ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal
Regular Issue, Vol. 13 (2024), e31674
eISSN: 2255-2863
DOI: https://doi.org/10.14201/adcaij.31674

# Performance Analysis of Software-Defined Networking in Band Controllers for Different Network Topologies

Hussein Ali Al-Gubouri

Iraqi Ministry of Education, Directorate of Nineveh Education, Vocational Education, Nineveh, Iraq
✉ hussein.mohammed@uoninevah.edu.iq

KEYWORDS

*SDN; in band controllers; SDN topologies; SDN controllers*

ABSTRACT

*With the great increase in the complexity of networking, software-defined networks have been developed to help administrators operate and configure network services with controllers such as Pox, Ryu, Floodlight and OpenDaylight. Those controllers offer an appropriate platform for applications that need high bandwidth. In this paper, several SDN controllers have been evaluated using in-band communication mode with different network topologies to check the performance of the in-band controllers. Some controllers cannot operate with an in-band controller such as Pox. The controllers were evaluated with Mininet by using iperf and ping networking tools, the packet latency round trip time RTT and the comparison of the throughput of the three topologies. The results of the experiments showed that in-band controllers can be implemented and have efficient results. Results showed that OpenDaylight has the lower value of RTT so it is the best for the applications that need fast response. Ryu has a greater bandwidth value, so it is the best for applications that need high bandwidth. Floodlight comes third in order, after OpenDaylight and Ryu, respectively.*

## 1. Introduction

The software defined networking (SDN) technology refers to a method for computer-networking function organization. It makes it possible for the network to be programmable and virtualized. The proposed method by SDN technology is to move the intelligence of the network from the forwarding

component (routers and switches) then put it in the logically central controller. SDN paradigm and its standards were recently established to be more flexible, cost-effective, programmable, scalable and vendor-agnostic networking by decoupling forwarding devices from control logic such as routers and switches. Those allow for centralized network programming and enforcement of the policy, which remarkably facilitates the management of networks (Abdullah et al., 2018). Figure 1 illustrates the SDN functional architecture and its components, highlighting the interactions among the layers and their roles in simplifying network management and enhancing scalability.

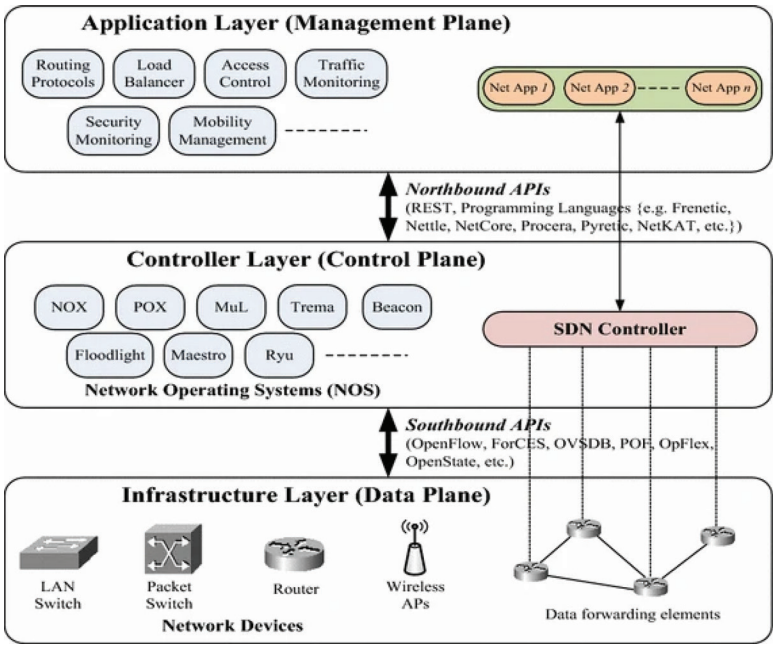



*Figure 1. SDN functional architecture and its components (Bholebawa & Dalal, 2018)*

The SDN architecture is multi-layered, consisting of three layers, namely, the control plane, the application layer, and the data plane which contains the forwarding equipment, such as routers, switches and firewalls. Meanwhile, the control plane is where the brain of the network controller resides. The application layer comprises several applications that describe diverse policies such as network security and traffic engineering. Communication, among all layers, is accomplished through two types of interfaces called the North Bound API, NBAPI, and the South Bound API, SBAPI (Mostafavi et al., 2020). Sometimes, there is a requirement to define the network's behaviour in a customized mode. This was potential only through using exclusive devices that were too expensive or difficult for experimenters and researchers to access. The requirement is for that functionality to exist so that large projects can be run and novel protocols executed. Thus, in the area of networking, the technology of OpenFlow, It allows network control through software and programs, managed by servers known as SDN controllers, and has gained significant interest from both industry and research communities. The OpenFlow technique was developed and started by the University of Stanford, whose goal was to allow programmable networking to experiment with novel protocols on the platforms of the Internet. The

fundamental idea of the SDN OpenFlow architecture is to decouple network switches from the control plane and embed them in servers called controllers. That makes routers and switches cheap. Furthermore, this imparts the flexibility of networks, as the control plane functionality moved to the centralized controllers, whereas the forwarding needed to be accomplished by hardware devices. OpenFlow technique is dependent on the fact that new switches and routers contain a patented forwarding information base (FIB) that is applied in the forwarding hardware by means of ternary content addressable memory (TCAMs).

SDN OpenFlow paradigm offers a flow table conception that is an FIB abstraction. Moreover, it offers a protocol for programming the FIB by deleting, adding and modifying open flow entries in the flow table. That is accomplished through using controllers that connect with the SDN switches by using the SDN.

The router/switch that exposes the flow table through the protocol of OpenFlow is named OpenFlow router/switch. The entry in the flow table comprises a set of packet fields that match it to incoming packets (named the flow), statistics for each flow that keep track of the matching packets, as well as actions that specify how the packets should be handled. When an OpenFlow switch/router receives a packet, it is compared to the flow table's entries. If there is a match, then the steps outlined in the matching entry are taken. If there is no match, then the packet is forwarded to the central controller. Subsequently, the SDN controller decides how to treat the packet. The controller may back the packet into a forwarding switch that represents the forwarding port of the switch, or it might add a flow entry so as to direct the OpenFlow switch on how to process packets with a similar flow. With SDN, control messages (the messages for adding flow entries into SDN switches) are needed to interchange between the switches and controller. Those messages could exchange in an out-of-band mode or an in-band mode. In in-band communication, the control messages are sent on the same communication channel

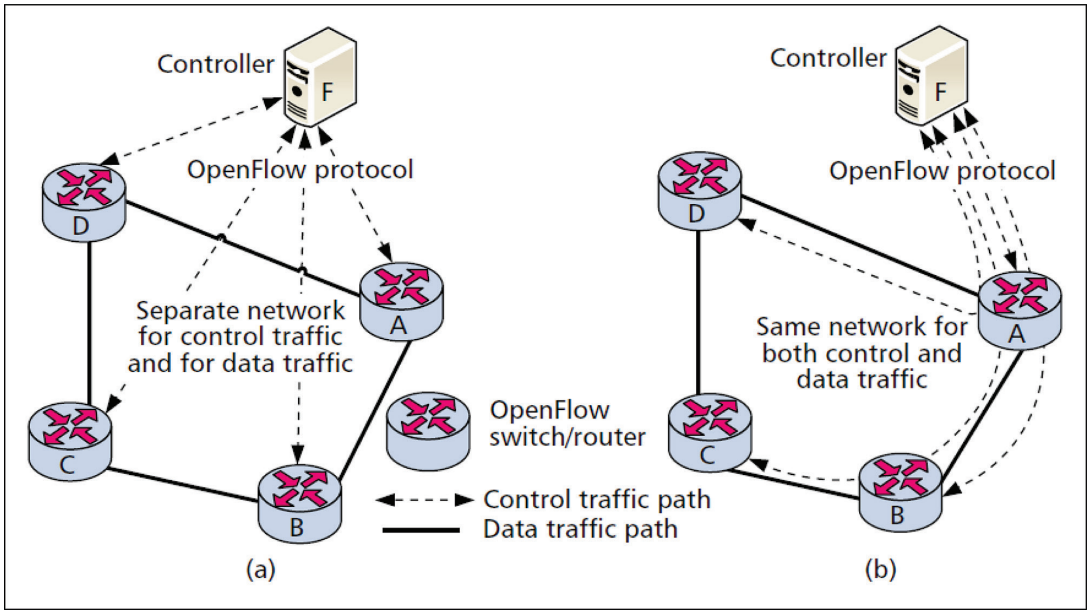



*Figure 2. SDN network showing (a) out-of-band mode, and (b) in-band mode (Sharma et al., 2013b, 2016)*

that is used to transfer data, while, in out-of-band communication, the control messages are sent on another communication channel. In in-band communication (Figure 2), switches (A, B, C and D) share the communication channel for data traffic and control messages, and in the out-of-band communication, each switch (A, B, C and D) uses a different communication channel for data traffic and control (Sharma et al., 2013a).

The out-of-band communication channel is easier and simpler to design and implement because the SDN controller is connected directly (physically) to each one of the switches. Despite this, in some scenarios, an out-of-band communication mode may not be possible, for example, broadly dispersed centralized offices in access networking. Furthermore, out-of-band communication mode is costly to implement in a real network because it requires an additional physical port for each switch. In contrast, in-band communication does not require extra physical ports for controlling traffic, making it more economical. SDN OpenFlow defines and describes a virtual port in SDN switch, which is named a local port, that enables the remote units (SDN controller) to communicate with SDN switch by an SDN OpenFlow protocol networking (in-band communicate mode) (Sharma et al., 2013a).

In this paper, we use three topologies (single, linear tree) to check the performance of in-band controllers. In the in-band network, the controller starts its own control networking over the switches, which are linked to the controller through the SDN OpenFlow network protocol. The network emulation is done by using three types of topologies, each topology has a number of switches and hosts, and the controller resides in the host (h3), for tree topology (the depth=4, fanout=2). The emulation results show that the OpenDaylight is the best relative to RTT (fast response) and Ryu is the best relative to bandwidth.

The rest of the paper is structured as follows: Section 2 introduces the research methodology, Section 3 reviews the related works, Section 4 describes the emulation environment and SDN controllers, Section 5 outlines the test bed setup, Section 6 presents the performance results, Section 7 draws conclusions, and finally, Section 8 puts forward a series of recommendations.

## 2. Research Methodology

The study has been performed to find details and information about the in-band controllers over SDN. Then, the needed tools and the software selected for the different controllers' implementation was studied. The software tools were selected for the experiments. After that, the experimental setup was designed which comprised the software installation and (Mininet and controllers) configurations. There are many software tools available for the accomplishment of SDN networks, among them Maxinet(MaxiNet: Distributed Network Emulation, n.d.), OFNet(OFNet SDN Network Emulator | Open-Source Routing and Network Simulation, n.d.), NS-3(Ns-3 | a Discrete-Event Network Simulator for Internet Systems, n.d.), OMNET++ (OMNeT++ Discrete Event Simulator, n.d.), EstiNet(EstiNet - Simulator | EstiNet, n.d.) And Mininet(Mininet: An Instant Virtual Network on Your Laptop (or Other PC) - Mininet, n.d.,SDN101:). For emulating Software Defined Networks there are an open-source networking Emulator Mininet has good ability. Mininet is a widely used network emulator. It can implement an extensive network with a collection of network elements, such as hosts, switches and routers based on Linux kernel. Complex network topologies can be designed for virtualization using Mininet. Popular examples of SDN central controller developments are NOX, Open Daylight, POX, Floodlight, Beacon, Pyretic and Ryu. Ryu is an open source SDN controller which has been initiated and designed to increase the network agility, making it simple for the network to adapt the way in which traffic is performed and managed. Pyretic is a Python controller which operates on the

SDN control layer. In SDN, controllers separate from the switches. The separation of the forwarding switches from control allows for the management of highly complex traffic. The OpenFlow protocol enables access to the forwarding plane of switch or router over the networking (Shamim et al., 2018).

Three SDN controllers were used in our experimental evaluation (Open Daylight ODL, Ryu, Floodlight) as a remote controller and the topology type was changed every time (single, linear, tree). Then, the network parameters were measured (throughput, delay (RTT), and packet loss) for each case in order to evaluate the performance of the SDN in band controllers in the network.

A laptop with an Intel Core ™ i5 processor, 4GB of RAM, and the Ubuntu 18.0.4 LTS-64-bit operating system, one of the most popular Linux kernel distributions, was used to install and operate the Mininet and controllers. Throughput can be measured in both ways between the two ends of the network using the Iperf tool, a benchmarking tool with client and server functionality that can generate the transmission control protocol (TCP) or the user datagram protocol (UDP) traffic. Using the ping command to check connectivity between two stations (hosts) in the network, delay has also been calculated as round trip time (RTT) between the two furthest stations and packet loss.

Ping sends one or more internet control message protocol (ICMP) echo request packets to a certain network destination IP and waits for a response, when the packet arrives at its destination; it sends an ICMP echo reply.

## 3. Related Works

Recently, several works have been accomplished to compare the controllers, some of which are reviewed in this section. In (Shah et al., 2013), four SDN controllers comprising Beacon, NOX, Floodlight and Maestro were tested and compared. The study pointed out that there are four major performance bottlenecks to consider, including switch partitioning, multi-core support, batching task and packet batching. Table 1 provides a comparative summary of the methodologies, tools, and key findings of these related works, highlighting their contributions to the field. The architectural plans and designs, comprising static batching and static switch partitioning, are utilized by NOX-MT, Floodlight and Beacon, while adaptive batching and queue sharing are utilized by Maestro. The results of the evaluation performance showed that packet batching and switch static partitioning designs are good for SDN controllers in the high throughput networking. Task batching and packet batching designs are best for controllers which control plane applications and are delay sensitive. The results also show that sending control messages out individually can improve the latency performance. Based on these results, a controller was proposed whose performance was better than the compared controllers.

Sharma et al. (2016) analyzed in-band queuing, failure recovery functionalities and control, and they performed many experiments. The experiments of in-band conclude that the planned method permits bootstrapping in all the network topologies. In that method, the switches of performed-pan European topologies have occupied a maximum of five seconds to process bootstrapping. Experiments of queuing shows that control traffic in in-band mode could be firstly served before any traffic, and so it can eliminate race with the data-traffic for networking resources. The conclusion of the failure recovery experiments was that restoration in SDN OpenFlow does not meet the 50 ms protection and recovery requirement for both data and control traffic. According to the results, the authors did not consider propagation delay. As a result, the restoration time might increase due to propagation delay, making it impossible to meet the 50-millisecond deadline as future work, the effects of propagation delay can be studied to quantify the degradation of the restoration time with an increase in propagation delay.

*Table 1. Comparison among author's contribution*

| Aspect | Methodology | Sample | Tools and Techniques | Results |
|---|---|---|---|---|
| (Sharma et al., 2016) | Conducted experiments on in-band queuing, failure recovery, and control in a Pan-European network | Various network topologies in a Pan-European setting, bootstrapping delay | In-band queuing tools, failure recovery and control experiments | Bootstrapping completed in all topologies within five seconds, control traffic prioritized, but failure recovery may not meet 50 ms deadline |
| (Alrashedy et al., 2017) | Used Mininet, iperf, and ping for testing OpenFlow controllers | Multiple networks and combinations, including topologies with loops | Mininet, iperf, ping | Floodlight controller adaptable and efficient, others struggled with topologies containing loops |
| (Khondoker et al., 2014) | Used analytic hierarchy process (AHP) for multi-criteria comparison of five controllers | Five SDN controllers compared across various criteria | AHP (analytic hierarchy process), metrics for support, REST API, etc. | Ryu had the highest priority vector value, followed by OpenDaylight, Floodlight, POX, and Trema |
| (Shalimov et al., 2013) | Used Cbench for performance and scalability tests, hcprobe for security and reliability tests | Seven SDN controllers, evaluated on throughput, latency, reliability, security, and scalability | Cbench for performance, hcprobe for security tests | Beacon achieved highest throughput, Ryu best in security, Floodlight and Mul had lower latency |
| (Al-Somaidai, 2014) | Evaluated OpenFlow versions and SDN tools with various emulators and controllers | Various versions of OpenFlow, SDN tools, and emulators, seven SDN controllers | EstiNet, Mininet, Trema, NS-3, various SDN tools and emulators | OpenDaylight and Floodlight had the best flexibility and documentation |
| (Kaur et al., 2014) | Simulated SDN with POX and Mininet, compared features of five SDN controllers | Five SDN controllers (Ryu, Floodlight, POX, OpenDaylight, Trema) compared on six features | POX, Mininet, and comparison of SDN controller features | Compared five controllers on six features, POX tested |
| (Govindraj et al., 2012) | Analyzed OpenFlow switches in terms of load balancing and bandwidth efficiency | OpenFlow switches assessed for commercial and data center environments | Tools for evaluating OpenFlow switches' performance in real-world applications | OpenFlow provides better load balancing and bandwidth savings in commercial and data centers |
| This paper | Evaluating SDN controllers using Mininet, iperf, and ping; measured RTT and throughput across different topologies | Different network topologies with in-band communication mode; evaluated using Mininet, iperf, and ping | Mininet, iperf, ping for evaluating RTT and throughput of SDN controllers | OpenDaylight had the lowest RTT, Ryu had the highest bandwidth; Floodlight ranked third after OpenDaylight and Ryu |

Alrashedy et al. (2017) tested OpenFlow controllers and compared them using iperf and ping with the Mininet software. Several combinations of network topologies and controllers were taken into account. The results showed that the Floodlight controller can adapt to the topology and is efficient. It can take advantage of loops and many paths in the network topology. Some SDN controllers do poorly with topologies that contain loops; hence, they are inappropriate for modern networking. The OpenDaylight SDN controller can adapt to network topologies with loops but does not benefit the existence of many paths to develop the performance of the SDN.

In (Khondoker et al., 2014) , five SDN controllers, namely, Ryu, POX, Floodlight, OpenDaylight and Trema were compared to one another. According to Khondoker et al. (2014), selecting the controller in the SDN is a multi-criteria decision making (MCDM) problem because some properties of SDN controllers are significant for the users of the network. The researchers selected analytic hierarchy process (AHP) in management science, because of an integrated consistency checking mechanism and pair-wise prioritization (Khondoker et al., 2014). The five controllers were compared according to the available supports—virtual switching, interfaces, supporting REST API, graphical user interface (GUI), productivity, having documentation, being an open-source project, modularity, supporting language, age, TLS, platform, supporting OpenFlow and OpenStack networking. The results display that Ryu has best value of (0.287) in that priority vector, with OpenDaylight, Floodlight, Pox and Trema coming in second, third, and fourth. Floodlight and OpenDaylight (0.275, 0.265) had priority vector values that were extremely near to Ryu's controller.

Shalimov et al. (2013) tested and compared SDN controllers, among them POX, NOX, Beacon, Mul, Floodlight, Ryu and Maestro. For comparison, performance criteria such as throughput, bandwidth and latency, dependability, security and scalability were studied. The experiment of performance and scalability were accomplished with Cbench. Security and reliability tests were processed by hcprobe. In the findings of throughput experiments, with varied numbers of thread, hosts and switches, the Beacon controller got the maximum throughput. The time response of SDN controllers connected with (105) hosts, Beacon, Floodlight and Mul had minimum latency. The assessment of reliability among the controllers showed that, all the tested controllers, except for Maestro and Mul, were able to withstand the test load. For security, five experiments were executed, including invalid OpenFlow, improper message length, version, malformed packet-in message, incorrect OpenFlow message type and malformed port status message. Over the course of the five tests, Ryu fared best in these four scenarios.

Al-Somaidai et al. (2014) discussed several versions of the OpenFlow standard switch (1, 1.1, 1.2, 1.3 and 1.4). The authors considered diverse platforms for emulation and simulation of SDN, including EstiNet, Mininet, Trema, and NS-3, as well as seven kinds of SDN controllers, including POX, NOX, OpenDaylight, Floodlight, Ryu, Beacon and Mul. Varied switch tools and software were considered. The authors concluded that OpenDaylight and Floodlight were the SDN controllers with best flexibility and documentation.

To simulate an SDN, Kaur et al. (2014) employed a POX SDN controller and Mininet, and tested the network application's behavior in POX controller. In addition, the best five controllers, namely, Ryu, Floodlight, POX, OpenDaylight and Trema, were compared according to six features, including OpenFlow support, language support, having a GUI, being open-source, platform support and REST API.

According to the study of Govindraj et al. (2012), OpenFlow switching provides load balance and saves bandwidth waste when compared to other switches (because of the topology with no loops). Hence, this type of switch is more widely used in businesses and data centers. In addition, the authors concluded that the usability of OpenFlow could be increased. Numerous other SDN surveys (Kreutz et al., 2015; Lara et al., 2014; Nunes et al., 2014) are available.

While several studies have explored the performance of SDN controllers such as Pox, Ryu, Floodlight, and OpenDaylight, the majority have primarily focused on out-of-band communication modes or evaluated individual controllers under limited network conditions. The existing literature lacks a comprehensive comparative analysis of these controllers in in-band communication mode, especially across a variety of network topologies. This research addresses this gap by conducting a thorough performance evaluation of four key SDN controllers using different network topologies. Our approach leverages Mininet, iperf, and ping to measure critical performance metrics such as round-trip time (RTT) and throughput, offering new insights into controller performance under in-band communication scenarios that have been underexplored.

# 4. Ryu, OpenDaylight, Floodlight Controllers and Mininet

This study has been performed to obtain information about in-band controllers over SDN. Thus, it was first assessed what tools were needed and software was selected for the different controller's implementation. Software tools were selected for the experiments. After that, the experimental setup was designed which comprised software installation and (Mininet and controller) configurations. Many software tools are available to accomplish SD.

## 4.1. Ryu Controller

The Ryu SDN Controller is an open-source project that is licensed under the Apache 2.0 license and is entirely written in Python, deployed and supported by NTT data centers. Source code on GitHub, supported by the Ryu community. It supports OF-config network management protocols and NETCONF as well as SDN OpenFlow. Considering the compatibility, Hewlett Packard, IBM, OpenFlow switches and NEC are certified and tested with Ryu. It supports the SDN protocol to the modern version 1.5 (Asadollahi et al., 2018).

## 4.2. Floodlight Controller

Floodlight controller is built on University of Stanford Beacon controller and works with both real and virtual SDN switches. The features of this controller are as follows: Java-based, Apache-licensed, event-driven, modular, thread-based, synchronized lock and asynchronous application framework. Controller components are for the management of topology, used for the discovery of non-OpenFlow endpoints (LLDP).

The controller has REST APIs for accessing and setting the state of the controller, passing emitted events from Java Event Listeners and, event notification systems. In addition, this controller has model applications, including hub application, learning switch and static flow pusher, load balancer and Firewall (Rowshanrad et al., 2016).

## 4.3. OpenDaylight Controller

The OpenDaylight initiative was established as a Foundation of Linux project in 2013. This controller establishes a standard of northbound API that may be used to program various southbound protocols such as I2RS, NETCONF and OpenFlow. Features of OpenDaylight are modular, Java-based, supporting numerous southbound protocols and pluggable. This controller supports the

programming of a bidirectional OSGI framework and REST supports the applications that are executed in the same address space. A service abstraction handles both external and internal requests. A core service abstraction acts as the foundation for building higher-level services and is made accessible to the appropriate southbound plugin. The functionality of this layer depends on the specific capabilities of the plugins. Other built-in services in this controller include discovery and topology abstraction, PCE-P (CSPF), I2RS, OpenFlow and NETCONF (Rowshanrad et al., 2016).

## 4.4. Mininet Emulator

In this research, we evaluated the performance of software-defined networking (SDN) controllers using in-band communication across three different network configurations. Experiments were conducted using Mininet to simulate the networks, and to test the Ryu, OpenDaylight, and Floodlight controllers. The network was set up as listed in Table 2. The network emulation was done by using three types of topologies (single, linear, tree), the number of switches and hosts is shown in Table 2, and the controller resides in the host (h3), for tree topology (the depth=4, fanout=2). Performance was measured using multiple criteria such as response time, bandwidth, and packet loss rate. Ping tests were used to assess response time, and bandwidth tests were conducted using tools such as the iperf tool. To ensure the results are reproducible, each experiment was conducted, and the average, minimum, and maximum performance for each criterion were calculated. Each controller was set to default settings with minor adjustments to ensure compatibility with the experimental environment.

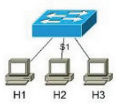
*Table 2. Number of switches and hosts in experiments*

| Topology | No. of Switches | No. of Hosts | Controller |
|---|---|---|---|
| Single | 1 | 12 | H3 |
| Linear | 16 | 32 | H3 |
| Tree | 15 | 16 | H3 |

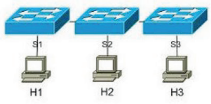
A network simulator has been used to generate virtual networks and SDN implementation by developers, lecturers, and academics. It was created using Mininet to execute virtual network controllers, hosts and switches on a single system. The main topology in Mininet contains kernel switch linked to two hosts and SDN controller. Hosts can run Linux commands such as «ipref», which checks the bandwidth and throughput between a server and a client, and «topo», which creates topologies through the python API for creating virtual networks (Rowshanrad et al., 2016).

In Mininet, there are common topologies which are presented in Figure 3 below:

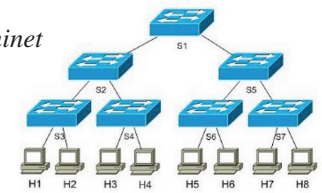
*Figure 3. Three predefined topologies in Mininet*




A- Single-topology.




B- Linear-topology.




C- Tree-topology.

Linear, tree and single topologies. There is just one controller, one switch, and an undetermined number of hosts in the single topology. A linear topology has serial connections between switches and hosts with a predetermined number of switches and hosts. Multiple topological levels with a defined number of levels make up a tree topology (depth and fan-out).

Mininet can also make use of remote controllers. Hence, in virtual machine (VM), virtual network devices are used to link to any remote SDN controller. Other benefits of Mininet are complex topology testing, support for system-level regression tests and command line interface (CLI) which is OpenFlow-aware and topology-aware. SDN controller code, a customized switch, or a Mininet host can be deployed in physical devices with minimal effort. As a result, for line-rate packet forwarding, a network plan in Mininet may be moved directly to hardware switches (Rowshanrad et al., 2016).

## 5. Test Bed Setup

Ubuntu 18.0.4 with 4 GB RAM was used to install and implement all network instances. For Mininet installation, installation on Ubuntu from source has been considered. Ryu, OpenDaylight and Floodlight all support OpenFlow protocol version 1.3 and OpenVswitch version 2.0.1. For each topology, a number of switches and hosts have been considered.

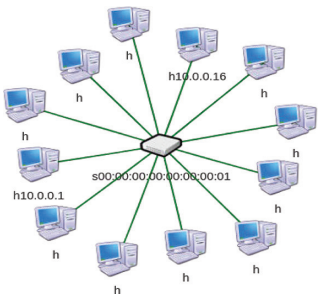



*Figure 4. Single Topology*

*Figure 5: Linear Topology*

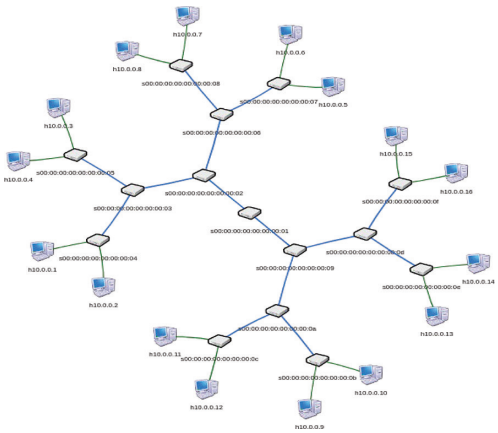


*Figure 6. Tree Topology*

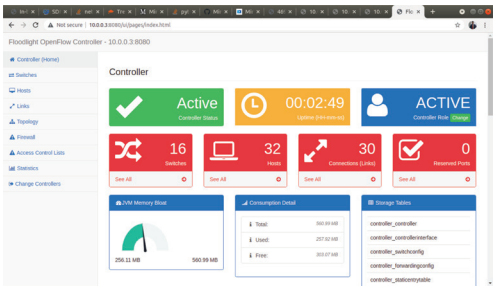



*Figure 7. Screenshot of running Floodlight Controller*

The Mininet Emulator was used to execute all of the simulations over software-defined networking. Mininet makes virtual hosts by using the network namespace mechanism and a process-based virtualization method, which is a function, supported in Linux version 2.2.26, for isolated routing tables, network interfaces, address resolution protocol (ARP) tables and routing tables for different virtual hosts. The network topologies used for testing in-band controller communication are shown below. Node (h3) hosts the controller. The details of the topologies used in the experiments are shown in Figures 4, 5, 6, and 7. Figure 4 represents the single topology, Figure 5 represents the linear topology, Figure 6 represents the tree topology, and Figure 7 shows a screenshot of the running Floodlight controller.

# 6. Performance Results

Ubuntu Experiments were conducted in which a ping command was sent on a specified path in the topology to check network connectivity. The command delivered packets to a distant host and then logged the round-trip time (RTT) of the packets. The iperf tool was used for throughput testing. The path used was h1 to h12 in the case of single topology, h1 to h16 in linear topology and h1 to h15 in the case of the tree topology, as it is the longest path in the network. One hundred pings were delivered, and the maximum, minimum and median times were recorded. Bandwidth testing was done using the iperf tool to provide the throughput of the network for each controller topology combination, and the maximum bandwidth. The tests were conducted using the common controllers, Ryu, Floodlight and OpenDaylight. The results of the ping command tests are given in Tables 3, 4 and 5. The OpenDaylight controller median and minimum ping times were always lower than those of the Ryu and Floodlight controllers. The median and minimum ping times for the three topologies were smaller with OpenDaylight than with other controllers. However, the maximum ping time with OpenDaylight was smaller than that achieved with other topologies.

The results show that the Ryu controller excels in environments with high bandwidth needs, while the OpenDaylight controller features lower response time. Ryu's high performance in high-bandwidth environments can be attributed to Ryu's packet processing efficiency. Comparisons with previous studies indicate that our results align with (Khondoker et al., 2014), who also found Ryu to exhibit excellent performance in certain scenarios. Although there is a lot of scientific research regarding the comparison between controllers in software-defined networks, the present scientific paper provides a unique contribution because it compares in band controllers in software-defined networks.

These results highlight the importance of choosing the appropriate controller based on the application's nature and network needs, which is essential for ensuring performance efficiency in SDN applications.

## 6.1. Delay (RTT)

As indicated in the previous part, RTT delay was calculated by creating a connection via ping command between the first station and the last station in the network. Of all of the different network topologies, OpenDaylight is the least delayed and outperforms Ryu and Floodlight Controllers.

## 6.2. Throughput

The iperf tool was used to assess TCP bandwidth for the three topologies, and the findings are presented in Table 5. The Ryu controller has the uppermost throughput with the three topologies, which

is 31 Gigabits per second (Gbps) for the single topology, 18.8 Gbps for the linear topology, and 19.5 Gbps for the tree topology. Regarding bandwidth, Floodlight controller came in second order, after Ryu and OpenDaylight came last. Thus, Ryu is the best in bandwidth performance.

Table 6 shows the bandwidth results (Gbps) for the three topologies using the iperf tool, comparing the performance of OpenDaylight, Ryu, and Floodlight controllers. Figures 8 and 9 illustrate the bandwidth (Gbps) and average RTT (ms) results for the three controllers across different topologies. In light of the results of the experiments, the Ryu controller demonstrates excellent performance in high-bandwidth environments but may be less efficient in environments requiring rapid response. On the other hand, the OpenDaylight controller shows a low response time, making it an ideal choice for time-sensitive applications, though its performance declines with increasing bandwidth requirements.

The Floodlight controller serves as a middle ground between Ryu and OpenDaylight, showing balanced performance in most scenarios but may not be the best choice in cases requiring very specific performance characteristics. It is important for practitioners to evaluate the strengths and weaknesses of each controller based on their specific needs and work environment to ensure optimal performance.

*Table 3. Ping Test Results of OpenDaylight*

| RTT | Single | Linear | Tree |
|---|---|---|---|
| Min(msec) | 0.041 | 0.102 | 0.092 |
| Avg.(msec) | 0.125 | 0.177 | 0.173 |
| Max(msec) | 0.681 | 1.992 | 2.186 |
| Mdev | 0.093 | 0.189 | 0.209 |
| Packet loss | 0 % | 0 % | 0 % |

*Table 4. Ryu Ping Test*

| RTT | Single | Linear | Tree |
|---|---|---|---|
| Min(msec) | 0.035 | 0.034 | 0.037 |
| Avg.(msec) | 0.164 | 0.199 | 0.200 |
| Max(msec) | 7.748 | 11.818 | 11.118 |
| Mdev | 0.762 | 1.167 | 1.097 |
| Packet loss | 0 % | 0 % | 0 % |

*Table 5. Floodlight Test Results*

| RTT | Single | Linear | Tree |
|---|---|---|---|
| Min(msec) | 0.030 | 0.089 | 0.115 |
| Avg.(msec) | 1.113 | 1.280 | 1.311 |
| Max(msec) | 104.03 | 95.713 | 97.936 |
| Mdev | 10.344 | 9.640 | 9.848 |
| Packet loss | 0 % | 0 % | 0 % |

*Table 6. Bandwidth Test (Gbps) using Iperf*

| Controller | Single | Linear | Tree |
|---|---|---|---|
| OpenDaylight | 20.1 | 16.4 | 15.3 |
| Ryu | 31.0 | 18.8 | 19.5 |
| Floodlight | 23.8 | 17.0 | 16.7 |

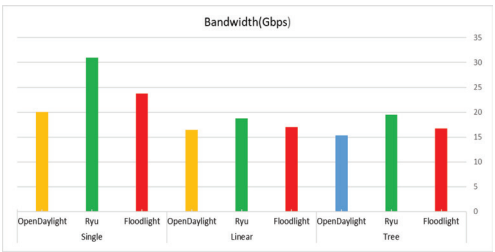



*Figure 8. Bandwidth (Gbps)*

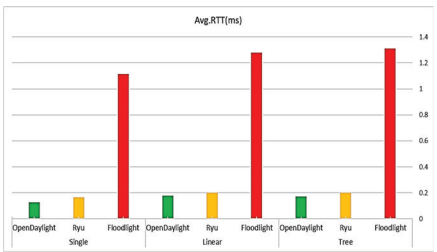



*Figure 9. Average RTT (ms)*

There are several limitations to consider when interpreting the results of this study. Firstly, the experiments were conducted in a simulated environment using Mininet, which may not accurately reflect performance in real networks. Performance may differ due to simulation environment constraints, such as limited availability of resources or variations in actual network characteristics. Secondly, the default settings used for the controllers may impact the results. Although the controllers were configured to fit the testing environment, these settings may not be optimal for all possible scenarios. Finally, there may be bias in data collection or analysis due to limitations in the number of nodes or data quality. Therefore, we recommend conducting additional experiments in real environments to validate and strengthen the results.

## 7. Conclusions

Software-defined networks (SDNs) are a new architecture for network service administration. In this research, Ryu, Floodlight and OpenDaylight SDN controllers, with in-band communications, were compared and tested by using iperf and ping commands. Mininet virtual networking was used to check the performance of the in-band controllers. Many combinations of network topologies and controllers were taken into account. The presented results show that the OpenDaylight SDN controller is efficient in terms of RTT. The Ryu controller is efficient in terms of applications that need high bandwidth. Pox cannot operate with a network that has in-band communication. The in-band controllers are efficient in the three-network topologies.

## 8. Recommendations

Considering our findings, we recommend using the Ryu controller for applications requiring high bandwidth, such as data centers and cloud computing applications. Conversely, the OpenDaylight controller is preferred for environments requiring low response times, such as IoT networks or real-time applications. For applications requiring a balance between performance, bandwidth, and response time, the Floodlight controller can be considered a suitable option. Practitioners should experiment with different settings and optimize the performance of controllers according to their specific needs.

## References


Abdullah, M. Z., Al-awad, N. A., & Hussein, F. W. (2018). Performance Comparison and Evaluation of Different Software Defined Networks Controllers. In *International Journal of Computing & Network Technology* (Vol. 06, Issue 02, pp. 36–41). https://doi.org/10.12785/ijcnt/060201

Alrashedy, K., Kimmett, B., & Gulliver, T. A. (2017). *Performance of Software-Defined Networking Controllers for Different Network Topologies*. *August*. https://doi.org/10.1109/PACRIM.2017.8121925

Al-Somaidai, M. B. (2014). Survey of Software Components to Emulate OpenFlow Protocol as an SDN Implementation. *American Journal of Software Engineering and Applications*, *3*(6), 74. https://doi.org/10.11648/j.ajsea.20140306.12

Asadollahi, S., Goswami, B., & Sameer, M. (2018). Ryu controller's scalability experiment on software defined networks. *2018 IEEE International Conference on Current Trends in Advanced Computing, ICCTAC 2018*, 1–5. https://doi.org/10.1109/ICCTAC.2018.8370397

Bholebawa, I. Z., & Dalal, U. D. (2018). Performance analysis of SDN/openflow controllers: POX versus floodlight. *Wireless Personal Communications*, *98*(2), 1679–1699. https://doi.org/10.1007/s11277-017-4939-z

*EstiNet - Simulator | EstiNet*. (n.d.). Retrieved August 7, 2021, from https://www.estinet.com/ns/

Govindraj, S., Jayaraman, A., Khanna, N., & Prakash, K. (2012). OpenFlow: Load Balancing in enterprise networks using Floodlight Controller. *Morse.Colorado.Edu*, 1–11.

Kaur, S., Singh, J., & Ghumman, N. S. (2014). *Network Programmability Using POX Controller*.

Khondoker, R., Zaalouk, A., Marx, R., & Bayarou, K. (2014, October 3). Feature-based comparison and selection of Software Defined Networking (SDN) controllers. *2014 World Congress on Computer Applications and Information Systems, WCCAIS 2014*. https://doi.org/10.1109/WCCAIS.2014.6916572

Kreutz, D., Ramos, F. M. V., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, *103*(1), 14–76. https://doi.org/10.1109/JPROC.2014.2371999

Lara, A., Kolasani, A., & Ramamurthy, B. (2014). Network innovation using open flow: A survey. *IEEE Communications Surveys and Tutorials*, *16*(1), 493–512. https://doi.org/10.1109/SURV.2013.081313.00105

*MaxiNet: Distributed Network Emulation*. (n.d.). Retrieved August 7, 2021, from https://maxinet.github.io/

*Mininet: An Instant Virtual Network on Your Laptop (or Other PC) - Mininet*. (n.d.). Retrieved August 7, 2021, from http://mininet.org/

Mostafavi, S. A., Hakami, V., & Paydar, F. (2020). Performance Evaluation of Software-Defined Networking Controllers: A Comparative Study. *Computer and Knowledge Engineering*, *2*(2), 63–73. https://doi.org/10.22067/CKE.V2I2.84917

*ns-3 | a discrete-event network simulator for internet systems*. (n.d.). Retrieved August 7, 2021, from https://www.nsnam.org/

Nunes, B. A. A., Mendonca, M., Nguyen, X. N., Obraczka, K., & Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys and Tutorials*, *16*(3), 1617–1634. https://doi.org/10.1109/SURV.2014.012214.00180

*OFNet SDN network emulator | Open-Source Routing and Network Simulation*. (n.d.). Retrieved August 7, 2021, from https://www.brianlinkletter.com/2016/11/ofnet-a-new-sdn-network-emulator/

*OMNeT++ Discrete Event Simulator*. (n.d.). Retrieved August 7, 2021, from https://omnetpp.org/?__cf_chl_jschl_tk__=pmd_c006c72e94d395f3446b6b2dcb1c8e8521460117-1628366263-0-gqNtZGzNAc2jcnBszQci

Rowshanrad, S., Abdi, V., & Keshtgari, M. (2016). Performance evaluation of sdn controllers: Floodlight and OpenDaylight. In *IIUM Engineering Journal* (Vol. 17, Issue 2, pp. 47–57). https://doi.org/10.31436/iiumej.v17i2.615

«SDN 101: Using Mininet and SDN Controllers». [Online]. Available: http://pakiti.com/sdn-101-using-mininet-and-sdn-controllers/. [Accessed: 07-Aug-2021].

Shah, S. A., Faiz, J., Farooq, M., Shafi, A., & Mehdi, S. A. (2013). An architectural evaluation of SDN controllers. *IEEE International Conference on Communications*, 3504–3508. https://doi.org/10.1109/ICC.2013.6655093

Shalimov, A., Zuikov, D., Zimarina, D., Pashkov, V., & Smeliansky, R. (2013). Advanced study of SDN/OpenFlow controllers. *ACM International Conference Proceeding Series*, *January 2014*. https://doi.org/10.1145/2556610.2556621

Shamim, S., Shisir, S., Hasan, A., Hasan, M., & Hossain, A. (2018). *Performance Analysis of Different Openflow* (Vol. 18, Issue 1).

Sharma, S., Staessens, D., Colle, D., Pickavet, M., & Demeester, P. (2013a). Automatic bootstrapping of openflow networks. *IEEE Workshop on Local and Metropolitan Area Networks*. https://doi.org/10.1109/LANMAN.2013.6528283

Sharma, S., Staessens, D., Colle, D., Pickavet, M., & Demeester, P. (2013b). Fast failure recovery for in-band OpenFlow networks. *2013 9th International Conference on the Design of Reliable Communication Networks (DRCN)*, 52–59.

Sharma, S., Staessens, D., Colle, D., Pickavet, M., & Demeester, P. (2016). In-band control, queuing, and failure recovery functionalities for openflow. *IEEE Network*, *30*(1), 106–112. https://doi.org/10.1109/MNET.2016.7389839

Hussein Ali Al-Gubouri received his B.S and M.S degrees in Computer engineering from University of Mosul, Iraq (2010) and The University of Nineveh, Mosul, Iraq (2020) respectively, my master's thesis was software-defined networks for drone network management, and I have several published research papers in this field.