Performance Evaluation of SDN Controllers

Disha Dudhal
ddudhal@purdue.edu
Purdue University
Indiana, USA

Sandhya Arumugam Karunanithy
arumugam@purdue.edu
Purdue University
Indiana, USA

Sai Phani Sharath Chandra Danthalapelli
sdanthal@purdue.edu
Purdue University
Indiana, USA

Tanmaya Udupa
tudupa@purdue.edu
Purdue University
Indiana, USA

Mohamed Yilmaz Ibrahim
ibrahi35@purdue.edu
Purdue University
Indiana, USA

ABSTRACT

Traditional networks lack the ability to scale and suffer from several limitations not allowing them to adapt to current network requirements. Software Defined Networking (SDN), a new network architecture in which the control plane and data plane are split, has evolved as a result of technological advancements. A controller is recognized as the vital element in the control plane, which is responsible for performing all significant operations for managing the data plane. Hence, it is necessary to evaluate the performance of these controllers under different conditions and settings based on a few QoS measures as defined in the paper "SDN Controllers: Benchmarking Performance Evaluation". Although our project focuses on reproducibility, we also aim to evaluate the controllers' QoS under novel conditions if time permits. For the course of our project, we are considering internal controller NOX and external controllers POX, ONOS, OpenDayLight, and FloodLight (or Ryu). The settings to be considered for our experiments include but are not limited to different Host-Switch topologies, network load, no. of operations, etc. The parameters we would like to consider for evaluating our SDN controllers include throughput, latency, jitter, percentage packet drop rate etc.

KEYWORDS

Software Defined Networks, Control Plane, Data Plane, Host-Switch Topologies, Latency, Throughput

ACM Reference Format:
Disha Dudhal, Sandhya Arumugam Karunanithy, Sai Phani Sharath Chandra Danthalapelli, Tanmaya Udupa, and Mohamed Yilmaz Ibrahim. 2022. Performance Evaluation of SDN Controllers. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Traditional networks suffer from scalability issues. Internet Service Providers and network backbones are experiencing issues like traffic congestion, lengthy installation periods, and prolonged network convergence times as a result of the exponential growth in internet traffic. With these problems in mind, Software Defined Networks have recently gained ground and appeal. Software Defined Networks allow us to separate the control plane and the data plane. The control plane is then utilized as a centralized entity that controls the underlying network resources. With this approach, network administrators may simply change network resources and policies in response to shifting traffic demands and priorities and have greater control over network operations. After understanding the flexibility offered by SDNs in traffic management and flow management of networks, the networking industry and academia have started using SDNs for building large and complex networking systems.

An SDN controller is the fundamental and core element of SDN architecture. They are called the brains of the network since they have a complete view of the underlying network topology, network devices, and how the traffic flows in the network. It is the medium that connects the application to the underlying network. SDN controllers allow us to define policies that help in automated network management. Based on the policies defined by the network operators, the data plane can be programmed to perform a particular function through protocols like OpenFlow. As new requirements emerged for various applications, there was an increase in the development of controllers to meet these requirements. This made it necessary to compare these controllers on both a qualitative and quantitative level. Engineers and researchers would be able to choose controllers more effectively depending on the requirements at hand with the help of a clear comparison between the various controllers.

This study compares five distinct SDN controllers—RYU, POX, Opendaylight, Floodlight, and ONOS—in depth on both a qualitative and quantitative level. Using benchmarking tools like Cbench and Iperf, these controllers are compared in terms of metrics like controller latency and throughput, node-to-node latency, throughput, and round-trip timings. The qualitative comparisons were made based on the programming language used, the architecture of the controller, the ability to support multithreading, and the platforms that are supported by the controllers. There are other numerous

controllers present, but the scope of this paper will be limited to the above-mentioned controllers.

2 SDN CONTROLLERS

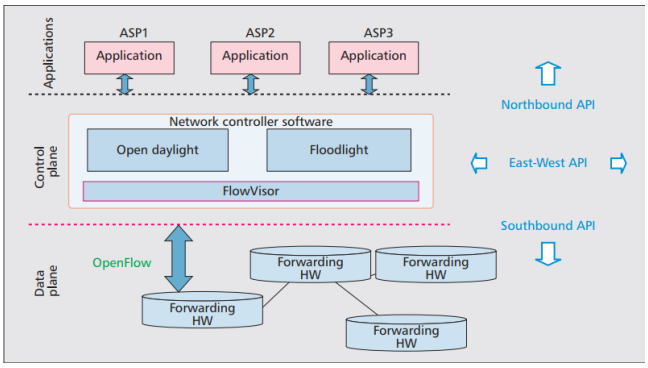

Figure 1: The Software defined architecture(https://tinyurl.com/2jajawd8).

Figure 1 shows a simple architecture of Software Defined Networks. The controller is like the medium between the user applications and the underlying network devices. The architecture shown here is general architecture. The architecture of each of the SDN controllers may differ from the others. SDN Controllers have 2 main components, the controller core, and the controller interfaces. The core contains different modules like topology manager, link discovery manager, flow manager, etc. These modules are used for managing traffic and maintaining the topology-related information of the underlying network. The topology manager has the topology map of the network and uses algorithms to find optimal paths between nodes and find routes for efficient traffic management. In some controllers, it also helps in visualizing the topology map. This helps network administrators to understand and get a view of how the network is built. The link discovery module helps in rebuilding the topology map when a network device is connected or disconnected from the network. The flow manager interacts with the network devices flows and the flow tables. The controller interfaces are used to communicate with the applications and the underlying network devices. The southbound interfaces of the controller are used as a communication channel to send packets and receive packets to and from the network devices. Openflow is the most commonly used Southbound interface (SBI). The East Bound interfaces and West Bound interfaces are used for inter controller communications and the Northbound Interfaces are used to communicate with the user applications. We considered the top 5 most popular controllers in the market for our project namely Ryu, Floodlight, ONOS, POX and OpenDayLight. The following sections give a brief overview of each of these controllers.

2.1 Ryu

Ryu (means flow in Japanese) is a comopnent based SDN controller written in Python and has a centralised architecture [1]. It is supported by Linux and MacOS and supports multithreading. It uses REST as its Northbound interface and Openflow as its South bound interface.

One of the key advantages of RYU is its simplicity and ease of use. Because it is written in Python, RYU is easy to learn and use, even for developers who are new to SDN and network programming. This makes it a good choice for organizations that want to quickly get up and running with SDN without a steep learning curve.

2.2 Opendaylight

The OpenDaylight controller[8] is JVM software that can be run from any OS and hardware as long as it supports Java. The core of the ODL platform is the Model-Driven Service Abstraction Layer (MD-SAL). In ODL, underlying network devices and network applications are all represented as objects, or models, whose interactions are processed within the SAL.ODL includes support for the broadest set of protocols in any SDN platform – OpenFlow, OVSDB, NETCONF, BGP and many more – that improve programmability of modern networks and solve a range of user needs. ODL gives flexibility of installing features as per the requirement of task at hand from it's given pool of features.

The ODL SDN Controller[12][13] Platform provides unified network intelligence, enterprise-class scalability and high availability, and a platform to deploy a wide range of network applications, including data center network virtualization. OSCP uses industry standard protocols, like OpenFlow, to create a common abstraction and universal data model for the underlying network data plane elements. When combined with open and published application programming interfaces (APIs), OSCP offer the most flexible platform to deploy universal, network-wide applications. The features include but not limited to : Topologies: Support for non-OF networks, VLAN-network tagging , High Availability (active/standby), OpenFlow Support, switch support for both hardware and virtual switches (OVS).

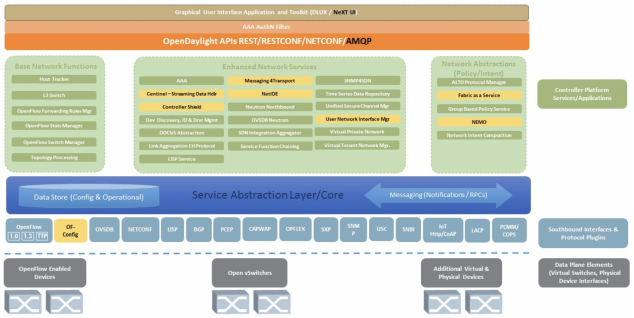

Figure 2: OpenDaylight Architecture(https://developer.cisco.com/site/opendaylight/discover/overview/)

2.3 Floodlight

Floodlight Controller is an SDN Controller [4, 10] developed by an open community of developers, many of which from Big Switch Networks. It uses the Open Flow protocol to organize traffic flows. Big Switch initially provided the Floodlight Controller as a component of the OpenDaylight Project, but Big Switch withdrew in June

2013. The OpenDaylight Project is not affiliated with the Floodlight Controller, despite the fact that it is still open source. Developers may benefit from the Floodlight Controller because it is designed in Java and enables them to quickly adjust software and construct apps. REST APIs are included, which make it simpler to programmatically interact with the product. Additionally, the Floodlight website provides coding samples to help developers create the product.

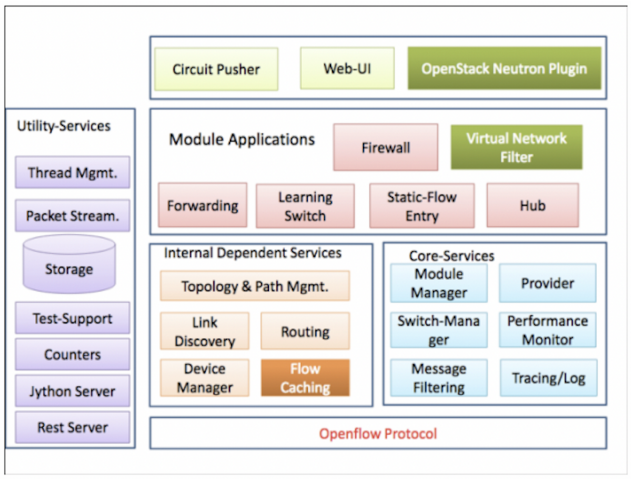

Figure 3: Floodlight architecture (https://tinyurl.com/u5csvbra)

Tested with both physical and virtual OpenFlow-compatible switches, the Floodlight Controller can work in a variety of environments and can coincide with what businesses already have at their disposal. It can also support networks where groups of OpenFlow-compatible switches are connected through traditional, non-OpenFlow switches. The Floodlight Controller is compatible with OpenStack [6], a set of software tools that help build and manage cloud computing platforms for both public and private clouds. Floodlight can be run as the network backend for OpenStack using a Neutron plugin that exposes a networking-as-a-service model with a REST API that Floodlight offers. Floodlight Controller architecture is shown in Figure 3.

2.4 Onos

ONOS[3][2] is an open-source software-defined networking (SDN) controller designed for managing large-scale, high-performance networks. It is written in Java and uses a modular architecture to support a variety of network protocols and applications. ONOS is highly scalable, with features such as active-active clustering and distributed data storage, and is built to be resilient and available. The ON.Lab, a non-profit organization dedicated to advancing networking technology, actively develops and maintains ONOS. Its microservices-based architecture allows for easy deployment and management of various network applications, and it supports a range of protocols and technologies, including Ethernet, MPLS, and OpenFlow.

2.5 Pox

[5] POX is a Python based open source OpenFlow/Software Defined Networking (SDN) Controller developed by James "Murphy" McCauley. POX is used for faster development and prototyping of new network applications. POX controller comes pre installed with the mininet virtual machine. The forwarding.l2 learning component makes OpenFlow switches act as a type of L2 learning switch. While this component learns L2 addresses, the flows it installs are exact-matches on as many fields as possible. [11] For example, different TCP connections will result in different flows being installed.

3 BENCHMARKING TOOLS

We have used Cbench, iperf and MTCbench for analysis. We measured Latency, throughput and other parameters using them.

Cbench[14] is a tool for measuring the performance of network controllers. It was developed by the Open Networking Foundation to help network administrators evaluate the capabilities of different OpenFlow switches and identify potential bottlenecks in their networks. Cbench can simulate various traffic patterns and network scenarios, allowing administrators to test the performance of a network controller under different conditions. This can help them optimize their network for specific applications and workloads.

Iperf is a network performance testing tool that is commonly used to measure the bandwidth and throughput of a network connection. It works by transmitting a specified amount of data over the network and measuring the time it takes to complete the transfer, allowing users to calculate the network's throughput. iperf can be run in either client or server mode, allowing users to test network performance between two points on a network. It is a useful tool for network administrators and engineers who need to evaluate the performance of a network and identify potential bottlenecks or other issues.

In addition to measuring bandwidth and throughput, iperf can also be used to test other aspects of network performance, such as jitter and packet loss. It supports a range of options and configuration settings, allowing users to customize their tests to simulate specific network conditions and scenarios. iperf is available for a wide range of operating systems, including Windows, Linux, and macOS, and it is free and open-source software. It is commonly used in conjunction with other network performance tools, such as ping and traceroute, to diagnose and troubleshoot network issues.

4 EXPERIMENTAL SETUP

In our study, we have used mininet [7] for building network topologies. Mininet is basically a network emulator which emulates network devices such as switches, hosts and the links between them. Mininet allows us to create different network topologies such as linear, single, reversed, tree, torus and also custom topologies. The hosts mininet creates are standard Linux based hosts and the switches have support for OpenFlow allowing controllers to program these switches. The controllers were run on Amazon EC2 t2.large instance with intel Xeon processor and had 2 CPU cores. We used l2 learning switches for our experiments. Benchmarking tools like Cbench and MtCbench[9] (for multithreading experiments) were installed on the EC2 instance and interacted with the controller for determining the performance metrics.

Table 1: Qualitative comparison of the controllers

Controller	Programming Language	Architecture	Multithreading	Platforms
Ryu	Python	Centralised	Yes	Linux, MacOS
Opendaylight	Java	Distributed Flat	Yes	Linux, MacOS, Windows
ONOS	Java	Distributed Flat	Yes	Linux, MacOS, Windows
Floodlight	Java	Centralised	Yes	Linux, MacOS, Windows
POX	Python	Centralised	No	Linux, MacOS, Windows

5 RESULTS AND DISCUSSION

We present the observations using graphical representation of our results and further it with some inferences we obtained. Table 1 shows the qualitative comparisons between the controllers.

5.1 Cbench Latency Mode

Latency of the controller denotes the time it takes to process a single packet. We tested the controller for latency performance. For this metric, we ran Cbench in latency mode which sends a packets to the controller and waits for response before sending another packet. We evaluated the controller swith different number of virtual OpenFlow switches emulated by Cbench. Each latency test was run for 10 iterations.

Fig. 4 shows the results obtained using Cbench tool's latency mode. Cbench latency identifies the communication delay between the controller and the switch. Here every switch is directly connected to the controller. In Cbench latency mode all the switches send packets to the controller in a serial manner and wait for its reply before sending the next packet. The reciprocal of the average number of flows processed by the controller gives the Cbench latency metric.

For this particular experiment, we kept the number of test runs within the Cbench latency mode constant at 10 across all the controllers.

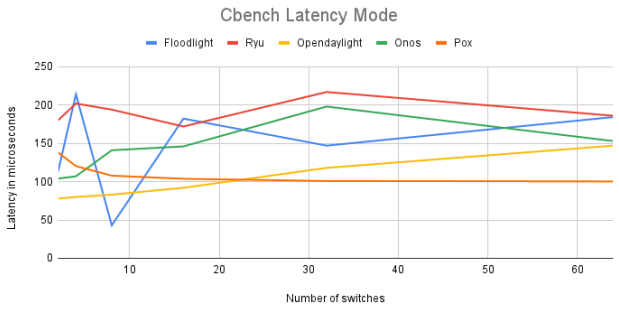

Figure 4: Cbench - Latency

Following were our observations regarding different controllers based on Cbench latency mode results:

(1) We observed that the POX controller shows a minimal change in latency as the number of switches increases. However, less latency does not translate to an outright winner as the capabilities of the controller itself must be considered.
(2) For OpenDayLight, latency increases fairly and linearly with the number of switches.
(3) Latency was barely affected for the Ryu controller. It remained constant throughout the experiment.
(4) Floodlight and ONOS controllers showed fluctuating performance where after 45 switches, Floodlight showed a linear increase in latency whereas ONOS showed a linear decrease in latency.

5.2 Cbench Throughput Mode

In throughput mode, each switch sends as many packets as the switches' buffer allows to the controller without waiting for a reply. This is done because we essentially want to measure how many packets can be handled/processed by the controller at once per switch. Thus, the throughput mode measures the maximum flow rate that a controller can handle. In Cbench Throughput mode there are 10 thousand hosts connected to each switch.

We present our results from the Cbench throughput mode in Fig 5 and list the observations.

(1) Ryu and OpenDayLight were consistently the lowest performers. This was contrary to the results we expected, both Ryu and OpenDaylight because both of these controllers support multithreading.
(2) Floodlight showed increased throughput with an increased number of switches indicating that it might be suitable for tasks that would want to maintain their throughput rates for large-scale networks.
(3) Pox surprisingly showed fairly constant throughput rates. This might be true for small networks however we might need additional experiments to verify POX's performance in huge networks.

5.3 Node to Node Latency Mode

Node-to-node latency measures the amount of time required to send a packet and receive a reply between two hosts that are connected to L2-learning switches which are in turn connected to the controller. In our graphs, we present statistics gathered using the **Iperf** benchmarking tool as well as Ping statistics.
In this experiment we tried to ping the farthest hosts within a network to get the worst possible ping time in that network, and tested this setting on different network topologies. We used TCP packets to ping the hosts.

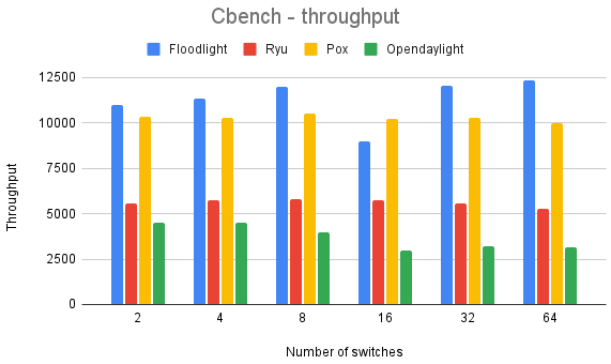

Figure 5: Cbench - throughput

We conducted 15 test rounds of pings between hosts, where the packet size was fixed to the default ping packet size. The results presented here include the ping time averaged over these 15 tests. Please note that the first ping between any two hosts for any of the controllers showed the highest ping time.

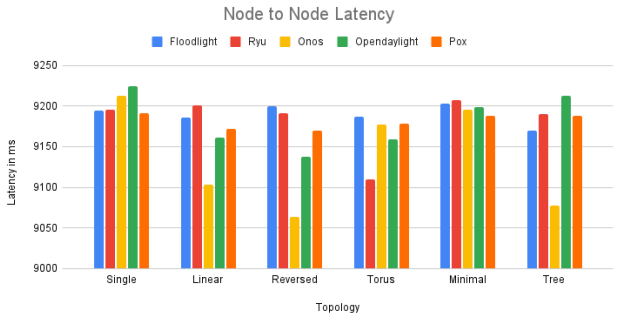

Figure 6: Node to Node Latency

Fig 6 summarizes the average node-to-node latency time for each controller for every topology.

(1) Lower latency means better performance.
(2) ONOS gives the least latency, however, its performance oscillates between different topologies.
(3) Floodlight and POX have stable latency times across all the topologies for a fixed number of hosts. This can be attributed to the simpler designs of these controllers however a simpler design does not necessarily translate to better controller capabilities.
(4) We believe commenting on OpenDayLight and Ryu performances for these topologies wouldn't be apt at this time since we might need additional experimental analysis with huge networks to actually evaluate their performance. Both of these controllers support multi-threading and are hence better suited in distributed networks.
(5) Single and Minimal topologies show similar performance trends.

5.4 Node to Node Throughput Mode

In Node to Node throughput mode, we measure the maximum bandwidth utilized between 2 hosts using the Iperf tool. Iperf provides results in the form of "Interval", "Transfer Size" and "Bandwidth". *Interval* represents the time duration for which data was transferred between two hosts. *Transfer Size* specifies the amount of data to be transfered in Mega bytes and the Bandwidth specifies the number of Mega bits transfered per second.
For the node-to-node throughput, we did our Iperf tests on TCP flows. We capture our results and visualize them in Fig 7 Our observations are as follows:

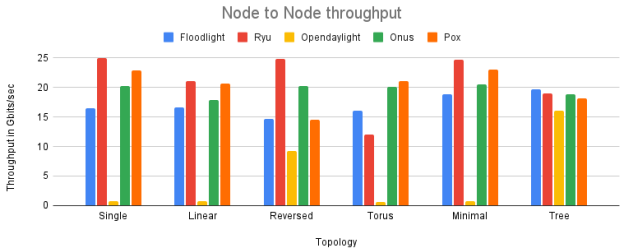

Figure 7: Node to Node throughput

(1) Ryu was consistently the best performer out of all the controllers across all topologies.
(2) Following Ryu, ONOS, POX and Floodlight gave a stable throughput and did not show high variation.
(3) Opendaylight had the worst throughput out of all the controllers.
(4) For tree topology all the controllers performed fairly the same.
(5) Single and Linear topologies displayed similar performance trends for all the controllers.
(6) OpenDayLight had the worst performance in Torus topology.

5.5 Jitter

For understanding what jitter is we need to understand the formal definition of Delay. *Delay* is the difference between the transmission time from the sender and the reception time on the receiver. *Jitter* on the other hand expresses the variation in Delay on packet flow between two To evaluate jitter, we used Iperf tests across topologies in the UDP mode.
Fig 8 summarizes our results for Jitter analysis

(1) Ryu has the least jitter out of all the controllers under consideration, aka it has a fairly consistent delay.
(2) ODL has fluctuating jitter while ONOS and POX have minimal jitter variation across topologies.

5.6 Throughput for Multithreaded Controllers

We also extended our research by testing the effects of multithreading on controller throughput. We observed that Ryu and Opendaylight have better throughput with increasing number of threads. However, Floodlight's performance drastically decreases with increasing thread. This result is contrary to the general behavior of

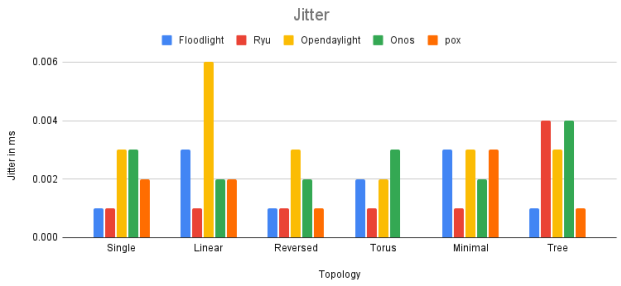

Figure 8: Jitter

this controller hence there is a need for more experimental analysis since this observation deviates from our original hypothesis.

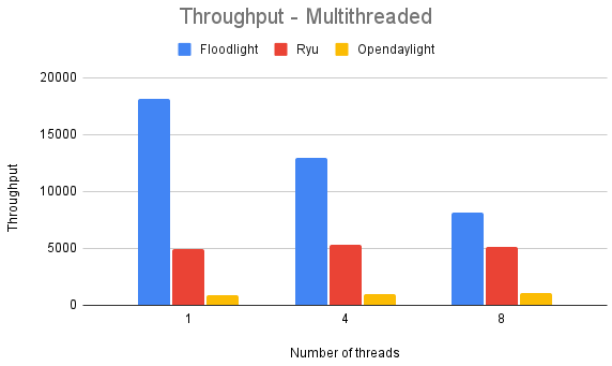

Figure 9: Throughput - Multithreaded

6 INFERENCES

- Multi-threaded controllers perform significantly better than single-threaded controllers like (POX and RYU).
- Topology has an impact on the performance metrics and, efficient handling of switches is characteristic of every controller.
- Placement of the controller in physical topologies greatly affects the evaluation of performance parameters. We plan to include our studies on different data center topologies like Fat and Short trees with many devices.
- Limitations of evaluation tools directly affect benchmarking results. Changes in underlying physical resources or compilers may have a noticeable impact on the results collected.
- Isolating the performance of a controller from its results is not possible since the features of the tools available greatly impact the performance of the controller in a given topology.
- Simple controllers are better suited for lightweight tasks. Feature-based controllers are better suited for heavy performance tasks.
- Comparing controllers is use-case and application-dependent, simply comparing them on performance metrics would not be enough.
- Extending topologies with many hosts is necessary.
- All controllers should be installed directly onto the machine, using docker containers or emulated controllers can directly affect the results that are collected.

7 FUTURE WORK

We wish to conduct a more comprehensive evaluation of the five most common SDN controllers. This could involve testing the controllers across a wider range of test scenarios, including different network topologies and a wider range of network loads. This would help to provide a more comprehensive understanding of the controllers' performance and identify any differences in their behavior under different conditions. Additionally, extending the work to include other SDN controllers like Trema and TinySDN would provide a more comprehensive view of the different controller options available. This could help organizations to make more informed decisions about which controller to use in their SDN deployments. Using other performance benchmarking tools like PktBlaster and Browbeat could provide more detailed and accurate metrics on the controllers' performance. This could help to identify any strengths and weaknesses of the different controllers and provide more specific insights into their behavior. Finally, extending the work to specific use cases like datacenters, WANs, and IOT could help to tailor the results to the specific needs of these applications. We also as an extension we wish to perform load balancing on the 5 controllers to analyse the impact on the performance. This could provide valuable insights into the best controller options for different use cases and help organizations to make more informed decisions about their SDN deployments.

8 CONCLUSIONS

From our project we have concluded that overall, all the controllers performed differently for different topologies. We have inferred that Multi-threaded controllers perform significantly better than single-threaded controllers like (POX and RYU). Different topologies also has an impact on the performance metrics and, efficient handling of switches is characteristic to every controller. And our inference is that different controllers can be used in different use cases depending on the task at hand such as simple controllers are better suited for lightweight tasks and Feature based controllers are better suited for heavy performance tasks.

9 ACKNOWLEDGMENTS

The project has been supported by the Purdue Department of Computer Science, CS 536: Data Communication And Computer Networks and also Professor Muhammad Shahbaz.

10 AUTHORS AND CONTRIBUTIONS

10.1 Disha Dudhal

- Analysis of Pox Controller
- Experimental setup for Pox controller with mininet in AWS platform
- Performance evaluation and tests for Pox controller with benchmarking tools Iperf and Cbench
- Latency analysis and comparison for all controllers

10.2 Sandhya Arumugam Karunanithy

- Analysis of Opendaylight Controller
- Feature installations for Open daylight controller and setup with mininet in AWS platform
- Performance evaluation and metrics tests for ODL controller with benchmarking tools Iperf and Cbench
- Node to Node Throughput analysis and comparison for all controllers

10.3 Sai Phani Sharath Chandra Danthalapelli

- Analysis of Onos Controller
- Experimental setup for Onos controller with mininet in AWS platform
- Performance evaluation and tests for Onos controller with benchmarking tools Iperf and Cbench
- Jitter analysis and comparison for all controllers

10.4 Tanmaya Udupa

- Analysis of Ryu Controller
- Experimental setup for Ryu controller with mininet in AWS platform
- Performance evaluation and tests for Ryu controller with benchmarking tools Iperf and Cbench
- Latency, Throughput analysis and comparison for all controllers with Cbench

10.5 Mohamed Yilmaz Ibrahim

- Analysis of Floodlight Controller
- Experimental setup for Flood light controller with mininet in AWS platform
- Performance evaluation and tests for Floodlight controller with benchmarking tools Iperf and Cbench
- Throughput single and multi threaded analysis and comparison for all controllers with cbench and mtcbench.

REFERENCES

[1] 2014. Ryu SDN Framework. Retrieved December 1, 2022 from https://ryu-sdn.org/

[2] 2020. Open Network Operating System - Wiki. Retrieved December 1, 2022 from https://wiki.onosproject.org/display/ONOS/ONOS

[3] 2022. Open Network Operating System (ONOS) SDN Controller for SDN/NFV Solutions. Retrieved December 1, 2022 from https://opennetworking.org/onos/

[4] Saleh Asadollahi and Bhargavi Goswami. 2017. Experimenting with scalability of floodlight controller in software defined networks. In *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*. 288–292. https://doi.org/10.1109/ICEECCOT.2017.8284684

[5] "POX Documentation". "". "POX Controller Documentation". ("" ""). "https://noxrepo.github.io/pox-doc/html/"

[6] Floodlight. 2016. Floodlight Framework. https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343557/User+Documentation

[7] Karamjeet Kaur, Japinder Singh, and Navtej Singh Ghumman. 2014. Mininet as software defined networking testing platform. In *International conference on communication, computing & systems (ICCCS)*. 139–42.

[8] Zuhran Khan Khattak, Muhammad Awais, and Adnan Iqbal. 2014. Performance evaluation of OpenDaylight SDN controller. In *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*. 671–676. https://doi.org/10.1109/PADSW.2014.7097868

[9] Nikos Anastopoulos Konstantinos Papadopoulos. 2017. intracom-telecom-sdn/mtcbench: Multithreaded Cbench. Retrieved December 1, 2022 from https://github.com/intracom-telecom-sdn/mtcbench

[10] Laura Victoria Morales, Andres Felipe Murillo, and Sandra Julieta Rueda. 2015. Extending the Floodlight Controller. In *2015 IEEE 14th International Symposium on Network Computing and Applications*. 126–133. https://doi.org/10.1109/NCA.2015.11

[11] Adarsh V Srinivasan et.al " "N. Saritakumar. "2019". "Performance Evaluation of Pox Controller for Software Defined Networks". ("July" "2019"). "https://www.ijitee.org/wpcontent/uploads/papers/v8i9S2/I11390789S219.pdf"

[12] ODL. 2018. ODL SDN Framework. https://www.opendaylight.org

[13] ODLframework. 2016. ODL Framework. https://wiki.opendaylight.org/pages/viewpage.action?pageId=336424

[14] Liehuang Zhu, Md Monjurul Karim, Kashif Sharif, Fan Li, Xiaojiang Du, and Mohsen Guizani. 2019. SDN controllers: Benchmarking & performance evaluation. *arXiv preprint arXiv:1902.04491* (2019).