Turnitin Originality Report

System Utilization Analysis of SDN/OpenFlow Controllers: Ryu vs Pox by Gautam Jee

From thesis version 1 (sdn)

Processed on 08-Jun-2020 17:20 IST
ID: 1340047030
Word Count: 8341

Similarity Index	Similarity by Source	
7%	Internet Sources:	5%
	Publications:	2%
	Student Papers:	6%

sources:

1 2% match (student papers from 01-Jun-2019)
Class: SDN
Assignment: Strict Thesis
Paper ID: 1135733698

2 < 1% match (student papers from 05-May-2016)
Submitted to National Institute of Technology, Silchar on 2016-05-05

3 < 1% match (student papers from 02-Jun-2020)
Submitted to National Institute of Technology, Silchar on 2020-06-02

4 < 1% match (publications)
Jehad Ali, Seungwoon Lee, Byeong-hee Roh. "Performance Analysis of POX and Ryu with Different SDN Topologies", Proceedings of the 2018 International Conference on Information Science and System - ICISS '18, 2018

5 < 1% match ()
http://ethesis.nitrkl.ac.in/4626/

6 < 1% match (Internet from 26-Sep-2014)
http://www.whileifblog.com/2012/04/15/linux-find-out-cpu-architecture-information/

7 < 1% match (student papers from 08-Dec-2010)
Submitted to British Institute of Technology and E-commerce on 2010-12-08

8 < 1% match (Internet from 08-Oct-2008)
http://wing.comp.nus.edu.sg/publications/theses/eugeneEzekielThesis.pdf

9 < 1% match (Internet from 05-May-2020)
https://www.scribd.com/document/236696219/2

< 1% match (Internet from 15-Apr-2020)

10 https://mro.massey.ac.nz/handle/10179/12919?show=full

11 < 1% match (Internet from 09-Feb-2020)
https://kolesnikov.pw/kak-proverit-informaciju-o-processore-v-linux

12 < 1% match (Internet from 26-Mar-2020)
https://www.ijert.org/water-tank-monitoring-system

13 < 1% match (student papers from 16-Apr-2018)
Submitted to Western Governors University on 2018-04-16

14 < 1% match (student papers from 08-Nov-2017)
Submitted to Wright State University on 2017-11-08

15 < 1% match (student papers from 06-Aug-2018)
Submitted to Texas A&M University, San Antonio on 2018-08-06

16 < 1% match (Internet from 05-Aug-2018)
https://community.centminmod.com/threads/intel-processor-flaw-kernel-memory-leaking-spectre-meltdown.13632/

17 < 1% match (Internet from 29-Feb-2020)
http://www.faqs.org/rfcs/rfc8456.html

18 < 1% match (publications)
Hanjiang Luo, Kaishun Wu, Rukhsana Ruby, Yongquan Liang, Zhongwen Guo, Lionel M. Ni. "Software-Defined Architectures and Technologies for Underwater Wireless Sensor Networks: A Survey", IEEE Communications Surveys & Tutorials, 2018

19 < 1% match (publications)
Ola Salman, Imad H. Elhajj, Ayman Kayssi, Ali Chehab. "SDN controllers: A comparative study", 2016 18th Mediterranean Electrotechnical Conference (MELECON), 2016

20 < 1% match (student papers from 22-May-2015)
Submitted to IIT Delhi on 2015-05-22

21 < 1% match (student papers from 08-Aug-2016)
Submitted to University of Portsmouth on 2016-08-08

22 < 1% match (publications)
Intidhar Bedhief, Meriem Kassar, Taoufik Aguili. "From Evaluating to Enabling SDN for the Internet of Things", 2018 IEEE/ACS 15th International Conference on Computer Systems and

Applications (AICCSA), 2018

23 < 1% match (student papers from 04-Oct-2015)
Submitted to University of Duhok on 2015-10-04

24 < 1% match (Internet from 14-Nov-2019)
https://tel.archives-ouvertes.fr/tel-02081080/file/These-2018-MATHSTIC-Informatique-ZERKANE_Salaheddine.pdf

25 < 1% match (student papers from 27-Apr-2020)
Submitted to Glasgow Caledonian University on 2020-04-27

paper text:

System Utilization

15**Analysis of SDN/OpenFlow Controllers:** Ryu Versus **Pox**

9**A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Technology**

Gautam Jee (18-2-5-107)

2**Department of Computer Science & Engineering NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR May,**

2020 System Utilization

15**Analysis of SDN/OpenFlow Controllers:** Ryu Versus **Pox**

10**A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Technology**

Gautam Jee (18-2-5-107) under the supervision of Umakanta Majhi (Assistant

5**Professor) Department of Computer Science & Engineering National Institute of Technology** Silchar **May,** 2020 **Department of Computer Science**&

Engineering **NATIONAL INSTITUTE OF TECHNOLOGY**

SILCHAR Declaration Thesis Title: System Utilization Analysis of SDN/OpenFlow Controllers: Ryu Versus Pox

1**Degree for which the Thesis is submitted: Master of Technology I declare that the presented thesis represents largely my own ideas and work in my own words. Where others ideas or words have been included, I have adequately cited and listed in the reference materials. The thesis has been prepared without** resort- ing **to plagiarism. I have adhered to all principles of academic honesty and integrity. No falsified or fabricated data have been presented in the thesis. I understand that any violation of the above will cause for disciplinary action by the Institute, including revoking the conferred degree, if conferred, and can also evoke penal action from the sources which have not been properly cited or from whom proper permission has not been taken. Date**

: Gautam Jee Reg. No : 18-2-5-107 i Department of

1**Computer Science & Engineering NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR Certificate It is certified that the work contained** within **this thesis entitled**

“System Utilization Analysis of SDN/OpenFlow Controllers: Ryu Versus Pox” submitted by GAUTAM JEE Scholar no: 18-2-5-107

3**for the award of** M **.Tech is absolutely based on his own work carried out under my supervision and this** thesis **has not been submitted elsewhere for any degree. Date** : Place : Umakanta Majhi **Assistant Professor Department of** Computer Science &**Engineering**

NIT Silchar, India ii Abstract Throughout the years, various SDN or Software Defined Network working frame- works is created in various dialects models. The level of significance in a SDN engi- neering is its control plane which is administered by the substance called ”Controller ”. This require a need to comprehend the benefits and bad marks of the controllers with the goal that one may pick a proper usage for a system situation. Here, an investigation of two SDN controllers is done with the expect to comprehend the framework usage of SDN Controller under overwhelming traffic and gives a superior thought regarding its running nature, framework, bottlenecks, and so forth which could additionally tell the engineers the improvement territories to work in its up and coming renditions. iii Acknowledgement Achievements in life are always obtained through some key ingredients added into the recipe which is your work. A spoon of guidance, a touch of inspiration, a squeeze of determination and heaps of blessings are the most crucial among them. This work would not have been possible without the constant support and motivation given to me by my teachers and family. Hence,

1**I take this opportunity to express gratitude to**

my guide and mentor, Umakanta Majhi,

1**Assistant Professor, Department of Computer Science and** Engi- neering, **National Institute of Technology, Silchar for** his expert **guidance, innovative suggestions, constant encouragement and**

1**patience with which** he **handled** me and **the project.**

His calm minded approach to the project instilled the confidence within me and taught me to embrace the challenges that were ahead. I am forever grateful to Dr. Samir Kumar Borgohain, Head of the Department, Department of

2**Computer Science and Engineering for his** encouragement **and** consid- erations **throughout my research work. His** constant **support** has **been invaluable to me in the** pursuit **of my research work.** Finally, **I**

express my gratitude to my beloved parents, family and friends who directly or indirectly extended help and prayers to make this thesis successful. Gautam Jee Reg. No- 18-2-5-107 iv Contents Declaration Certificate

8**Abstract Acknowledgement List of Figures List of Tables 1 Introduction 1.1** Traditional Networks . . . . **1.2** Software Defined Networks . . . . **1.3** Motivation & Problem Statement **2**

Literature Survey 3 Overview 4 Experimental Setup 4.1 Hardware Configuration . . . . 4.2 Software & Tools . . . . 5 Results & Discussions 5.1 CPU Utilization . . . . 5.2 Instructions Per Second . . . . 5.3 CPU Clock Speed 5

Chapter 1 Introduction The term network system passes on, an association of various PCs. For the most part, these systems have two kinds of availability, which are wired and remote. As on account of any innovation, it is development came out as a need. For any coun- try, its most extreme need is to keep up military certainty. The first network system, was the beginning of the era and was essential set up in a military resistance venture called ARPANET, representing the Advanced Research Projects Agency Network. PC arranges throughout the years have developed a wide margin regarding highlights, in- tricacy, and power. Nowadays, even a layman is familiar to the terms such as switches and routers. 1.1 Traditional Networks Even something as new and intense as network systems have demonstrated age. The maturing of innovation is by all accounts a lot snappier than people. Inside two decades, it built up an area of systems known as traditional networks. The spearhead- ing plan of networks is the traditional network. A structure was solid for the time and had no issues for quite a while that was given by Adam Smith of computer network- ing. Indeed, even now, the traditional network system engineering cannot be blamed as is yet a subject of innovative work. Nevertheless, concerning each innovation that exists, there will be advantages and disadvantages. The computer network has been created for the sole purpose of communication. This implies that the advanced data was moved starting with one PC then onto the next remote unit(s), through wire or air. It is proportionate to sending a post starting with one individual then onto the next. 1 Figure 1.1: Traditional Networks [1] In the case of computer networks, this function of submitting a post is the same and is referred to as routing. No human intervention exists in computer networks; instead, there are independent computers responsible for transmitting this data. Tasks like packet forwarding is assigned to devices that make up a large share in today's global network, called routers. Routers play a vital role in a network, i.e., forwarding packets through specific routes by identifying the current scenario of the neighboring devices and keeping track of the connected devices. Similar to how someone seems to have a PO box connected to their location, every device in a local area network is listed under a router. Such routers are dispersed across the network and are necessarily the foundation of the network. The same explanation is that this method has been tested for years in such a way that it has strong error detection and error correction. Such a network with millions of routing protocol devices ensures redundant information, so there is no single risk. This routing technique has been used for a long period of time. This will tend to be done for the lifespan of data networks. Across time, routing methods have im- proved significantly, and numerous techniques and network topologies [2] have been researched and developed, rendering computer networks even more efficient in mod- ern times. However, simply because something works well does not mean that it can not be improved. Although the debate is not regarding developing routing methods, 2 in particular, it is stressed that the understanding of computer networks is so bland and outdated. The idea of transmitting a data packet with a header added to it to the enormous complexity of the global network appears to bind a message to a car- rier pigeon and allow it to ride. Yeah, the pigeon knows how to reach the target by a step-by-step method, but neither the transmitter nor the receiver nor the network administrator knows where the packet is before it enters the informal network where either the recipient or the network administrator is constantly watching and hoping for the packet to arrive. A simple layout or architecture of the traditional network can be viewed from the Fig. 1.1. There are several small networks connected to a common access point. These access-points are either switches, hubs, bridges or routers. These smaller networks are further connected to a common ISP, which gives these smaller networks access to the Internet. Packet forwarding is one aspect of the role of the network. Different highlights incorporate observing the system and checking which courses are accessible and which courses are down or under substantial loads with the goal that system traffic can be diverted to another course. Customary systems have some notable answers for such issues; huge numbers of them are ungainly and exploit the assumption that systems administration gadgets are because of quick preparing. They can manage numerous shortcomings simultaneously absent a lot of exertion. Although that might be real, it opens the path to change. Starting at now, there is no single controller to assess the exhibition of the huge system. The switches and routers deal with their immediate neighbours and assume that their neighbours can do likewise. At that point, it was a mind-blowing thought, yet it appears to be excessively simple. Another model of conventional systems is that every protocol, organize topology or design, expect that the whole network would run a solitary protocol for a lifetime. Then again, as it were, no protocol considered the requirement for versatility or sim- ilarity for change contingent upon the situation. A few conventions would adjust to deal with arrange inconsistencies and keep up reliable and unsurprising execution. In any case, the idea of making a network that could change its equipment and program- ming at some random time was unrealistic. On account of network issues, there are numerous measures sent to back up and ensure the system can be brought back ready for action. Be that as it may, the acknowl- 3 edgement of network flaw over the whole network is excessively delayed as far as fig- uring speeds. In the event that one switch or a server fail someplace in this immense network, the mistake message from its neighbours needs some an ideal opportunity for it to get spread out and diffused to each other gadget in the system. This deferred reaction on account of system deficiencies is something that can be improved. 1.2 Software Defined Networks Traditional networks follow equipment hardware programming engineering. At the end of the day, the systems administration gadgets utilized, for instance, a switch, originated from a particular seller, and the merchant utilizes either their own exclusive programming on it or an outsider programming specially designed for their equip- ment. Rather, take the instance of a PC. It has equipment produced using a merchant, yet the equipment configuration permits us to introduce any working framework on it and utilize the framework in any methods. Some working frameworks offer more or preferable highlights over some others. The whole network has an equipment stage which is reasonable for any or differ- ent working frameworks one after another. The whole network isn't characterized by the equipment, rather than the working framework that sudden spikes in demand for it and deals with the system. Such a design as appeared in Fig. 1.2 is executed by adopting a concentrated net- work strategy on decentralized network engineering. The sentence seems like a con- fusing expression, however, its substance is to broaden the decentralized network en- gineering by keeping up the data forwarding techniques in its place and on the whole bringing together the system dynamic procedures into a legitimately single element. At the end of the day, the directing usefulness from the switches are evacuated, and they act as switches while making a central element to play out the routing choices and also screen the whole network. This element is named as "Controller ". This postula- tion predominantly centres around controllers and the server where it is introduced. The Fig. 1.2 describes the layered architecture of SDN which has basically three distinct abstracted layers namely

14**Application Plane (Application Layer), Control Plane (Control Layer) and Data Plane** (Infrastructural **Layer).**

To communicate between these layers, there is use of few APIs like North-Bound API which acts as an interface be- 4 tween Application Plane and Control Plane; South-Bound API acts as an

24**interface be- tween Control Plane and Data Plane.**

There is use of East-West Bound APIs which acts an interface between its and other Controller 's Control Plane. Figure 1.2: SDN Architecture [1] Similar to Layer 1 or Physical Layer of OSI Model in Networking, the Infrastruc- tural Layer or Data Plane of SDN Architecture has the main task is to transmit raw bits over a physical data link connected to various network nodes which can be implement either by using a physical or a virtual device. Addition to this, it also has responsibility of

13**Layer 2 or Data-Link Layer** i.e. managing **the**

13**communications links and handling frame traffic**

and govern a protocol to access the physical network medium. All these tasks are implemented by use of switches like OpenVSwitch. The Control Plane of SDN Architecture can be seen as the Layer 3 or Network Layer of OSI, which is responsible for packet forwarding including routing through intermediate routers or other nodes. Unlike normal routers, the Control Plane of SDN is implemented at a server rather than on a router. The Control Plane features are established by use of a SDN Controller, an application. Some popular SDN Controllers are Pox, Ryu, Beacon, Nox, Floodlight, ONOS, OpenDayLight, Faucet, RunOS, etc. The

Control Plane is also often termed as Network Operating System. The Application Layer is the top most layer in SDN Architecture where several applications are installed on the server itself and run on top of Controllers via use of 5 various North-Bound APIs build using REST, JSON, or similar languages. Tools like Load Balancers, Network Monitoring, Traffic Controllers, etc are some popular ones in the Application Layer. OpenFlow is an open and standardized protocol and a flow-based switch speci- fication, which acts as a South-Bound Interface in SDN. It is utilised for collaborating with the sending practices of switches from different merchants. It also control the behavior of switches throughput in a dynamic network and is also programmable. It helps to controller to communicate with switch over a secured channel. It also defines the format of packets. The corresponding switches are known as OpenFlow Switches. These switches has 3 parts namely an OpenFlow protocol,

23**a flow table, and a secure channel.** These switches has **the** responsibility to forward **the**

flow's packets to given ports or,

18**drop the flow packets** or, **forward the flow's packets** to **the**

controller by encapsulating it. As mentioned above, a divide and conquer approach is been followed in the SDN for networking tasks. [3]. In place where every router in the network is performing both the tasks of routing and also of storing these routes to a routing table so to use these routes in future and also to notify the neighboring devices whenever there is a change in network which allows routers to focus on data-forwarding and this also reduces the repetitive calculations of find an optimal route. Whenever there is such new routes, the routers just need to ping the neighbours. All the functionality like calculation of optimal routes, screening of the topology, or health of the network, are performed by the controllers. This helps to manage the entire network like an interface. This further brings in various benefits to the network like the communication to other middleman devices is no longer needed to know the status of the network. Instead, now there is a central device i.e., Controller(s), is/are simply dumping all information within itself and thus is making sure whether the network run smoothly or not. Another preferred position of utilizing a different substance for controlling the system is that it permits the system to be convention autonomous. As it were, the working framework or the software that is running on the controller isn't fixed. The SDN engineering permits to change the working framework at whatever point re- quired. [4] In all actuality, it will be a tedious assignment before the system is done and running once more, however it surely is a snappier and less expensive option in 6 contrast to changing the product on all the gadgets in the network. In any innovation, the prizes will consistently come at either an exchange of or in danger. From moderate yet advantageous transportation of the bike to the marvellous however hazardous aeronautics innovations, improvement consistently includes some significant downfalls. So also, in SDN too, there are a few bills to be paid. Right off the bat, the execution of an aggregate substance called the controller is impeding placing all investments tied up on one place. This technique, anyway productive, despite everything, represents a hazard. The general guideline in any ad- ministration innovation is to build excess. In the event that, for sizeable SDN, there is a solitary controller, if there should arise an occurrence of any deficiencies running from power blackouts to disastrous events can make the whole system come up short. Another drawback of SDN is that the focal controller doesn't really screen the genuine traffic itself rather just the network's well-being and various routes. It doesn't really screen the traffic is on the grounds that controllers are equipped for checking network traffic-utilizing outsider applications. It is truly feasible for the controller to constantly ping a switch so as to get the parcels it handles and read its substance. Be that as it may, it is unbelievably illogical, and along these lines seeking after such an element would yield fewer returns than the exertion put in. So normally, designers don't concentrate on traffic checking as much when contrasted with organizing ob- serving and steering. This fundamentally diminishes the extent of SDN as systems in which traffic can't be observed effectively would represent a hazard as far as genuine- ness and security. Except if the whole network is trusted by the system manager and the network members, such a technique would appear to be less liked. To wrap things up, as an expansion to the focuses preceding the past one, since a solitary element controls the whole system's directing and screens the system well- being and topology, this will clearly bring about an extreme overhead on the controller. It was referenced that SDN networks are working framework autonomous, however they are, as it were, reliant on the equipment confinements. For example, process- ing power, bandwidth , and will keep the network from developing past an outright scale. As the network scales up, so does the complexity in taking care of such a broad network. Routing, topology discovery, etc., becomes awkward except if the controller equipment is updated.[5] Additionally, this makes the network versatile, which means it will in general 7 need to come back to its unique measurements, due to overemphasizing on the con- troller. Except if the controller is changed, which, despite the fact that not so costly or tedious, creates a level of lack of quality in the system, in traditional networks, load adjusting on routes would be performed by neighbouring routers, and the undertak- ing of network checking was assigned, this implied the network overhead could never go past a cutoff. In the event that all the switches in the network arrived at its cutoff, only including more switches and separating huge networks to littler subnets would be sufficient. Nonetheless, that is unsure on account of programming characterized systems. Circulated controllers take care of this issue to a degree, yet at the same time need more opportunity to form into an increasingly hearty methodology. 1.3 Motivation & Problem Statement SDN or Software Defined Networking brings an another point of view to orga- nize networks. The idea of having a concentrated way to deal with decentralized net- work design brings the benefits of the two models. SDN engineering gives the infras- tructural adaptability and vigor of the traditional systems with the additional element of having a central interface to screen and deal with the whole network. There are many SDN implementations, each with a different controller operating system. However, there comes a question

25**to choose the suitable controller for a** net- work **scenario**

where there are limited resources. Here, controllers are analyzed and tested for system-level performance metrics under a massive network traffic that ap- pears similar to real-world networks. It helps to identify system-level bottlenecks of the controllers and makes it possible to classify controllers based on their system per- formances. It also generates the idea of whether the provided system is suited to a given controller and vice-versa. 8 Chapter 2 Literature Survey A starter handle of SDN was required to get hypotheses and details that assume significant jobs in the system. The paper underneath was critical for looking for data on the changed networking frameworks that were being used and how a product char- acterised arrange was distinctive in contrast with the traditional network design. Distributed in 2014, the paper "A survey and a layered taxonomy of software- defined networking" [3] advances a framework that groups distributed research works and exposes the best possible way to seek after research. The paper depicted that there is a need to arrange and look at SDN Controllers yet from an alternate point of view. As opposed to simply estimating crude execution, the situation must be considered in which the working framework, runs just as what the system's goal is. Also, the paper titled " Software-defined Networking (SDN): a survey "[6] and published in 2016, apart from the description about SDN, its controllers and archi- tecture, it provides details about the issues prevailing in adopting SDN by existing networking giants. Further research was done to see how to look at changed network working frame- works. There are many open-source organize controllers accessible. Be that as it may, to locate the best controller, first, there is a need to set up the significance of "the best controller" or whether there is any importance to such an idea. A paper distributed in 2014, titled "Software-Defined Networking: A Compre- hensive Survey" [7] played out an investigation of the hardware utilized in the software- defined network design. The network utilizes basic switch like gadgets instead of com- pletely useful routers. While data forwarding components got idiotic, the general ef- fectiveness of network traffic altogether improved as routers were adequately straight forwarding packets. The control plane components are presently spoken to by a soli- 9 tary substance called the "Controller" or Network Operating System. Further outsider applications are likewise permitted to be actualized on the network logic with autho- risation from the controller to give a few highlights explicit to the working framework and are a lot simpler to create and convey. So as to analyze controllers, first, there is a requirement for a lot of measurements and apparatuses with which the controller 's performance can be recognized and fur- ther grouped based on network situations. Published in 2016, the paper "SDN Controllers: A Comparative Study" [8] tested the performance of different controllers using CBench. It concluded that

19**controllers coded by the C-language gave the** best **performance and**

when all is said, the Java- based distributed controller OpenDayLight performed well. Published in 2015, the paper "On the performance of SDN controllers: A reality check" [9] also tested throughput and latency using CBench for 5 different controllers namely Pox, Nox, Ryu, Floodlight, and Beacon. Their results concluded

22**that Beacon has** a better **performance.** Also, **they** found **that Ryu is** easy to learn **and**

highly acces- sible. Published in 2018, the paper "Performance Analysis of POX and Ryu with Dif- ferent SDN Topologies" shows a comparative study between two python-based con- trollers and tests them under different network topologies. It concludes that Ryu out- performs Pox in terms of throughput and latency. Now the measurements that were generally used to look at SDN controllers were comprehended, there is a need to discover appropriate instruments and experiments. The following paper provides information as a part of collecting a choice of tools for our analysis. Published in 2014, the paper "Using Mininet for Emulation and Prototyping Soft- ware Defined Networks" [10] is about SDN controller emulation tools like Minine and related techniques.

21**A realistic virtual network** that too **running** on **real kernel** along with **switches and application** codes can be created easily on a given machine like na- tive, cloud, or virtual by the use of Mininet. Published in 2018, the paper “Benchmarking Methodology for Software-Defined Networking (SDN) Controller Performance” [11] characterizes a lot of measurements and comparing systems

17**for benchmarking the control-plane** execution **of Software- Defined Networking (SDN) Controllers.**

10 Published in 2019, the paper titled “SDN Controllers: Benchmarking & Perfor- mance Evaluation” [12] not only show results on comparative performances of various SDN Controllers but also provide a comparative study on multiple tools like CBench. It also includes the CPU Utilisation Analysis, where OFNet is used to send bulk pack- ets. Published in 2012, the paper “A Flexible Open-Flow Controller Benchmark” [13] proposes changes to the OpenFlow protocol as they discovered inherent CPU perfor- mance bottlenecks. It also shows how multiple instances of CBench is used to send packets and measure CPU utilisation. Published in 2014, the paper titled “OFCProbe: A platform-independent tool for OpenFlow controller analysis” [14] shows both OFCProbe and OFCBench being used as a CPU and RAM utilisation monitor and finding bottlenecks for the controller. How- ever, both only works with Ryu, Floodlight, and Nox. It implements the CPU and RAM monitors at the host machine using SMTP messages. Summary The writing review included distributions identified with either clarification of basics and ideas of programming characterized systems administration, or research demonstrating various uses of controllers in programming characterized organizing dependent on true situations. A few papers likewise talked about strategies for looking at controllers dependent on execution and other applicable elements. Various measure- ments utilized for controller execution correlation were utilized, and the importance of those measurements was additionally talked about. The survey clarifies that there is a need for a comparison of different controllers. Previous papers mostly published performance results of controllers based on network size, throughput, and latency. Those who published about system utilisation only include information about CPU and RAM and excluded information about cache memory, etc. which are also major factors in performance. Furthermore, the tools necessary for sending bulk packets to the controller operating system, such as OFNet, is now a dead project and is not avail- able. OFCBench and OFCProbe use SMTP messages to get data from the controller and are limited to a few Benchmarks, whose data is less accurate. 11 Chapter 3 Overview There are diverse SDN controllers accessible. Some of them are open source ven- tures created and kept up by the network, while private associations build up a few. Notwithstanding open source or private, all controllers follow a similar rule of equip- ment autonomy and the partition and overcome approach. Similarly, as with any situ- ation where there are numerous choices of a certain something, a decision is made in light of the fact that we can’t send all the alternatives in a similar domain at the same time. To choose a controller which is best, first, the idea of the best controller must be comprehended like what are the measurement standards that need to be discussed or must confirm whether there is even something termed as a best controller. Like how we measure separation utilising a ruler, or time using a clock, we need a standard against which controllers can be thought about, and afterwards contrast the examina- tions and one another. This standard of measurements is urgent to choosing which controller would be perfect. There are a few papers which notice a couple of essential measurements, let us investigate. Controllers must be analyzed on a few models. Past papers talked about a wide assortment of properties of controllers going from physical execution to adaptation to internal failure. A portion of the measurements incorporate time for setting up the sys- tem, tearing down the system, preparing power Usage, potential to scale the system, CPU use,

memory use, throughput, latency, ping delay, etc. On an industrial level, these above metrics clearly gave an idea of the controller 's performance. Nevertheless, these all criterion seems very well researched and are also evolving with coming hardwares and new features in the tools and controllers. Factors like system utilization also have to be evaluated to determine whether the controller 12 uses the hardware constraints efficiently. It also gives a clear idea about the controller 's nature while it is running and helps identify the system bottlenecks. A paper published in 2018 came up with methodologies for measuring the per- formance of all controller implementations and benchmarked the control- plane per- formance [11]. Another paper, published in 2019, describes some repeated tests by using a bench- marking tool and a network emulator that included the CPU utilization of various Con- trollers on a machine. It uses a benchmark tool which operates on a different system, that sends a considerable amount of random packets to the controller and then reads CPU utilization values. Fig. 3.1 below depicts the average CPU utilization at a given time. The diagram gets populated by information from the benchmarking tool, which plays out the test consistently. In this, a virtual machine is used to run the controller operating system [12]. Currently, the benchmark OFNet used in that experiment is no longer available as the concerned owner has removed it. A paper, published in 2014, describes how OFCBench and OFCProbe can be used as CPU and RAM utilization monitor and find bottlenecks of the controller. The im- plementation of these monitors is by sending SMTP messages from client machines. However, it is worth noting that both these tools are limited to work with Floodlight, Ryu, and Nox Controllers. [14] One more paper, published in 2012, shows they used multiple instances of CBench to send packets and measure the CPU Utilization. They discovered an inherent perfor- mance bottleneck concerning CPU load in OpenFlow switch implementations of those days. [13] The throughput method of CBench in which the switches are arranged to pause and heap up a lot of packets and send all the packets without a moment's delay to the controller. This powers the controller to dispense a huge portion of its resources to each or two switches in turn. This method gauges the pressure limit of the controller. The controller must have enough memory to store and procedure all the solicitations from a switch. It need not have the option to run such a large number of strings simul- taneously. The tests acted in the paper [15] think about numerous controllers, running in a system whose size increments. For this component, they expanded the number of hubs after each arrangement of tests and rehashed them while recording the outcomes. 13 Figure 3.1: CPU Utilization Results Published [12] What appears to be neglected in this test is that the outcomes are legitimate, furnish us just with a fundamental thought of the controller 's performance. The controller with the best development of the response rate in relation to organizing size was delegated the best controller. The controller gives more reactions as the network size increment. It is on the grounds that there is more packets in occasions from switches. As the network size builds, more courses should be determined and kept up. More changes should be refreshed normally. Nonetheless, one factor is ignored. The way that more requests are created isn't really on the grounds that there are more switches; it is on the grounds that there are more has in general associated with the network. The trial proceeded as in paper [16] by expanding the number of switches while keeping up the number of hosts per switch. As the network grows, more has would need to impart, in this way promoting the ascent in route requests or packets in occasions. 14 Traffic Intensity is a measurement technique to test floodibility of a network's traf- fic. As the network gets scaled up, the number of hosts increases, thus increasing traffic Intensity. This expansion in network traffic intensity powers the controller to expand its exhibition rates to adapt to the network necessities. There is a requirement for a strategy to think about the controller with the end goal that base outside elements are impacting the controller 's performance. In the experiment, the traffic intensity is kept very high to perceive how various controllers handle the situation and how their per- formance is influenced. For any test, nature inside which the subject to be tried must be separated from outer factors however much as could be expected. So in the experiment, the cores of CPU where the controller runs, needs to be isolated using isolcpus or similar tool. Chapter 4 Experimental Setup Two open-source controllers operating systems were compared. Ryu and Pox are installed on a server machine. Conversely, Mininet and CBench are introduced on two distinctive host machines in the accompanying equipment and programming arrangements. Table 4.1: Features

4**of Pox and Ryu** [17] **Features Pox Ryu** Platform **Support**

Windows, Mac Linux, Linux Last Updated Nov 2017 May 2020 License Provider Apache Apache Distributed Support No Yes Learning Curve Easy Moderate

4**GUI Yes Yes REST API Yes Yes**

4**OpenFlow Version v1.0 v1.0** to **v1.**

5, Niciria extensions 4.1 Hardware Configuration The setups, as referenced above, were not deliberately picked; rather, it was the asset limitations during the task's time. All things considered, these designs could be additionally changed dependent on rehashed tests. Thus, even the most exceedingly terrible performing controller is best considering the constrained processor and RAM. 16 Table 4.2: Physical constraints of Server and Host Machines

6**Architecture x86 64 CPU op-mode(s) 32-bit, 64-bit CPU(s) 8 Thread(s) per core 2**

Vendor ID GenuineIntel

16**Model name Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz**

11**L1d cache 128 KiB L1i cache 128 KiB L2 cache 1 MiB L3 cache 8 MiB**

4.2 Software & Tools The controller is run on a Linux Ubuntu Server 18.04 LTS OS so that it got dedi- cated RAM and processor cores [18], and 5 out of 8 cores were isolated for the controller using isolcpus tool. Perf was also installed along with the controllers on the server machine itself but on a different single isolated core to prevent influence from any other running process(s). Oracle VM VirtualBox is installed on two separate host machines. An instance of Linux Ubuntu Desktop 16.04.02 LTS OS is created. Mininet is installed on the Host OS and is for setting the switches, hosts, con- trollers, and for setting various possible connection combinations. CBench device copies a lot of hosts per switch and switches and interfaces for

SDN controller indicated. Generally, it creates them in the network and measures the presentation of the controller in 2 modes. • Latency Mode: Packets in these events come in bulk from each switch. • Throughput Mode: Packets in events come in sequential order. However, in our case, we took the usage of CBench in throughput mode and generated massive traffic for the controller. 17

12**Figure 4.1: Experimental Setup Figure 4.2:** Workflow of **the** Experiment The overall **experimental setup** can be viewed **as in** Fig. **4.1. The**

Ryu and Pox along with perf, are installed on server machine, and whenever they are run, were run on isolated cores. In the experiment, 6 isolated cores are assigned for Pox. In 18 one of the isolated core, perf is allowed to run. In the experiment, there are 2 hosts / client machines used. These two machines was set to send traffic to the server where controller is running. In both of the hosts, CBench is installed, and further Mininet is installed on a separate virtual machine in the system. The experiment is carried on as the per the workflow, depicted in the Fig. 4.2. The controller is allowed run on isolated cores. Then the CBench and Mininet is allowed to send random packets to the running controller from other two remote machines. Perf tool is run to record the cores where controller is running for specified time and later stopped. This is done for both Ryu and Pox, and a number of times (here 5 times) to get relatively more accurate information. The information collected is further displayed in graphical chart using Google Chart. Chapter 5 Results & Discussions All the published papers declared results for their experiment where the con- trollers were tested either for CPU or RAM utilization. They performed tests using the OFNet tool to create a vast virtual network, send large packets to the server, and use OFCProbe or OFCBench to monitor the CPU and RAM utilization via the use of SMTP messages. However, instead of OFNet, multiple instances of CBench can also be used to send those packets. In our experiment, we have used multiple instances of CBench and Mininet. We have also used Perf to read the system utilization information directly at the server. After having our experiment, as illustrated in the workflow depicted in Fig. 4.2, the collected raw data is represented in a graphical format. We received and processed the system metrics such as CPU Utilization, Clock Speed, Instructions per Second, Branch mispredictions, and various level cache miss rates. These results are further discussed one by one. 5.1 CPU Utilization Usage of the CPU refers to the use of computing power by a process or the amount of work a CPU performs. Specific tasks require massive CPU time, while oth- ers require less because of non-CPU resource requirements. A method is said to be working better if, for the same number of tasks, it utilizes less CPU. In our result, it shows that RYU continuously utilizes less CPU than POX, de- picted in Fig. 5.1 However, it was not such as seen in Fig. 3.1 taken from the paper published in 2018, that depicted for at least fewer time Pox too utilize the same as Ryu or even less CPU. [12] 20 Figure 5.1: CPU Utilization 5.2 Instructions Per Second Instructions per Cycle (IPC), generally referred to as clock instructions, are one component of processor performance: the total number of instructions per clock cycle. Figure 5.2: IPC With a specific processor, the number of instructions executed per clock is not a constant; it depends on how the individual program running communicates with the processor, and indeed the entire system, particularly the memory hierarchy. Nonethe- 21 less, certain CPU characteristics tend to contribute to designs with higher than average IPC values; the inclusion of several arithmetic logic units (an ALU is a CPU module capable of performing basic arithmetic and logical operations); and short pipelines. When evaluating various instruction sets, a simplified instruction set will contribute to a higher IPC number than applying a more complicated instruction set with the same chip technology; furthermore, with fewer instructions, the more complex instruction set may do more valuable work. The obtained result is depicted in

Fig. 5.2. The efficient use of pipelining can be seen by both the controllers where both scored IPC values greater than 1. However, Pox, in contrast to Ryu, has better IPC. 5.3 CPU Clock Speed CPU check speed is determined in Hertz — in gigahertz (GHz). Clock speed is an element of what number of clock cycles a Processor will execute every second. For instance, a 2.5 GHz clock-rate CPU will execute 2,500,000,000 clock cycles every second. It looks so plain on the face. The more cycles a CPU can play out, the more it can do. Figure 5.3: CPU Clock Speed On average both Ryu and Pox relatively has a similar clock speed of about a 2GHz. However, it is also seen from Fig. 5.3 that Pox even touches a 4GHz score or 22 more when required. Thus, confirming that the controller uses the Turbo Boost features provided by the system used for the experiment. 5.4 L1 Cache Miss Rate If the performance of the cache has to be improved, reducing the missing rate becomes one of the necessary steps. Decreasing access time to the cache also boosts its performance. Figure 5.4: L1 cache miss rate From the experiment as depicted in Fig. 5.4, it can be observed that on average, both Pox and Ryu has 80% and 75% of l1 miss rates respectively. This high miss rate is not enough for any real-time applications. However, during the initiation phase Pox even has the miss rate as high as 90%. 5.5 L2 Cache Miss Rate For level-2 cache, the miss rates are quite less than as compared to level-1 cache. The L2 miss rates for Ryu and Pox is observed to be below 40%. However, while considering controllers as a real-time applications that need to run 24*7, this miss-rate scores seem to be quite high. 23 Figure 5.5: L2 cache miss rate 5.6 Branch Misprediction Branch Predictions are an architectural innovation that provides an alternative to conditional control transfers, enforced by computer commands, such as conditional branch, conditional call, tentative return and tables for branches. Predication oper- ates by implementing instructions from both branch directions, enabling only specific instructions from the path taken to change architectural condition. Figure 5.6: Branch Misprediction Rate 24 The branch indicator means to help stream inside the instruction pipeline. In a few current pipelined microchip designs, for example, x86, branch indicators assume a basic job in accomplishing profoundly successful execution. Indicator inactivity can likewise profoundly affect productivity. Branch Misprediction Rate is the proportion that shows the pace of mispredicted branches. When there is a high rate of a branch misprediction, an optimization (minimiza- tion) problem is used where the emphasis is on achieving the lowest possible miss rate, low power consumption and low resource complexity. As shown in Fig. 5.6, the result clearly shows, for both Ryu and Pox, this rate is quite low, i.e. below 5%. Also, during initialization time (first 10 - 15 secs), Pox has this rate significantly lesser than Ryu. However, overall on average, Ryu has about 22% fewer misprediction rates than Pox. 25 Chapter 6 Conclusion & Future Works The experiments were done successfully on two different SDN controllers (Ryu and Pox) as planned under heavy traffic and with the latest stable releases of both controllers. The results provide a clear picture of their system-level performances. It also shows system performance analysis of SDN controllers is a challenging task. It can be observed that Ryu utilized less CPU than Pox as the former does not support multi-threading while the later does. The utilization of first level cache by both the controllers was reduced and scored more than 70%. However notably, both controllers efficiently use other higher-level caches and branch predictors. Further is also observed that both controllers have a decent score of instructions per cycle, i.e. greater than 1, which signifies the pipelining is used efficiently. Pox even has a better efficiently coded initializing module(s) than of Ryu's, as seen from the lesser branch mispredictions and higher IPC during the starting time. This experiment also makes itself different and competitive from others as it was performed under heavy traffic and analyzed on the latest releases of both controllers. Since OFNet is no longer available [], this experiment successfully shows the efficient mutual use of CBench and Mininet to send random packets. The use of the perf tool in the experiment shows how this tool is capable of mea- suring system-level metrics right from the server itself, rather than using OFCBench or OFCProbe which were limited to Floodlight, Ryu and Nox and also ran on

host machines, sending SMTP messages to get data from server. In future works, a portion of the confinements looked by this exploration could be investigated, for example, the absence of equipment and time. This experiment also displayed how system bottlenecks identification can be made, which will help the developers to improve SDN Controllers running on a server and also identify which 26 system will be better for a given controller or which controller will be better for a given system. This experiment also lays down information to help developers know about the programming lags. It thus will help them write more optimized modules of SDN Controllers to utilize the system better. This type of profiling will also encourage to use the compilers like LLVM, Numba, etc. to develop or improve controllers. Unlike OMNET++, a popular network application used to simulate computer networks has been a part of SPEC CPU Benchmark Suite since 2006; one of the vari- ous SDN Controllers also have a higher possibility to get added to such Benchmark- ing Suites, viewing the increasing popularity among multiple networks and hardware corporate giants, that controller will be developed using the profiling as done in this experiment. Bibliography [1] Geng Liang and Wen Li. “A novel industrial control architecture based on Software- Defined Network”. In: Measurement and Control 51 (July 2018), p. 31. [2] “Graphical representation of computer network topology and activity”. In: (July 1996). [3] Yosr Jarraya, Taous Madi, and Mourad Debbabi. “A survey and a layered tax- onomy of software-defined networking”. In: IEEE Communications Surveys and Tutorials 16.4 (2014), pp. 1955–1980. [4] Larry Roberts. “The Arpanet and computer networks”. In: 2004, pp. 51–58. [5] Hun-Jeong Kang Myung-Sup Kim. “A flow-based method for abnormal net- work traffic detection”. In: IFIP Network Operations and Management Symposium 1 (2004), pp. 599–612. [6] Kamal Benzekki, Abdeslam El Fergougui, and Abdelbaki Elbelrhiti Elalaoui. “Software- defined networking (SDN): a survey”. In: Security and communication networks 9.18 (2016), pp. 5803–5833. [7] Fábio Botelho et al. “SMaRtLight: A Practical Fault-Tolerant SDN Controller”. In: (2014), pp. 1–61. [8] D.A. Tran and H. Raghavendra. “Congestion Adaptive Routing in Mobile Ad Hoc Networks”. In: IEEE Transactions on Parallel and Distributed Systems 17.11 (Nov. 2006), pp. 1294–1305. [9] Y. Zhao, L. Iannone, and M. Riguidel. “On the performance of SDN controllers: A reality check”. In: (2015), pp. 79–85. [10] Rogerio Leao Santos De Oliveira et al. “Using Mininet for emulation and proto- typing Software-Defined Networks”. In: 2014 IEEE Colombian Conference on Com- munications and Computing, COLCOM 2014 - Conference Proceedings (2014). 28 [11] Bhuvaneswaran Vengainathan et al. Benchmarking Methodology for Software-Defined Networking (SDN) Controller Performance. RFC 8456. Oct. 2018. DOI: 10.17487/ RFC8456. URL: https://rfc-editor.org /rfc/rfc8456.txt. [12] Liehuang Zhu et al. “Sdn controllers: Benchmarking & performance evaluation”. In: arXiv preprint arXiv:1902.04491 (2019). [13] M. Jarschel et al. “A Flexible OpenFlow-Controller Benchmark”. In: (2012), pp. 48– 53. [14] M. Jarschel et al. “OFCProbe: A platform-independent tool for OpenFlow con- troller analysis”. In: (2014), pp. 182–187. [15] Justin A Boyan and Michael L Littman·. Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach. Tech. rep. [16] Riten Gupta et al. “Routing in Mobile Ad-Hoc Networks using Social Tie Strengths and Mobility Plans”. In: (2017). [17] Jehad Ali, Seungwoon Lee, and Byeong-hee Roh. “Performance Analysis of POX and Ryu with Different SDN Topologies”. In: (Apr. 2018), pp. 244–249. DOI: 10. 1145/3209914.3209931. [18] Barry D. Wessler Robert Lawrence. “Computer network development to achieve resource sharing”. In: Advanced Research Projects Agency Washington, D. C. (). Annexure - A Tools used for Creating Virtual Environment • Ubuntu Desktop 16.04.2 LTS http : //old − releases.ubuntu.com/releases/16.04.2/ubuntu − 16.04.2 − desktop − i386.iso • Ubuntu Server 18.04 LTS https : //releases.ubuntu.com/18.04.4/ubuntu − 18.04.4 − live − server − amd64.iso • Oracle VM Virtual Box https : //download.virtualbox.org/virtualbox/6.1.8/ Tools used for Emulation of Controllers • Mininet http : //mininet.org/download • Ryu git clone https : //github.com/f aucetsdn/ryu.git • Pox git clone https : //github.com/noxrepo/pox.git • CBench git clone git : //gitosis.stanf ord.edu/of lops.git Tools used for System Monitoring • Perf apt install linux − tools − $(uname − r)linux − tools − generic 30 Annexure - B Creating virtual networks • Mininet Configuration – To create virtual network and attach to the controller Example: sudo mn –topo tree,depth=5,fanout=3 –controller=remote,ip=192.168.5.12,port=6633 – NOTE: Default port used by Ryu is 6653 whereas Pox uses 6633. – To open terminal for any host use xterm Example: xterm h1 h2 h10 • CBench Configuration – Command to run CBench Example: cbench -c 192.168.5.12 -p 6633 -l 100 -s 32 -M 100000 -t This is to run the Cbench tool against 32 swtiches (-s) with 100000 hosts (-M) per switch in throughput mode (-t) or latency mode (-l) at localhost controller ip (-c) on port 6633 (-p) Implementation of Controllers • Ryu Usage: ryu-manager –verbose ryu/app/simple switch 13.py • Pox Usage: pox.py samples.pretty log forwarding.l2 learning Perf Usage • Syntax sudo perf stat -e < events list > sleep < delay >< process ids > 31 • Example sudo perf stat -e task-clock,br misp retired.conditional,branch-instructions,branch-misses, l2 rqsts.miss,l2 rqsts.references,bus-cycles,cache-references,cpu-cycles,instructions sleep 30 14880,14883,14882,14884 32 15 19 27 29