2014**, *17*, 27–51. [CrossRef]
11. Amin, R.; Reisslein, M.; Shah, N. Hybrid SDN networks: A survey of existing approaches. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 3259–3306. [CrossRef]
12. 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), Dhaka, Bangladesh, 3–5 May 2019; pp. 1–5.
13. 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), San Jose, CA, USA, 24 April 2012; pp. 1–6.
14. Shalimov, A.; Zuikov, D.; Zimarina, D.; Pashkov, V.; Smeliansky, R. Advanced study of SDN/OpenFlow controllers. In Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia, Moscow, Russia, 24–25 October 2013; pp. 1–6.
15. 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), Hammamet, Tunisia, 17–19 January 2014; pp. 1–7.
16. 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, Istanbul, Turkey, 25–29 April 2016; pp. 889–894.
17. 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), Barcelona, Spain, 25–28 March 2013; pp. 1009–1016.
18. 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), Budapest, Hungary, 9–13 June 2013; pp. 3504–3508.
19. Erickson, D. The beacon openflow controller. In Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking; 2013; pp. 13–18.
20. Khorramizadeh, M.; Ahmadi, V. Capacity and load-aware software-defined network controller placement in heterogeneous environments. *Comput. Commun.* **2018**, *129*, 226–247. [CrossRef]
21. 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. *J. Syst. Softw.* **2018**, *145*, 180–194. [CrossRef]
22. Jalili, A.; Keshtgari, M.; Akbari, R.; Javidan, R. Multi criteria analysis of controller placement problem in software defined networks. *Comput. Commun.* **2019**, *133*, 115–128. [CrossRef]
23. Karakus, M.; Durresi, A. A survey: Control plane scalability issues and approaches in software- defined networking (SDN). *Comput. Netw.* **2017**, *112*, 279–293. [CrossRef]

24. Bannour, F.; Souihi, S.; Mellouk, A. Distributed SDN control: Survey, taxonomy, and challenges. *IEEE Commun. Surv. Tutor.* **2017**, *20*, 333–354. [CrossRef]
25. Abuarqoub, A. A review of the control plane scalability approaches in software defined networking. *Future Internet* **2020**, *12*, 49. [CrossRef]
26. Zhang, Y.; Cui, L.; Wang, W.; Zhang, Y. A survey on software defined networking with multiple controllers. *J. Netw. Comput. Appl.* **2018**, *103*, 101–118. [CrossRef]
27. Benamrane, F.; Benaini, R. Performances of OpenFlow-based software-defined networks: An overview. *J. Netw.* **2015**, *10*, 329. [CrossRef]
28. Isong, B.; Molose, R.R.S.; Abu-Mahfouz, A.M.; Dladlu, N. Comprehensive review of SDN controller placement strategies. *IEEE Access* **2020**, *8*, 170070–170092. [CrossRef]
29. 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 Comput. Surv.* **2020**, *53*, 1–40. [CrossRef]
30. Mostafavi, S.; Hakami, V.; Paydar, F. Performance evaluation of software-defined networking controllers: A comparative study. *Comput. Knowl. Eng.* **2020**, *2*, 63–73.
31. 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), Cairo, Egypt, 15–16 December 2020; pp. 1–6.
32. Gupta, N.; Maashi, M.S.; Tanwar, S.; Badotra, S.; Aljebreen, M.; Bharany, S. A comparative study of software defined networking controllers using Mininet. *Electronics* **2022**, *11*, 2715. [CrossRef]
33. 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), Montreal, QC, Canada, 15–19 September 2014; pp. 1–6.
34. Salman, O.; Elhajj, I.H.; Kayssi, A.; Chehab, A. SDN controllers: A comparative study. In Proceedings of the 2016 18th Mediterranean Electrotechnical Conference (MELECON), Limassol, Cyprus, 18–20 April 2016; pp. 1–6.
35. Paliwal, M.; Shrimankar, D.; Tembhurne, O. Controllers in SDN: A review report. *IEEE Access* **2018**, *6*, 36256–36270. [CrossRef]
36. Sakellaropoulou, D. A Qualitative Study of SDN Controllers. Master's Thesis, Athens University of Economics and Business, Athina, Greece, 2017.
37. 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), Caparica/Lisbon, Portugal, 7 July 2023; pp. 6–11.
38. Mamushiane, L.; Lysko, A.; Dlamini, S. A comparative evaluation of the performance of popular SDN controllers. In Proceedings of the 2018 Wireless Days (WD), Dubai, United Arab Emirates, 3–5 April 2018; pp. 54–59.
39. 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, 12–14 March 2012*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 85–95.
40. 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, Budapest, Hungary, 1–3 September 2014; pp. 127–128.
41. Veryx Technologies. PktBlaster SDN Controller Test. Available online: http://sdn.veryxtech.com/ (accessed on 15 January 2024).
42. Jarschel, M.; Lehrieder, F.; Magyari, Z.; Pries, R. A flexible OpenFlow-controller benchmark. In Proceedings of the 2012 European Workshop on Software Defined Networking, Darmstadt, Germany, 25–26 October 2012; pp. 48–53.
43. Carneiro, G.J. NS3: Network Simulator 3. Available online: https://www.nsnam.org/tutorials/NS-3-LABMEETING-1.pdf (accessed on 15 January 2024).
44. 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 2014 IEEE Colombian Conference on Communications and Computing (COLCOM), Bogota, Colombia, 4–6 June 2014; pp. 1–6.
45. 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, Marrakech, Morocco, 15–17 May 2014; Springer: Berlin/Heidelberg, Germany, 2014; pp. 330–334.
46. 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 2014 IEEE Colombian Conference on Communications and Computing (COLCOM), Bogota, Colombia, 4–6 June 2014; pp. 1–6.
47. 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 Third Workshop on Hot Topics in Software Defined Networking, Chicago, IL, USA, 17–22 August 2014; pp. 1–6.
48. 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), Nis, Serbia, 14–17 October 2015; pp. 223–226.

49. 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), Bandung, Indonesia, 25–26 November 2015; pp. 1–6.
50. 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, Kuala Lumpur, Malaysia, 9–12 February 2015; pp. 205–210.
51. Rowshanrad, S.; Abdi, V.; Keshtgari, M. Performance evaluation of SDN controllers: FloodLight and OpenDayLight. *IIUM Eng. J.* **2016**, *17*, 47–57. [CrossRef]
52. Jasim, A.; Hamid, D. Enhancing the performance of OpenFlow network by using QoS. *Int. J. Sci. Eng. Res.* **2016**, *7*, 950–955.
53. Kaur, K.; Kaur, S.; Gupta, V. Performance analysis of python based openflow controllers. In Proceedings of the 3rd International Conference on Electrical, Electronics, Engineering Trends, Communication, Optimization and Sciences (EEECOS 2016), Washington, DC, USA, 9–11 February 2016; pp. 1–4.
54. Bholebawa, I.Z.; Dalal, U.D. Design and performance analysis of OpenFlow-enabled network topologies using Mininet. *Int. J. Comput. Commun. Eng.* **2016**, *5*, 419–429. [CrossRef]
55. Rastogi, A.; Bais, A. Comparative analysis of software defined networking (SDN) controllers—In terms of traffic handling capabilities. In Proceedings of the 2016 19th International Multi-Topic Conference (INMIC), Islamabad, Pakistan, 5–6 December 2016; pp. 1–6.
56. 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), Wuhan, China, 13–15 October 2016; pp. 207–210.
57. Bholebawa, I.Z.; Jha, R.K.; Dalal, U.D. Performance analysis of proposed OpenFlow-based network architecture using Mininet. *Wirel. Pers. Commun.* **2016**, *86*, 943–958. [CrossRef]
58. Fancy, C.; Pushpalatha, M. Performance evaluation of SDN controllers POX and FloodLight in Mininet emulation environment. In Proceedings of the 2017 International Conference on Intelligent Sustainable Systems (ICISS), Palladam, India, 7–8 December 2017; pp. 695–699.
59. 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), Mysuru, India, 15–16 December 2017; pp. 1–5.
60. Research, P.J.; 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), Bangalore, India, 1–2 February 2018; pp. 1–8.
61. 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, 14–16 December 2017*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 913–923.
62. 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), Dhaka, Bangladesh, 22–24 December 2017; pp. 1–6.
63. Abdullah, M.Z.; Al-Awad, N.A.; Hussein, F.W. Performance Comparison and Evaluation of Different Software Defined Networks Controllers. *Int. J. Comput. Netw. Technol.* **2018**, *6*, 1–7. [CrossRef]
64. Bholebawa, I.Z.; Dalal, U.D. Performance analysis of SDN/OpenFlow controllers: POX versus FloodLight. *Wirel. Pers. Commun.* **2018**, *98*, 1679–1699. [CrossRef]
65. Ali, J.; Lee, S.; Roh, B.H. Performance analysis of POX and Ryu with different SDN topologies. In Proceedings of the 1st International Conference on Information Science and Systems, Jeju, Republic of Korea, 27–29 April 2018; pp. 244–249.
66. 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), Bangalore, India, 1–2 February 2018; pp. 1–5.
67. Jawaharan, R.; Mohan, P.M.; Das, T.; Gurusamy, M. Empirical evaluation of sdn controllers using Mininet/Wireshark and comparison with cbench. In Proceedings of the 2018 27th International Conference on Computer Communication and Networks (ICCCN), Hangzhou, China, 30 July–2 August 2018; pp. 1–2.
68. Sameer, M.; Goswami, B. Experimenting with ONOS scalability on software defined network. *J. Adv. Res. Dyn. Control Syst.* **2018**, *10*, 1820–1830.
69. 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. *Int. J. Comput. Netw. Commun.* **2018**, *10*, 27–39. [CrossRef]
70. 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. *Int. J. Sci. Eng. Res.* **2018**, *9*, 1753–1758.
71. Shamim, S.; Shisir, S.; Hasan, A.; Hasan, M.; Hossain, A. Performance analysis of different open flow based controller over software defined networking. *Glob. J. Comput. Sci. Technol.* **2018**, *18*, 11–15.

72. Altangerel, G.; Chuluuntsetseg, T.; Yamkhin, D. Performance analysis of SDN controllers: POX, FloodLight and OpenDayLight. *arXiv* **2021**, arXiv:2112.10387.
73. 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), Pattaya, Thailand, 10–11 October 2019; pp. 1–4.
74. 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), Denpasar, Indonesia, 21–22 September 2019; pp. 51–55. [CrossRef]
75. 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), Bangalore, India, 17–18 May 2019; pp. 752–756. [CrossRef]
76. Manuel, T.; Goswami, B. Experimenting with scalability of beacon controller in software defined network. *Int. J. Recent Technol. Eng.* **2019**, *7*, 550–555.
77. Badotra, S.; Panda, S.N. Evaluation and comparison of OpenDayLight and open networking operating system in software-defined networking. *Clust. Comput.* **2019**, *23*, 1281–1291. [CrossRef]
78. Sheikh MN, A.; Halder, M.; Kabir, S.S.; Miah, M.W.; Khatun, S. SDN-Based approach to evaluate the best controller: Internal controller NOX and external controllers POX, ONOS, Ryu. *Glob. J. Comput. Sci. Technol.* **2019**, *19*, 21–32. [CrossRef]
79. Pramudita, A.; Suartana, I. Perbandingan Performa Controller OpenDayLight dan Ryu pada Arsitektur Software Defined Network. *J. Inform. Comput. Sci. (JINACS)* **2020**, *1*, 174–178. [CrossRef]
80. Noman, H.; Jasim, M. POX controller and open flow performance evaluation in software defined networks (sdn) using Mininet emulator. *IOP Conf. Ser. Mater. Sci. Eng.* **2020**, *881*, 012102. [CrossRef]
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), Chongqing, China, 9–11 August 2020; pp. 85–90. [CrossRef]
82. Islam, M.T.; Islam, N.; Refat, M.A. Node to node performance evaluation through Ryu SDN controller. *Proc. Wirel. Pers. Commun.* **2020**, *112*, 555–570. [CrossRef]
83. Silva, J.B.; Silva FS, D.; Neto, E.P.; Lemos, M.; Neto, A. Benchmarking of mainstream SDN controllers over open off-the-shelf software-switches. *Internet Technol. Lett.* **2020**, *3*, e152. [CrossRef]
84. Eltaj, A.; Hassan, A. Performance Evaluation of SDN Controllers: FloodLight, POX and NOX. *Proc. Int. J. Eng. Appl. Sci.* **2020**, *7*, 16–43. [CrossRef]
85. Abidin, N.Z.; Fiade, A.; Aripiyanto, S.; 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), Bengkulu, Indonesia, 22–23 September 2021; pp. 1–5. [CrossRef]
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), Bhilai, India, 19–20 February 2021; pp. 1–9. [CrossRef]
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), Greater Noida, India, 19–20 February 2021; pp. 959–965. [CrossRef]
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), Taipei, Taiwan, 7–8 October 2021; pp. 24–25. [CrossRef]
89. Bhattacharyya, A.; Prakasam, 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), Kottayam, India, 29–31 July 2022; pp. 1–5. [CrossRef]
90. Elmoslemany, M.M.; Tag Eldien, A.S.; Selim, M.M. Performance Analysis in Software Defined Network (SDN) Multi-Controllers. *Proc. Delta Univ. Sci. J.* **2023**, *6*, 181–192. [CrossRef]
91. Naim, N.; Imad, M.; Hassan, M.A.; Afzal, M.B.; Khan, S.; Khan, A.U. POX and Ryu Controller Performance Analysis on Software Defined Network. *EAI Endorsed Trans. Internet Things* **2023**, *9*, e5. [CrossRef]
92. Ram, A.; Dutta, M.P.; Chakraborty, S.K. A Flow-Based Performance Evaluation on RYU SDN Controller. *J. Inst. Eng. Ser. B* **2024**, 1–13. [CrossRef]
93. Koulouras, I.; Bobotsaris, I.; Margariti, S.V.; Stergiou, E.; Stylios, C. Assessment of SDN Controllers in Wireless Environment Using a Multi-Criteria Technique. *Information* **2023**, *14*, 476. [CrossRef]
94. Ganjali, Y.; 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), San Jose, CA, USA, 27 April 2010; pp. 1–3.
95. Ahmad, S.; Mir, A.H. SDN interfaces: Protocols, taxonomy and challenges. *Int. J. Wirel. Microw. Technol.* **2022**, *2*, 11–32. [CrossRef]
96. Levin, D.; Wundsam, A.; Heller, B.; Handigol, N.; 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, Helsinki, Finland, 13 August 2012; pp. 1–6. [CrossRef]

97. Jyotish, N.K.; Singh, L.K.; Kumar, C. A state-of-the-art review on performance measurement petri net models for safety critical systems of NPP. *Ann. Nucl. Energy* **2022**, *165*, 108635. [CrossRef]
98. Chang, Y.; Rezaei, A.; Vamanan, B.; Hasan, J.; Rao, S.; 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), Bengaluru, India, 4–8 January 2017; pp. 251–258. [CrossRef]