2021 International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT) | 978-1-7281-5791-7/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICAECT49130.2021.9392559

Performance Comparison of Software-Defined Network Controllers

Arash Shirvar and Bhargavi Goswami

School of Computer Science, Queensland University of Technology Brisbane, Australia

***Abstract:*—The concept of Software-Defined Networking (SDN) was introduced to replace the current network infrastructures suffering from being highly hardware-based, inflexible to change and development, error-prone for the configuration of proprietary devices, etc. Optimistically though, SDN is set to address these challenges by its intrinsic merits including its centralized software-based architecture with vendor-neutral devices, flexibility to change and development, the programmability of data transmission functionalities. Nevertheless, it has yet to be fully developed and implemented. One major reason behind this setback is believed to be a lack of advancement of a controller for centralized management of the SDN environment. To achieve a highly advanced and reliable controller, first, it is vital to find the most essential features and capabilities of the current controllers. With this, the paper intends to first, compare ten major existing SDN controllers against various essential criteria and then evaluate the scalability and performance based on various topologies using mininet, an SDN network emulator. Ultimately, after conducting both evaluative stages, the key results and contributions of this paper are 1. A detailed list of all the key features and capabilities that every controller must entail 2. Python-based controllers cannot meet the high performance and low latency in large networks whereas Java and C-based controllers can offer better scalability and performance 3. A verdict on the most suitable controller based on the Multi-Criteria Decision Making technique.**

Keywords— Software-Defined Networking, SDN controllers, mininet, Throughput, Latency, MCDM, Regression Analysis

I. INTRODUCTION

There has been a growing concern over the currently hardware-based network infrastructure with the increasing number of data forwarding devices such as switches and routers, heavily equipped with control requirements and policies. This concern stems from the fact that the traditional networks have become not only complex to be effectively configured and managed but also resistant to innovation for the provision of new services. There are arguably two major reasons behind these challenges. Firstly, the control planes as the brain of networks and the data planes responsible for data forwarding are both rigidly placed inside these heterogeneous and proprietary forwarding devices, as illustrated in Fig. 1. Secondly, the manual configuration of these devices is also overly error-prone and time-consuming [1] [2]. In the wake of these hurdles, however, a promising solution by which the conventional network infrastructures can be transferred to be flexibly programmable to enable innovation has been circulating in the network community for a while now [3]. This idea gradually led to the development of Software-Defined Networking (SDN) coined by and represented at Stanford University in 2009 [4].

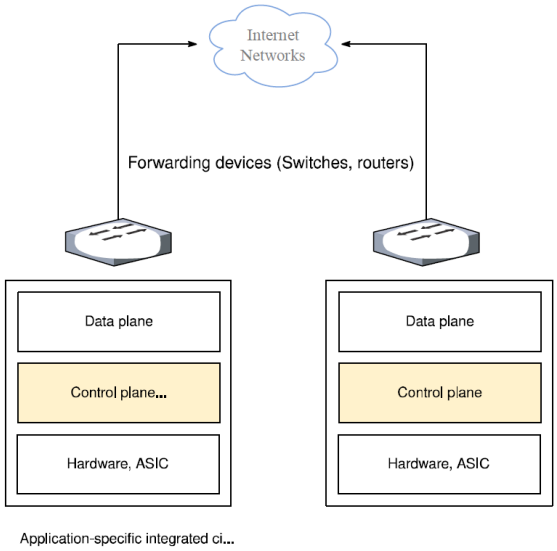


Figure 1. Traditional network infrastructure

SDN is defined as "The physical separation of the network control logic (Control plane), by which forwarding devices can be controlled, from the forwarding plane." according to the Open Networking Foundation (ONF) as the bearer of SDN [5]. This concept has brought the possibility of centralized control and programmability over networks with the help of abstracted layers including a northbound interface for applications and a southbound interface for forwarding devices as shown in Fig.2. Given this advantage, SDN can help the currently rigid network infrastructures be transformed into programmable, vendor-neutral, and software-based infrastructures that are managed through a centralized controller [6]. This special controller will be responsible for providing programmable interfaces and forwarding decision making by abstracting underlying infrastructures for user-written applications to control the behavior of network devices based on a set of high-level rules [7]. Providing the ability to dynamically control forwarding devices, bandwidth, and network resources by applications under the control of a controller are all intrinsic benefits of SDN [8]. Despite this, after a decade, SDN has been utilized by a handful of experimental use

cases such as Google Software-Defined WAN [9] and has not been fully-grownand globally deployed in the current network infrastructures.

The rest of the paper is organized as follows. Section II provides related works on SDN's architecture, challenges, and a detailed list of several works on the evaluation of theSDN controllers' performance. Section III defines the problem formulation that fulfills the goal of the paper. Section IV explains the methodology of this paper to address the research questions. Section V explains and shows the results of the comparison, performance, and latency evaluation of the selected controllers. Section VI concludes the paper.

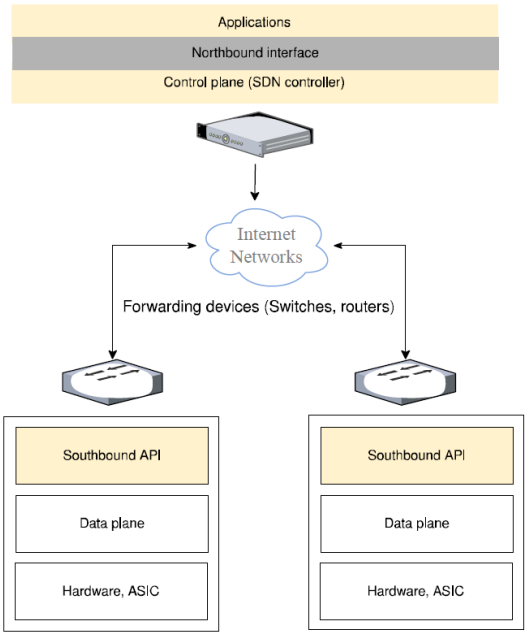


Figure 2. SDN network infrastructure and layers

II. Related Work

Since the advent of SDN, there has been a myriad of ongoing research and works dedicated to each layer of this concept. As such, a range of different use cases as an SDN controller, southbound, and northbound interface has been introduced to date [2]. For example, OpenFlow as the first protocol for the separation of control planes and data planes was introduced by ONF in 2008 [10]. Since the advent of this protocol, it has been considered a standard for the southbound interface of the SDN architecture [30]. However, aside from the OpenFlow and REST API for the northbound interface layer [11], all the current primitive controllers developed thus far have yet to become a de facto standard to push the boundaries of SDN to be effectively developed globally [2]. Given the prime value proposition of SDN that is to allow developers to create innovative applications through automation by using a controller [12], the magnitude of an SDN controller is apparent. Arguably, this setback lies with the fact that the latest prototypes have not holistically managed to meet the requirements of a controller and address the challenges of an SDN network [6]. In the next chapter, the requirements and challenges will be briefly explained. According to some papers, the challenges in the SDN environment include scalability, reliability, high-availability, interoperability, and security [6] [8]. According to one research on the SDN challenges, among others, scalability and security are the main hurdles in the SDN environment owing to the existence of a centralized controller that must be addressed. Regarding the former, by exchanging network packets between different nodes and a controller, a potential latency arises thereby reducing throughput, as the indicator of performance, and thus, this poses a risk to the scalability of SDN networks [12]. According to a comprehensive paper about SDN controllers, some requirements are introduced in the literature to be considered when choosing a controller for the SDN environment. These requirements consist of performance, available north and southbound interfaces, virtualization support, Graphical User Interface (GUI) support, modularity support, Transport Layer Security (TLS) support, cross-platform support, open-source support, and comprehensive documentation support [13]. Similarly, scalability, architecture, and programming language of a controller are also of importance as each utilizes different methods for memory management, modularity, and multithreading [10].

It is now evident that despite the introduction of an accepted protocol for the southbound and northbound interfaces, the selection of a globally accepted controller is yet to be achieved, and thereby existing network infrastructure has not become fully programmable and flexible. With this, two questions are raised that are i. What are the essential features of the recently developed SDN controllers to help developers learn the strengths and weaknesses? ii. Which currently available controllers can be the base, if at all, for the development of a more advanced controller in the future by addressing all the SDN challenges? To address these questions, and considering the paramount importance of an SDN controller, this research aims to provide the network community and developers with a base and a descriptive work of reference to speed the development of a highly advanced controller based on current primitive controllers. To achieve this aim, two specific objectives are also defined which are as follows. Firstly, we will provide an impartial comparison of ten wildly available SDN controllers against several important criteria/features to finally discover their weaknesses and trade-offs. Secondly, to represent the real performance of the selected controllers, seven controllers will be analyzed in terms of their functionalities to address the SDN challenges that are performance and scalability. This analysis will be undertaken using mininet an emulator that provides a network of virtual hosts, switches, and

remotely located SDN controllers, based on different networking topologies [14].

Several works will be also briefly explained as collected in Table I provided at the end of this paper after references for the performance evaluation of SDN controllers and their major findings. All the related work attempted to impartially compare and analyze several SDN controllers based on CBench and Mininet emulator. While we will explain about Mininet in detail, Cbench is a program for testing SDN controllers by creating OpenFlow switches to generate packet-in and packet-out flows to analyze the performance of SDN controllers [15]. Based on Table I, half of the papers compared and analyzed only a limited number of controllers in terms of throughout whereas some both throughput and latency. Moreover, most analyses were undertaken based onthe early version of controllers. Therefore, this report aims to present the recent comparative and analytic evaluation of several well-known SDN controllers in depth.

III. Problem Formulation

To meet the goal of this paper on the selection of a globally accepted SDN controller as mentioned before, we first compare different SDN controllers and then evaluate them in terms of their performance. As this selection will be based on different criteria, an impartial and effective decision-making approach should be adopted. With this, Multi-Criteria Decision Making (MCDM) based on the weighted Sum Model (WSM), has been opted for [31]. As such, S as the total score of each controller will be the sum of all the criterion's weight, as indicated by w, multiplied by the score's level, as shown by a that is given to each criterion of each controller. There are also M controllers/alternatives, N criteria, and based on the following formula of eq. (1).

$$S = \max_{i} \sum_{j=1}^{N} a_{ij} w_j \text{ , for } i = 1, 2, \ldots, M \quad (1)$$

Additionally, the potential scalability of each controller based on their throughput analysis in section V can be also calculated with the help of multiple linear regression analysis [32] based on the formula given in eq. (2), (3) and (4). As such, Y represents the dependent variable that is possible throughput based on bandwidth while a stands for Y-intercept that is the expected mean value of Y when X as independent variables are equal to zero. In this regard, X is the number of switches and cores, b and c denote the slope of a regression line that is the rate of change for Y as X, and ϵ shows the residual/error, which is the difference between the actual value of Y and its projected value.

$$Y = a + bx1 + cx2 + \epsilon \quad (2)$$

As such, intercept/a and slope/b can be also calculated as the below formulas

$$a = \frac{(\Sigma y)(\Sigma x^2) - (\Sigma x)(\Sigma xy)}{n(\Sigma x^2) - (\Sigma x)^2} \quad (3)$$

$$b = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{n(\Sigma x^2) - (\Sigma x)^2} \quad (4)$$

This analysis was carried out using Microsoft Excel and the received results indicate whether a controller can handle the burden of network flows when increasing the number of switches and cores beyond the defined scenarios based on Table IV. The potential scalability of each controller, thus, is the result of this calculation.

IV. Methodology

A. SDN controller comparison

In this section, a qualitative comparison of ten well known and available SDN controllers is carried out to answer the first research question to provide a clear insight into which controller benefits from which features as collected in Table II provided at the end of this paper after references. Some primary criteria are also explained which are as follows:

1) Architecture

The role of an SDN controller is so important that a brief explanation of how they perform their main task, routing and packet forwarding, is worthy of mention. When node A in an SDN environment starts communicating with node B, the first switch receives the first flow from host A and if the current flow table has no information about host B then it sends it to its controller. Next, the controller completes the packet-in based on its services/applications and then encapsulates it to form a packet-out. This will be then sent back to the switch along with all those switches between the source and the destination. In turn, each switch flow table will be populated with the necessary information to make the right routing decision. Based on the programming language, most controllers in this list employ similar techniques for their architecture. They are all open-source and can be implemented on Linux-based operating systems. All controllers also offer a centralized implementation except for IRIS, ONOS, and Runos that provide a distributed implementation.

2) Multithreading

Multithreading is the ability to share multiple threads of execution concurrently in a single processor or a single core in a multi-core processor. Each thread shares the memory space and computing units of a single core in a processor compared with a process that utilizes separate computing resources. The goal of multithreading in an SDN controller is to divide different processing work evenly among available cores of a processor to reduce the memory consumption of a system and higher throughput [23]. All controllers support multithreading other than

python-based controllers such as POX and RYU [24] [33] that are not multi-threaded as they cannot fully utilize this feature to the benefit of their performance due to GUI [29].

3) Switch partitioning and task batching
To deal with switches and flows, there are some techniques a controller can employ. To allocate the connected OpenFlow switches to running threads in a controller, switch partitioning is used while to allocate currently received flows to the running threads for processing, task batching is used. Beacon, Floodlight, and POX utilizes static partitioning and batching whereas Maestro employs shared-queue adaptive baching design [17].

4) Modularity
It is a software design technique that gives a programmer the ability to separate the features and functionalities/services of a program into independent modules. In the case of SDN controllers, a controller is designed with a modular architecture to allow independent development of each module or service. Therefore, any addition of new services would not impact the existing services in the future [25].

5) Northbound and southbound interfaces (APIs)
While northbound interfaces connect applications to underlying networks and an SDN controller, southbound interfaces connect forwarding devices to an SDN controller. The most widely-used southbound interface to date has been OpenFlow protocol while RESTful API has become a de facto for the northbound interface in the SDN environment [13].

6) Scalability
The potential scalability of each controller that is directly impacted by their performance will be determined by the evaluation of each controller's performance in the second IV and regression analysis for the prediction of their ability to keep up sufficient performance as the number of switches and cores increase by finding a relationship between their throughout and the number of switches and cores.

B. Performance evaluation
This section discusses the methodology used for the experimental evaluation of the performance of seven SDN controllers upon which a final verdict will be based. This performance analysis will be undertaken only on seven controllers that support OpenFlow protocols for the southbound interface based on throughput and latency. This analysis will be based on simulating five various network topologies with different numbers of virtual hosts and switches in a simulated-based SDN environment using mininet emulator to evaluate the impact of each topology on each controller's performance and latency. Mininet is an emulator of the SDN environment by utilizing OpenFlow protocol and Open vSwitches to provide the same functionalities of actual forwarding devices [14].

C. Evaluation metrics
1) Performance: The performance of an SDN controller is determined by two characteristics: throughput and latency. The goal of this evaluation is to discover the maximum throughput and minimum latency for each controller.
2) Throughput: The network throughput is the measurement of the number of packets per second an application such as an SDN controller can handle. Two forms of measuring throughput are also the measurement of network packets based on bits per second (bps) while another form is the measurement of the number of network packets or flows (Packet-in and packet-out) per second in the case of SDN. In this paper, throughput will be measured based on the former approach and TCP connection which is calculated from the following formula.

Throughput in bits p/s = TCP Window Size in bits / RTT in seconds To evaluate the maximum throughput of a controller in the SDN environment, Iperf3 will be used based on the concept of the client and server. It generates as many packets/flows as possible and then measures how much packet-in and packet-out a controller can handle in a specific time frame [26].

3) Latency: In general, there are two forms to measure the latency between two nodes in a network. They include, first, measuring the time it takes for a network packet from a source to its destination while the second way is the measurement of the time it takes a network packet to reach its destination and return to the source that is called Round Trip Time (RTT). This paper will follow the latter approach that will be the delay between the packet-ins and the packet-outs to calculate the average latency of a controller in the SDN environment. To achieve this, the approach is based on the next formula for the calculation of average RTT between two nodes by using MTR. MTR is a network diagnostic tool that combines the functionality of the traceroute and ping in asingle attempt to analyze latency between two nodes [27].

RTT = [The SUM of (Time of a received ACK for a packet – Time of the generation of the packet)] / Total number of received packets

4) Scalability: In terms of scalability of an SDN controller as a challenge, the changes in the parameters of both performance and latency when dealing with different network topologies with different numbers of switches and hosts, as well as different numbers of processors, and cores will show the level of the scalability of the target controller. Therefore, the result of the evaluation of both performance and latency will determine the level of

scalability of a controller as a whole.

D. Experimental setup

In this section, the experimental environment will be explained in detail.

1) Implementation: For all stages, several tools were implemented and used as shown in Table III. The processor of the testbed is an Intel Core I7 with 8 cores, 4 physical cores and 4 logical cores, with 16 GB of RAM. For the performance analysis based on multithreading, Oracle VirtualBox for the virtualization environment will be used so that we can customize the number of cores for each test.

2) Evaluation: For this stage, all the controllers should be installed and activated separately on a standalone virtual machine (VM) and, if need be, the layer 2 learning switch module is also to be activated for the connectivity of all nodes. Each VM will be configured with a different number of processors and cores for the benefit of performance analysis based on the multithreading capability of each controller. For both performance and latency, there will also be five various scenarios defining the network topologies with their entities as explained in Table IV. All the topologies will be created by mininet using tree and star topologies as shown in the following command except for topology 2, which is a customized Python-based network topology as shown in Fig. 3. In this regard, depth defines the depth while fanout defines the branches of a tree-based topology. The customized Python-based topology, shown in Fig. 4, helps determine how controllers can handle a customized Pythonbased topology using Mininet. sudo mn --controller=remote,ip=127.0.0.1,port=6633 --topo=tree,depth=x,fanout=y --switch ovsk,protocols=OpenFlow13 The links between nodes also support a maximum speed of 10000 Mbps depending on the power of the processor of the testbed. All the network topologies can also be equated with the network topologies of Internet Service Providers (ISP) and data centers with different scales.

Table III. Performance Evaluation Tools

Tools	Configurations Versions	Reasons
A computer as a testbed	CORE I7 X86 6700HQ 2.6 GHz 4 Cores (8 logical) RAM 16 GB	The performance analysis must be conducted on a multi-core and X86 CPU family
Operating system (OS)	Linux Ubuntu V 20.04.1	Linux Ubuntu is a free and powerful Linux-based operating system benefiting from a strong CLI with a variety of tools
Virtualization Platform	Oracle Virtual Box V 6	This hypervisor is a powerful tool for x86 CPU architecture
Open vSwitch	V 2.13.0	Supporting all the OpenFlow versions 1.0 ~ 1.5
Mininet Emulator	V 2.3.0d6	This allows us to simulate complex network topologies using virtual entities for the SDN environment.
SDN controllers	Stable/Latest versions	Seven publicly available controllers
Iperf	V 3	A tool for the measurement of network performance based on TCP/ UDP
MTR	V 0.93	MTR combines the functionality of the traceroute and ping in a single tool
Gnuplot	V 5.2	It is a powerful and cross-platform plotting tool to design graphs

Table IV. Performance Evaluation Scenarios

Topology	Type	Switch	Hosts
Scenario 1	Star	1	20
Scenario 2	Tree	8	50
Scenario 3	Tree	31	125
Scenario 4	Tree	85	256
Scenario 5	Tree	156	625

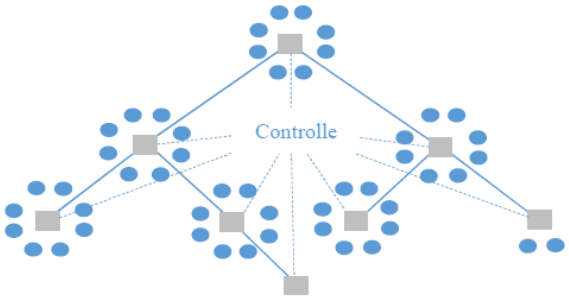


Figure 3. Topology 2

```
#!/usr/bin/python
from mininet.net import Mininet
from mininet.node import RemoteController, OVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
def emptyNet():
    net = Mininet( topo=None, build=False )
    info( '*** Adding controller\n' )
    net.addController('c0', controller=RemoteController,ip="127.0.0.1")
    info( '*** Adding hosts\n' )
    hosts=[]
    for h in range(50): # number of hosts
        hosts.append('h%s' % (h+1))
    info( '*** Adding switch\n' )
    switches = []
    for i in range(8): # number of switches
        switches.append('s%d' % (i+1))
    #create hosts
    for h in hosts:
        globals() [h]= net.addHost(h)
    #create switches
    for s in switches:
       globals() [s] = net.addSwitch(s, cls=OVSSwitch, protocols='OpenFlow13')
    info( '*** Creating links\n' )
    i=n=0;
    for h in hosts:
        net.addLink(h,switches[i])
        n+=1
        if(n==8): # number of host per switch
            i+=1
            n=0
    net.addLink(s1,s2)
    net.addLink(s1,s3)
    net.addLink(s2,s4)
    net.addLink(s2,s5)  # Connectivity of switches
    net.addLink(s4,s8)
    net.addLink(s3,s6)
    net.addLink(s3,s7)
    info( '*** Starting network\n')
    net.start()
    #info('*** Enable spanning tree\n')
    for s in switches:
            globals() [s].cmd('ovs-vsctl set bridge %s stp-enable=true' % (s))
```

Figure 4. Topology 2, Python-based topology

a) Throughput analysis: For both performance and latency evaluations, after running each topology in mininet, the first host and the last node will be selected and configured by using xterm a standard terminal emulator. Host A becomes the server while host B plays the client role in both experiments. To start the performance analysis, hosts A is configured using iperf3 –s –p 5566 –i 1and iperf3 -c 10.0.0.1 -p 5566 -t 60 will be used for Host B. As such aside from –p defining the port of the connectivity, –i defines the time interval that is one second, and –t on the client-side defines the time span of the analysis that is 60 seconds. In terms of the TCP Window size, instead of manually defining it, it will be on the control of Iperf3 based on the following formula.
TCP Windows size = bottleneck bandwidth (Max 10000 Mbps) * round trip time (RTT) between two host A and host B

b) Average RTT analysis: For this evaluation, at first, no connectivity test (Ping) will be undertaken so that the latency test will also factor in the speed at which a control can detect a node and establish a routing path between two nodes. mtr – c 60 –r client's IP will be issued on the server-side for the evaluation of RTT between host A as the server and host B as the client in each network topology. In this regard, –c defines the number of Ping requests that is 60 times, and –r forces mtr to use the report mode for log files.

3) Data filtering: After logging all the data of the performance and RTT evaluation for every seven controllers based on all five topologies, processors, and cores number, it is time to filter the data to finally plot our different graphs. To do so, Gnuplot a powerful plotting tool will be used. This tool should be fed by .dat files with columns of data for the definition of the X-axis and Y-axis. Filtering the data can be attained by using sed command a stream editor for filtering texts in files as well as awk a text processing and report tool. Thus, the following commands will be used for data filtering purposes.
sed -n 67p ./onos-topo1-1core.txt | awk '{print $4}' >> ./log1.tx

As such, sed –n will be used for finding a specific line that the sum of the throughput or the average latency will be placed based on a log file and it will be then piped to awk by which the exact location of those data can be found to finally save the target data in another text file. This new file will be then given to another awk command to add the processor and core numbers as the first column of the .dat file used by Gnuplot later on. This will be attained using FNR as the line number and {print "1 " $1;} to print the specific data on the first field of each line.

awk 'FNR == 1 { print "1 " $1; }' ./test1.txt >> ./onos-topo1.dat

V. Results And Discussion

The result of this paper is based on two objectives of this research paper defined in section I, which are controllers comparison and performance evaluation.

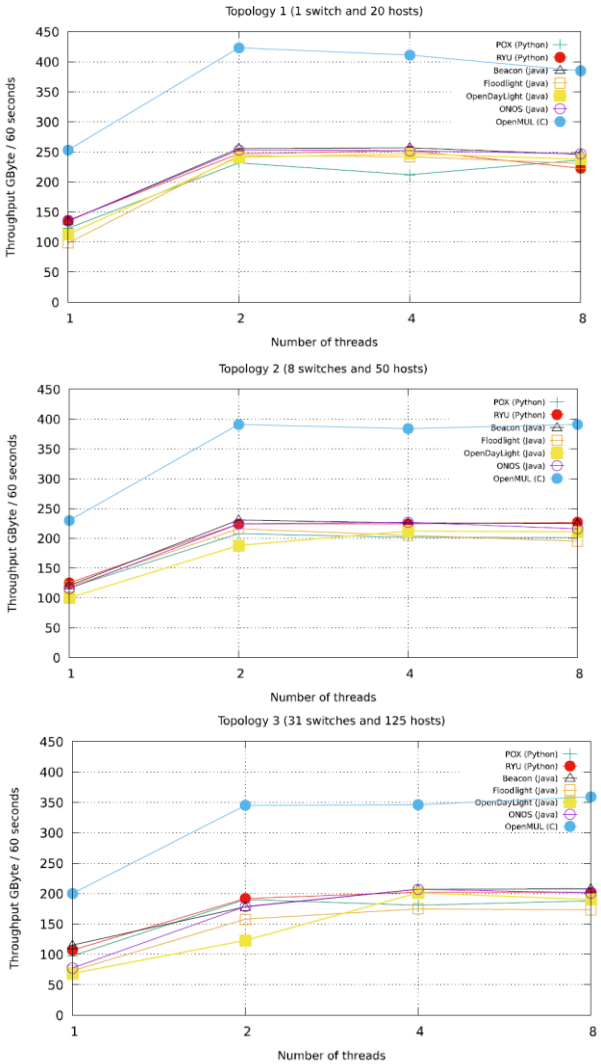
A. SDN controller comparison

We presented a detailed table that affords a qualitative comparison of ten well-known controllers in terms of their features and capabilities against different criteria. This table helps SDN developers determine which controller best suits their purposes in different environments in terms of scalability, performance (Throughput and latency), the APIs used in a controller, and so forth as shown in Table II. By comparing all ten controllers, some important facts can be seen. 1. All controllers benefit from several similar and different features for different purposes 2. The majority of controllers are designed for centralized management except IRIS, ONOS, and Runos that are distributed controllers. 3. The majority support all versions of the OpenFlow protocols for their southbound interfaces. 4. Most controllers benefit from a web-based graphical user interface but Maestro and POX. 5. The majority do not have comprehensive online guidelines and documentation for their users and future developers. Overall, amongst others, IRIS, ONOS, OpenDayLight, RYU, and Runos have relatively provided an up-to-date and powerful base for an SDN controller with essential features such as a reliable architecture, full support of OpenFlow protocol, strong organization support, and comprehensive

documentation.

B. Performance evaluation based on throughput

Multi-cores/threads analysis: This experimental analysis was based on all five network scenarios and a different number of cores/threads of one processor. As such, the Y-axis of all the following graphs, Fig. 5, shows the maximum throughput per 60 seconds against different numbers of threads for the X-axis. Moreover, this analysis determines which controller can effectively utilize different numbers of cores of a processor to the benefit of multithreading based on their programming languages. Based on topology 1 and 2, all controllers tend to achieve relatively equal throughput when using different numbers of threads except for OpenMUL whereas the difference between controllers in terms of throughput is more noticeable when increasing the numbers of threads based on topology 3, 4, and 5. Based on topology 1, OpenMUL best achieved a total of 423 GB when using two core even though by increasing the number of switches and nodes, there is a clear fall in throughput reaching nearly 250 GB in the heaviest scenario as shown in Topology 5. However, the throughput of other controllers when using one core is only between 100 and 150 GB while it reaches a maximum of 250 GB when raising core numbers to 8. Likewise, based on topology 2 and different core numbers, all the controllers performed similarly excluding OpenMUL with the highest throughput just below 400 GB and OpenDayLight with the slightly lowest throughout just above 200 GB. As the number of switches grows, the gap between the amount of throughout each controller achieved start widening. With this, based on topology 3, OpenMUL shows the highest throughput from 200 GB based on one core to more than 350 GB based on eight cores whereas OpenDayLight and Floodlight relatively the lowest from around 75 GB baed on one core to below 200 GB based on eight cores. As depicted in Topology 4 and 5, OpenMUL followed by Beacon both show a higher and gradual increase in throughout when dealing with heavier topologies with more forwarding devices while OpenDayLight, Floodlight, and ONOS show slightly lower throughput reaching just below 100 GB in particular when using only one core. Despite this, these three controllers indicate an effective utilization of multithreading while handling network flows particularly when increasing numbers of physical cores from one to four as shown in Topology 3, 4, and 5. Based on topology 5, OpenMUL, RYU, and Beacon best performed in terms of throughout at 150 GB and nearly 100 GB respectively when using one and two cores whereas OpenDayLight, Floodlight, and ONOS achieved only just under 50 GB when using one and nearly 150 GB when using four cores. It should be emphasized that as only four cores of the processor of the testbed are physical cores, between 4 and 8 processors there is no growth when dealing with lighter topologies with a lower number of nodes and a slight drop can be even noticed when experimenting with the heavier topologies as shown in Topology 4 and 5. Thus, it can be concluded that the selected controllers cannot utilize logical cores for multithreading. In most scenarios, the Python-based controllers, POX and RYU both show a steady throughput as the number of nodes increases. The only clear growth, however, is between one and two processors because POX normally runs on only two threads, one for input and output and one another for cooperative tasks [28]. Additionally, Python-based controllers don't take advantage of multi-cores in a processor as Python Global Interpreter Lock (GIL) allows only one thread to hold the control of the Python interpreter [29]. To have a clear view of the maximum throughput of each controller, considering only four physical cores, Fig 6 illustrates the performance capabilities of each controller against one another based on each topology on the X-axis and throughput per 60 seconds on the Y-axis. From the graph, OpenMUL achieved higher throughput by comparison with gradual decrease when handling more switches whereas Floodlight and POX tend to have the lowest performance. The exact results of each controller will be also shown in Table V.



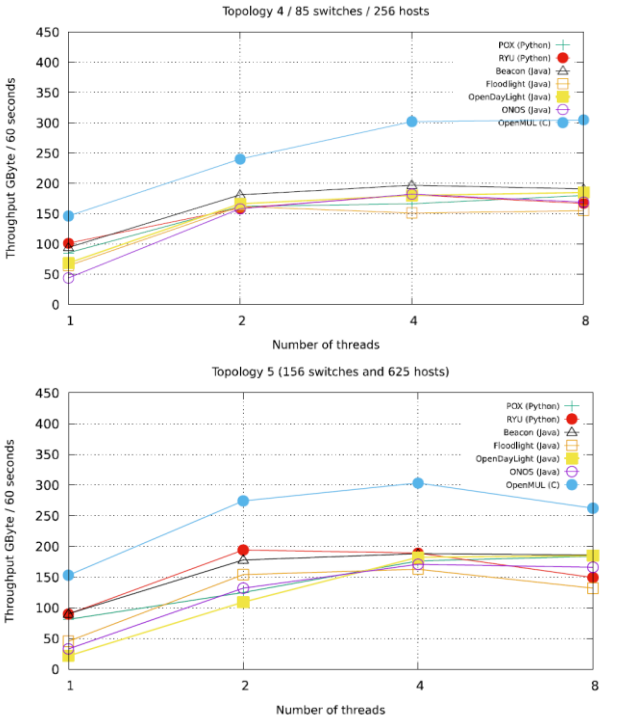


Figure 5. Multi-c ores/threads analysis

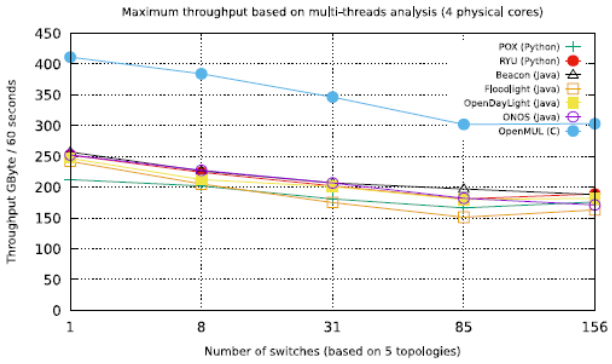


Figure 6. Maximum throughput of each controller

C. Average RTT:

C and Java-based controllers tend to have a lower Round Trip delay (RTT) than Python-based controllers as shown in Fig. 7. Surprisingly enough, OpenMUL demonstrates the lowest RTT despite having the highest throughput. OpenMUL based on C language proves that this programming language can be the best choice for the development of an SDN controller used in the latencysensitive environment. Among Java-based controllers, Floodlight and ODL have both lower RTT but with lower performance as well while Beacon and ONOS show a gradual increase in RTT when handling heavier topologies. Regarding the Python-based controller, however, they have the highest RTT when dealing with heavy topologies due to being an interpreted language and abstracted by CPython. This result could stem from a burden of the time that CPython should spend in interpreting and then executing all codes line by line as Python codes are interpreted at runtime instead of being compiled in advance during the execution. All the RTT times of each controller are also collected in Table V.

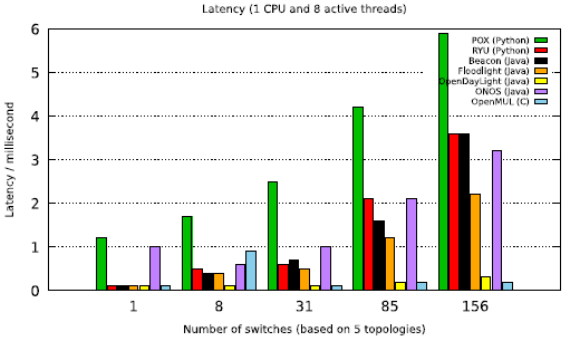


Figure 7. Latency comparison of controllers

D. Discussion

In an attempt to answer both research questions defined in the introduction, we, first, provide a detailed comparison of ten well-known and available SDN controllers as shown in Table I and then an analytical evaluation of seven controllers in terms of performance (Throughout and latency) as depicted in Table V. By analyzing and comparing all the seven controllers together, we believe that this paper comprehensively and partially evaluates different controllers based on their recent versions if any to ultimately find the best controller(s). Compared with other papers, this work provides the most comprehensive and partial comparison of different SDN controllers. We also believe that some controllers should be used for only educational purposes whilst some can be a powerful base for future and highly advanced SDN controllers. With this, based on the programming languages, we impartially attempt to give insight into the suitable controllers for different SDN environments before giving our verdict on the best controller. Among Java-based controllers, Beacon and ONOS offer a powerful base for a more advanced controller with essential features. Among Python-based, POX is a reliably promising controller for a less latencysensitive environment with necessary features and good performance but high latency. However, the only C-based controller, OpenMUL, shows a great level of performance with low latency along with necessary features. This noticeable superiority can be justified by the fact that C language is directly compiled into machine code resulting in removing the overhead of a higher-level programming language such as Java compiled into bytecode and then machine code and Python compiled into CPython and next machine code.

E. Final verdict

To provide our final verdict on the best controller(s), we use the Multi-Criteria Decision Making (MCDM) technique as discussed in section III. Next, four key features against which each controller will be compared were chosen as shown in Table VII. Based on this technique, each criterion needs a weight along with a

level with their scores. On the importance of each criterion, we ranked them from 0.1 to 0.9 and for each level from excellent to poor as shown in Table VI. For example, because POX has no UI interface but CLI, Fair is given to POX for User Interface as a criterion, and because OpenMUL benefits from the highest throughput and lowest latency, Excellent are given for both criteria. In terms of scalability, to give an impartial score to each controller, as discussed in section III, regression analysis was conducted based on the throughput of each controller within five different scenarios. With this, each controller was against three different scenarios including, 10, 300, and 500 switch numbers to predict their ability to handle different numbers of nodes beyond our experimental analysis. Finally, S for each controller as the total score of each controller was derived from the formula mention in section III. Two examples of this calculation are also provided below. By comparison, as it is clear from Table VII, OpenMUL followed by ONOS and OpenDayLight achieve the highest scores respectively. It is of high significance that the ranking of controllers relies on the selected criteria and their weights. Therefore, different results/scores can be seen if we choose different criteria or weights.

Beacon: *S = ((0.9 * 60) + (0.9 * 30)) + (0.8 * 30) + (0.7 * 60) +(0.6 * 30) = 54 + 27 + 24 + 42 + 18 = 165*
OpenDayLight: *S = ((0.9 * 30) + (0.9 * 100)) + (0.8 * 60) + (0.7 * 60) + (0.6 * 60) = 54 + 90 + 48 + 42 + 36 = 243*

Table V. Multi-Cores Analysis Per Gigabyte | Average Latency (RTT) Per Millisecond

Topology	Beacon		Floodlight		ODL		ONOS		OpenMUL		POX		RYU	
	Throughput	Latency	Throughput	Latency	Throughput	Latency	Throughput	Latency	Throughput	Latency	Throughput	Latency	Throughput	Latency
Topo 1	257	0.1	244	0.1	247	0.1	253	1	423	0.1	237	1.2	252	0.1
Topo 2	231	0.4	216	0.4	212	0.1	227	0.6	391	0.9	208	1.7	227	0.5
Topo 3	208	0.7	175	0.5	201	0.1	207	1	359	0.1	190	2.5	202	0.6
Topo 4	197	1.6	162	1.2	185	0.2	182	2.1	305	0.2	180	4.2	181	2.1
Topo 5	188	3.6	163	2.2	185	0.3	171	3.2	303	0.2	184	5.9	194	3.6

Table VI. Final Comparison of Controllers

Controller	Scalability		Distributed architecture Core features & services	User interface	Organization Support Documentation	Total score
	Performance	Latency				
Beacon	0.9 * 60	0.9 * 30	0.8 * 30	0.7 * 60	0.6 * 30	165
Floodlight	0.9 * 30	0.9 * 60	0.8 * 30	0.7 * 60	0.6 * 30	165
OpenDayLight	0.9 * 30	0.9 * 100	0.8 * 60	0.7 * 60	0.6 * 60	**243**
ONOS	0.9 * 60	0.9 * 30	0.8 * 60	0.7 * 100	0.6 * 100	**259**
OpenMUL	0.9 * 100	0.9 * 100	0.8 * 60	0.7 * 60	0.6 * 30	**288**
POX	0.9 * 30	0.9 * 0	0.8 * 30	0.7 * 30	0.6 * 60	108
RYU	0.9 * 60	0.9 * 0	0.8 * 30	0.7 * 60	0.6 * 60	156

Table VII. Criterion Scores For Ranking Controllers

Criteria	weights	Levels	Scores
Scalability and Reliability	0.9	Poor	0
Architecture and Features	0.8	Fair	30
User interface	0.7	Good	60
Support and Documentation	0.6	Excellent	100

VI. Conclusion

Software-Defined Networking (SDN) along with its entities defines a standard on which the data-forwarding tasks can be abstracted by an SDN controller to logically centralize the network intelligence. Since its advent, there has also been a plethora of research papers and proposals to enrich the SDN architecture including application, northbound interface, control, southbound interface, and infrastructure layers. Nevertheless, a highly advanced controller has not been developed and has become a standard to globally develop SDN. To speed this development, thus, the development of an advanced and

reliable controller is vital. Therefore, this paper attempts to shed light on the selection of a suitable controller that is capable of being a base for the future standard SDN controllers. To achieve this, the methodology of this paper is to first compare ten SDN controllers against several essential criteria and then analyze the performance and latency of seven well-known controllers. Based on the result of the performance evaluation, in terms of throughput, Java-based controllers tend to show different results based on their architectural design whereas the only C-based controller, OpenMUL, shows the highest throughput in different network topologies. Python-based controllers, however, show more stable throughout while lower. In termsof latency, Python-based controllers have the highest latency whereas Java and C-based controllers have lower latency in particular OpenMUL and OpenDayLight. Following the comparison and performance evaluation, seven controllers were compared against four essential criteria and for potential scalability regression analysis was conducted and the Multi- Criteria Decision Making (MCDM) technique was used for final decision making. Ultimately, OpenMUL followed by ONOS are regarded as the most suitable controllers with the highest scores, 288 and 259 respectively. To have the most reliable and impartial evaluation for future research, the performance analysis will have to be based on a more powerful testbed with more physical processors and the experiment will be undertaken on more and newer controllers based on various programming languages particularly the C family language. We hope that the impartial findings and contributions of this paper will be utilized for the future standard and highly developed controllers to pave the way for unlocking the full potential development of SDN.

ACKNOWLEDGEMENT

This research and paper writing would not have been achievable without the constant support of my supervisor, Professor Glen Tian.

REFERENCES

[1] Goswami B., Asadollahi S. (2018) Enhancement of LAN Infrastructure Performance for Data Center in Presence of Network Security. In: Lobiyal D., Mansotra V., Singh U. (eds) Next-Generation Networks. Advances in Intelligent Systems and Computing, vol 638. Springer, Singapore. https://doi.org/10.1007/978-981-10-6005-2 44.

[2] Asadollahi Saleh, Goswami Bhargavi (2017) Experimenting with scalability of floodlight controller in software defined networks. In Proceedings of the 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT 2017). Institute of Electrical and Electronics Engineers Inc, United States of America, pp. 288-292.

[3] A. T. Campbell, H. G. De Meer, M. E. Kounavis, K. Miki, J. B. Vicente, and D. Villela, "A survey of programmable networks," ACM SIGCOMM Computer Communication Review, vol. 29, no. 2, p. 1, Apr. 1999, doi: 10.1145/505733.505735.

[4] K. Greene, "Software-Defined Networking (SDN), A reference architecture and open API," MIT Technology Review, 2009. http://www2.technologyreview.com/news/412194/tr10-softwaredefined-networking (accessed Oct. 25, 2020).

[5] Open Networking Foundation (ONF), "Software-Defined Networking (SDN) Definition," Open Networking Foundation, 2011. https://opennetworking.org/sdn-definition/ (accessed Oct. 25, 2020).

[6] Hameed, S. S., & Goswami, B. (2018). SMX algorithm: A novel approach to avalanche effect on advanced encryption standard AES. In Proceedings of the 12th INDIACom (pp. 727–232). IEEE, New Delhi, India.

[7] Saleh Asadollahi, Bhargavi Goswami. (2017) Revolution in Existing Network under the Influence of Software Defined Network. Proceedings of the 11th INDIACom, Pages: 1012-1017, IEEE, New Delhi, India.

[8] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, "Softwaredefined networking (SDN): a survey," Security and Communication Networks, vol. 9, no. 18, p. 9, Dec. 2016, doi: 10.1002/sec.1737.

[9] S. Jain et al., "B4," Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM - SIGCOMM '13, 2013, doi: 10.1145/2486001.2486019.

[10] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, "SDN controllers: A comparative study," 2016 18th Mediterranean Electrotechnical Conference (MELECON), pp. 2, 3, Apr. 2016, doi: 10.1109/melcon.2016.7495430.

[11] Martin Casado, "Origins and Evolution of OpenFlow/SDN – Martin Casado," YouTube. Oct. 25, 2011, Accessed: Oct. 25, 2020. [Online]. Available:https://www.youtube.com/watch?v=4Cb91JTXb4&t=1160s.

[12] Kumar, Ankit, Goswami, Bhargavi, Augustine, Peter (2019) Experimenting with resilience and scalability of wifi mininet on small to large SDN networks. International Journal of Recent Technology and Engineering, 7(6S5), pp. 201-207..

[13] B., Wilson S., Asadollahi S., Manuel T. (2020) Data Visualization: Experiment to Impose DDoS Attack and Its Recovery on SoftwareDefined Networks. In: Anouncia S., Gohel H., Vairamuthu S. (eds) Data Visualization. Springer, Singapore. DOI:https://doi.org/10.1007/978-981-15-2282-6 8

[14] Asadollahi Saleh, Goswami Bhargavi (2017) Experimenting with scalability of floodlight controller in software defined networks. In Proceedings of the 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT 2017). Institute of Electrical and Electronics Engineers Inc, United States of America, pp. 288-292.

[15] Mohammed Sameer, Bhargavi Goswami, (2018), Experimenting with ONOS Scalability on Software Defined Network, Journal of Adv Research in Dynamical Control Systems, 14-Special Issue, Vol. 10, Pg. 1820-1830

[16] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of SDN/OpenFlow controllers," Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia on - CEE-SECR '13, pp. 1–6, 2013, doi: 10.1145/2556610.2556621.

[17] Manuel, Tony, Goswami, Bhargavi H. (2019) Experimenting with scalability of Beacon controller in software defined network. International Journal of Recent Technology and Engineering, 7(5S2), pp. 550-555.

[18] L. Zhu, M. M. Karim, K. Sharif, F. Li, X. Du, and M. Guizani, "SDN Controllers: Benchmarking & Performance Evaluation," arXiv:1902.04491 [cs], Feb. 2019, [Online]. Available: https://arxiv.org/abs/1902.04491.

[19] M. T. Naing, T. T. Khaing, and A. H. Maw, "Evaluation of TCP and UDP Traffic over Software-Defined Networking," IEEE Xplore, Nov. 01, 2019. https://ieeexplore.ieee.org/abstract/document/8921086 (accessed Oct. 26, 2020).

[20] M. Darianian, C. Williamson, and I. Haque, "Experimental evaluation of two OpenFlow controllers," 2017 IEEE 25th International Conference on Network Protocols (ICNP), pp. 1–6, Oct. 2017, doi: 10.1109/icnp.2017.8117602.

[21] R. Jawaharan, P. M. Mohan, T. Das, and M. Gurusamy, "Empirical Evaluation of SDN Controllers Using Mininet/Wireshark and Comparison with Cbench," 2018 27th International Conference on Computer Communication and Networks (ICCCN), pp. 1–2, Jul. 2018, doi: 10.1109/icccn.2018.8487382.

[22] S. A. Shah, J. Faiz, M. Farooq, A. Shafi, and S. A. Mehdi, "An architectural evaluation of SDN controllers," 2013 IEEE International Conference on Communications (ICC), pp. 3504–3508, Jun. 2013, doi: 10.1109/icc.2013.6655093.

[23] Z. Cai, A. L. Cox, and T. S. E. Ng, "Maestro: A System for Scalable OpenFlow Control," scholarship.rice.edu, p. 3, Dec. 2010, Accessed: Oct. 26, 2020. [Online]. Available: https://hdl.handle.net/1911/96391.

[24] RYU, "Ryu application API — Ryu 4.34 documentation," ryu.readthedocs.io,2020. https://ryu.readthedocs.io/en/latest/ryu_app_api.html (accessed Oct. 26, 2020).

[25] Anirban Paul, "Network Controller High Availability," docs.microsoft.com, Aug. 08, 2020. https://docs.microsoft.com/enus/windows-server/networking/sdn/technologies/networkcontroller/network-controller-high-availability (accessed Oct. 26,2020).

[26] M. Gates and A. Warshavsky, "iperf(1) — iperf — Debian testing —Debian Manpages," manpages.debian.org, Apr. 01, 2008.https://manpages.debian.org/testing/iperf/iperf.1.en.html (accessed Oct. 26, 2020).

[27] T. Cross, "Official repository for mtr, a network diagnostic tool," GitHub, Oct. 25, 2020. https://github.com/traviscross/mtr.

[28] M. Mc, "noxrepo/pox-document," GitHub, Feb. 2018.https://github.com/noxrepo/pox-doc/blob/master/include/apis.rst (accessed Oct. 26, 2020).

[29] T. Wouters, "Global Interpreter Lock – Python Wiki," wiki.python.org, Aug. 2017. https://wiki.python.org/moin/GlobalInterpreterLock

[30] Open Networking Foundation (ONF), "ONF SDN Evolution," 2016. [Online].Available:https://opennetworking.org/wpcontent/uploads/2013/05/TR-535_ONF_SDN_Evolution.pdf

[31] Multi-criteria Decision Making Methods, "Multi-criteria Decision Making Methods - A Comparative Study | Evangelos Triantaphyllou |Springer," Springer.com, 2019. https://www.springer.com/gp/book/9780792366072 (accessed Nov. 08, 2019).

[32] S. Cheusheva, "Linear regression analysis in Excel," Excel tutorials, functions and formulas for beginners and advanced users -Ablebits.com Blog, Aug. 2018. https://www.ablebits.com/officeaddins-blog/2018/08/01/linear-regression-analysis-excel/.

[33] S. Asadollahi, B. Goswami, and M. Sameer, "Ryu controller'sscalability experiment on software defined networks," 2018 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC), pp. 1, 2, Feb. 2018, doi: 10.1109/icctac.2018.8370397.

TABLE I. RELATED WORK

References	Authors	Controllers	Evaluation tools	Testbed specifications	Evaluation metrics	results
16	Alexander Shalimov et al. 2013	NOX, POX, Beacon, Floodlight, OpenMUL, Maestro, Ryu	Cbench	Intel Xeon E5645 with 6 cores (12 threads), 2.4GHz, and 48 GB RAM	Throughput Flows	With a 12 CPU threads configuration, m throughput is shown by Beacon with 7 billion flows p/s.
17	David Erickson 2010	NOX, POX, RYU, Floodlight, Maestro	Cbench	16 physical cores from 2 x Intel Xeon E5-2670 processors, 60.5GB RAM	Throughput Latency	Beacon focuses on being developer-friendly, high performance. Beacon showed surprisingly high performance and was able to scale linearly with processing cores, handling 12.8 million Packet-In messages p/s with 12 cores.
18	Liehuang Zhu et al. 2019	NOX, Beacon, Floodlight, Maestro, POX, ODL, OpenMUL, RYU,	CBench PktBlaster, OFNet	Intel 2.10 GHz i7-3612QM processor 12 GB of DDR3 RAM	Throughput Latency	Simulation/Emulation based evaluation can give only an indication of performance at best, and may significantly vary from actual production environment evaluation.
19	May Thae Naing et al. 2019	ONOS	Mininet	Intel(R) Core (TM) i5-5200U CPU @ 2.20GHz 2.19GHz / HP Laptop PC with 12 GB RAM	Throughput Packet loss	According to the results, the throughput of TCP falls off during the retransmission sequence and throughput is changing according to varying the packet size with a different buffer size of TCP & UDP traffic.
20	Mohamad Darianian et al. 2017	ONOS, ODL	Cbench	Cisco UCS C240 M4SX servers. 2 Intel Xeon E5-2670 CPUs (24 cores @ 2.30 GHz), 256 GB RAM	Throughput Latency Thread Scalability	The performance of an OpenFlow controller depends on different factors: controller configuration, underlying networks, the number of switches and threads, and the placement of threads across sockets. Enabling Hyper-threading results in an increase in performance for both.
10	Ola Salman et al. 2016	NOX, Beacon, Floodlight, Maestro, POX, IRIS, ODL, OpenMUL, RYU, LIBFLUID	Cbench	Intel Core™ i7-3630QM CPU @ 2.40GHZ x 8 (8 cores). 16 GB of memory	Throughput Latency	Due to the diversity of SDN applications and controller's uses, the choice of the best controller will be application dependent. OpenDaylight is also a good choice as a full-featured controller.
21	Ragaharini Jawaharan et al. 2018	ONOS, OpenMUL, POX	Mininet Cbench	Intel Xeon E5-2630 server with 64GB RAM.	Throughput Latency	We showed that Cbench underestimates the performance by up to 96% for controller latency and 98% for controller throughput.
22	Syed Abdullah Shah et al. 2013	NOX, Beacon, Floodlight, Maestro	Cbench	HP ProLiant DL160se G6 Server	Throughput Flows	Controllers designed for high throughput should use static switch partitioning and packet batching while those designed for delay-sensitive applications should use workload adaptive packet batching and reduce per-packet latencies. Beacon shows the best result.

TABLE II. CONTROLLER COMPARISION

Controllers	Architecture	Programing based	Southbound Interfaces	Northbound Interfaces	User interface	Organization support	Documentation Community Size	Scalability
Beacon V1.4	Centralized Multithreading Static partitioning Static batching	Java	OpenFlow 1.0	RESTful API	Web UI CLI	Stanford University	Fair Small	Performance: High Latency: Medium
Floodlight	Centralized Multithreading Modularity Static partitioning Static batching	Java	OpenFlow 1.0 – 1.5	RESTful API	Web UI CLI	Big Switch Networks	Thorough Small	Performance: Low Latency: Low
IRIS v2.2.1	Distributed Multithreading Modularity	Java	OpenFlow 1.0.1 ~ 1.3.2	RESTful API	Web UI CLI	Electronics and Telecommunications Research Institute	Thorough Small	Information not available
Maestro 0.1.0	Centralized Multithreading Modularity Adaptive batching	Java	OpenFlow 1.0	RESTful API	CLI	Rice University	Basic Small	Information not available
ONOS V2.3	Distributed Multithreading Modularity	Java	P4, OpenFlow 1.1 ~ 1.3, NETCONF, SNMP, RESTCONF	RESTful API	Web UI CLI	Open Networking Foundation	Thorough Big	Performance: Average Latency: Average
OpenMUL	Centralized Multithreading Modularity	C	OpenFlow 1,0, 1.3, 1.4 OVSDB, OFCONFIG	RESTful API	Web UI CLI	OpenMUL	Fair Small	Performance: Very high Latency: Low
OpenDayLight V8	Centralized Multithreading Modularity	JAVA	OpenFlow 1.0 ~ 1.4, NETCONF/ YANG, SNMP	RESTful API	Web UI CLI	Linux Foundation	Thorough Big	Performance: Average Latency: Low
POX	Centralized Multithreading Static partitioning Static batching	Python	OpenFlow 1.0	RESTful API	CLI	Stanford University	Thorough Small	Performance: Average Latency: Very high
RYU	Centralized Single-threaded using eventlets	Python	OpenFlow 1.0 ~ 1.5	RESTful API	Web UI CLI	NTT Ltd.	Thorough Small	Performance: Average Latency: High
Runos V2	Distributed Multithreading Modularity	C++	OpenFlow 1.3	RESTful API	Web UI CLI	Applied Research Center for Computer Networks	Thorough Small	Information not available