I.J. Information Technology and Computer Science, **2018, 11, 1-13**
Published Online November 2018 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2018.11.01

Design and Implementation of a Benchmarking Tool for OpenFlow Controllers

Eric Gamess
Jacksonville State University, MCIS Department, Jacksonville, AL 36265, USA
E-mail: egamess@jsu.edu

Daniel Tovar and Alberto Cavadia
Central University of Venezuela, School of Computing, Caracas, Venezuela
E-mail: alejorod18@gmail.com, osoflash_2@hotmail.com

Received: 11 September 2018; Accepted: 27 September 2018; Published: 08 November 2018

***Abstract*—**The growth of data traffic on the web, the virtualization of services, and the changes in the pattern of traffic between users and data centers have led to a reassessment of the current methods of doing network administration. Software Defined Networks (SDNs) propose a paradigm that delegate the control of packets and flows to applications, developed according to specific requirements, where the OpenFlow protocol can be used for communications. The development of this type of applications, as in any other development area, requires tests and measurement tools to facilitate a performance evaluation. However, the current open-source performance measurement applications for SDN networks cover only very basic characteristics, while there is a wide range of SDN controllers with support to many versions of OpenFlow, making the selection of the controller a difficult point to address. In this paper, we propose a distributed performance evaluation tool for SDN controllers, that can assess the throughput, latency, percentage of memory consumption, percentage of CPU utilization, and consumption in kB for input/output interfaces, using OpenFlow version 1.3. Our tool is based on Cbench, and adds new functionalities such as the graphical representation of results to analyze the outcomes. To validate our tool, we make a performance evaluation of well-known SDN controllers such as Ryu, OpenDaylight, OpenMUL, and Floodlight, in environments under great stress of requests.

***Index Terms*—**SDN, SDN Controller, OpenFlow, Performance Evaluation, Ryu, OpenDaylight, OpenMUL, Floodlight.

I. Introduction

SDN (Software Defined Networking) [1,2] is a new paradigm to experiment and cover new requirements, without interrupting the existing networks, that is leading to innovative network architectures and management systems. In SDN networks, the intelligence is logically centralized in the controllers, which support several protocols and standards for communication with the managed devices, where OpenFlow [3-5] seems to be the more common. OpenFlow is an emerging and open protocol that allows the controllers to communicate with OpenFlow switches, to create the forwarding path for flows of packets, inside networks.

Many SDN controllers have been proposed with diverse characteristics. Hence, it can be difficult for a network administrator to make a decision about which controller to select. An objective evaluation will be very helpful, that should include performance measures. As evidenced, a tool to evaluate the performance of SDN controllers can facilitate decisions.

In this research work, we propose a distributed performance evaluation tool for SDN controllers, using OpenFlow as the communication protocol. With our tool, users can assess the throughput, latency, percentage of memory consumption, percentage of CPU utilization, and consumption in kB for input/output interfaces, of SDN controllers. We used our benchmarking tool to compare four famous SDN controllers: OpenDaylight, Ryu, Floodlight, and OpenMUL. Our experiments show that OpenMUL and Floodlight have a similar performance. The lowest results were obtained by Ryu. We also validated our tool with Cbench, a famous open-source program to assess SDN controllers.

The paper is structured as follows. Section II discusses the related works. In Section III, we introduce the concepts related to SDN. Section IV describes the main features of the OpenFlow protocol. We introduce the most popular SDN controllers available and use them to validate the operation of our assessment tool, in Section V. Section VI describes the architecture, development, and functionalities of the proposed measurement tool. We discuss the results achieved in our tests in Section VII. Finally, Section VIII concludes the paper and gives directions for future work.

II. Related Works

Many works have been done to compare SDN network controllers with respect to architecture, efficiency, and controllers' features. For example, the authors of [6]

discussed five different versions of OpenFlow switch standard (1.0, 1.1, 1.2, 1.3, and 1.4), four different platforms for simulation and emulation of SDN (Mininet, EstiNet, ns-3, and Trema), seven types of controllers (NOX, POX, Floodlight, OpenDaylight, Ryu, Mul, and Beacon), and different switch software and tools. However, there are just a few works that contrast them in the area of performance evaluation.

Hassan and Ahmed [7] developed Cbench, an open-source tool to evaluate the performance of OpenFlow controllers. The authors realized experiments to measure the performance and latency in testbeds with different amounts of SDN switches controlled by a single SDN controller, ranging from 1 switch to 128 switches. In the performance tests, "Packet-In" messages were sent to measure the stress capacity supported by the controller. The aim was to obtain the maximum number of messages that can be handled by the controller in a specific amount of time. Hassam and Ahmed experimented with six SDN controllers: NOX, Beacon, Floodlight, Maestro, OpenMUL, and OpenIRIS.

Jarschel, Lehrieder, Magyari, and Pries [8] presented OFCBenchmark, an OpenFlow benchmarking tool that creates a set of virtual switches, which can be configured independently from each other to emulate a certain scenario, and can keep their own statistics. OFCBenchmark provides statistics of response rate, response time, and number of unanswered packets for each switching device. Unlike Cbench [7] which offers aggregated statistics of controller throughput and response time for all the switching devices, OFCBenchmark gives fine-grained statistics for individual switching devices.

In their work, Darianian, Williamson, and Haque [9] conducted an experimental evaluation of two open-source distributed OpenFlow controllers (ONOS and OpenDaylight), using Cbench. They reported throughput, latency, and thread scalability of these two controllers in both physical and virtualized (OpenStack) environments. The experimental results show that ONOS provides higher throughput and lower latency than OpenDaylight, which suffers from performance problems on larger networks.

In [10], Xiong, Yang, Zhao, Li, and Li proposed a novel analytical performance model of OpenFlow networks based on queueing theory. They modeled the packet forwarding of OpenFlow switches and the "Packet-In" message processing by the SDN controller, as the queueing systems MX/M/1 and M/G/1, respectively. Subsequently, they built a queueing model of OpenFlow networks in terms of packet forwarding performance and solved its closed-form expression of average packet sojourn time and the corresponding probability density function.

Bholebawa and Dalal [11] made a performance evaluation of two famous SDN controllers (POX and Floodlight) by analyzing network throughput and round-trip delay using Mininet [12,13], an efficient network simulator. They used four topologies (single, linear, tree, and user-defined) to connect the switches together, and an external controller. According to their experiments, Floodlight outperforms POX, for both round-trip time and throughput.

Rowshanrad, Abdi, and Keshtgari [14] evaluated Floodlight and OpenDaylight, in terms of latency and loss, with Mininet. They used different topologies (single, linear, and tree) and network loads. Their results show that OpenDaylight outperforms Floodlight in low loaded networks and also for tree topologies in mid-loaded networks in terms of latency. Furthermore, Floodlight can outperform OpenDaylight in heavily loaded networks for tree topologies in terms of packet loss, and in linear topologies in terms of latency. The authors also concluded that there is no significant difference in performance between Floodlight and OpenDaylight in other cases.

In [15], Wang, Chiu, and Chou used the EstiNet OpenFlow network simulator and emulator to compare two open-source popular OpenFlow controllers - Ryu and NOX. They studied the behavior of the controllers when they control a network with loops and how quickly they can find a new routing path for a greedy TCP flow after a link's status has changed. Their simulation results show that (1) Ryu results in the packet broadcast storm problem in a network with loops; (2) Ryu and NOX have different behavior and performance in detecting link failure and changing to a new routing path.

III. Software Defined Networking

Since their creation, communications networks have required management. Initially, it was a difficult task, given the number of different network devices (switches, routers, firewalls, etc) from different manufacturers and with different configuration environments. To facilitate the task, network administration protocols emerged (e.g., SNMP [16]) that solved the problem of the numerous manufacturers, but were very limited in relation to the direct administration of devices with proprietary protocols and interfaces, hence, making it impossible to maintain a single interface for administration.

In response to the need to improve the administration and versatility of networks, Software Defined Networks (SDNs) [1,2] were developed. They are founded on a set of techniques to make the networks programmable. The traditional architecture of the network devices is modified and now consists of two planes: data and control.

In a conventional network, when a packet arrives at a switch, the rules integrated into the proprietary firmware of the switch decide what to do with the packet. Through the same output port, the switch forwards all the packets of a flow to the same destination. In an SDN network, it is possible to have access to the switch flow tables, in order to modify, eliminate, or add new flow rules, which allows controlling the traffic of the network in a dynamic way.

SDN is a new paradigm that separates the control plane from the rest of the layers of the devices as shown in Fig.1 [8]. Most data are processed only by the data plane of the switches; this plane consists of a series of ports that

are used for the reception and retransmission of data, according to the flow tables. Thus, the data plane is responsible for the reception, storage, and dispatch of data, together with the modification of headers when necessary.

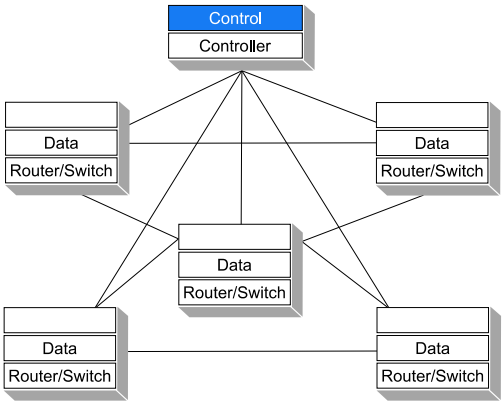

Fig.1. Architecture of SDN.

The control plane is physically separated from the SDN switches, so that routing decisions are made outside the switches, in a centralized device known as the SDN controller, dedicated to answering requests from SDN switches, and configured through APIs.

The logical structure of a software defined network is shown in Fig. 2 [17]. As previously stated, the central controller performs all complex functions, including routing, policy declaration, and security checks. This constitutes the SDN control plane, and consists of one or more SDN servers.

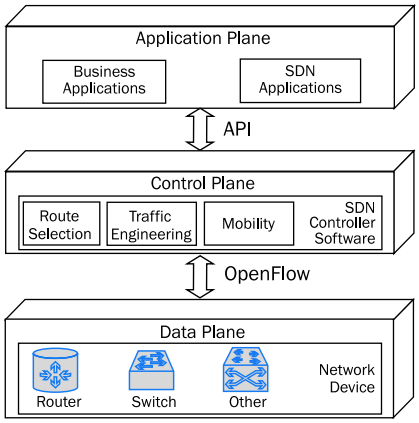

Fig.2. Logical Structure of Software Defined Networking.

As soon as a packet arrives at the network device, it looks for a match in its flow table. If the table has an entry for this flow, it forwards the packet through the indicated output port. If there is no entry in its flow table, the network device encapsulates the packet and sends a request to the SDN controller (Packet-In). According to its policies, the SDN controller decides whether the flow is permitted in the network. If so, it finds a path for this flow in the network and adds the necessary entries in the flow table of each node of the chosen path. Subsequently, the network devices forward the packet based on the entry of the flow table specified by the controller.

With all the complex functions absorbed by the SDN controller, the network devices (SDN switches) only handle the data flows, with the entries in their flow tables. The communications between the SDN controller and the other network devices are made through standardized protocols (Southbound protocols); one of the most common is the OpenFlow [3-5] protocol. The controller is responsible for handling the flow tables of the network devices. The management of the SDN controller is done through a flexible API, that facilitates a dynamic behavior of the network, without having to deal with low-level aspects of the network.

In the application layer, network developers can design programs that perform various tasks without needing to know how the network devices operate, so they can abstract these concepts and be dedicated solely to the development of the logic of the applications. This allows a faster development and deployment of new applications that direct network flows to meet specific needs in terms of security or performance.

IV. OpenFlow

OpenFlow [3-5] is an open standard designed as a communication protocol between the control and data planes of SDN. It was developed at Stanford University in 2008 [18] with the purpose of allowing research and innovation on existing infrastructures, under the usage of multiple flows, that do not interfere with production traffic.

SDN promotes the centralization of the control plane and creates multiple flows with programmable characteristics defined through a high level of abstraction. Hence, through OpenFlow, users can experiment with new protocols to test the current paradigms that define the way in which we define and create networks. OpenFlow is based on the premise that the control plane should be separated from the data plane. In the architecture, the controller takes care of the control plane, and the network devices (SDN switches) maintain the data plane, both intercommunicating through the OpenFlow protocol, as can be seen in Fig. 3.

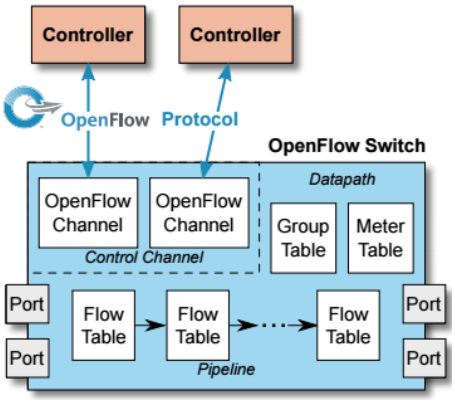

Fig.3. Structure of OpenFlow.

OpenFlow switches are categorized into two groups, (1) “dedicated” and (2) “hybrid”. Those that fall into the first

group are unable to perform common layer 2 and layer 3 processing, and those that enter the last group are common switches and routers, with added support for the OpenFlow protocol. The OpenFlow ports are similar to network interfaces for the transfer of packets. They can be classified as physical, logical, and reserved, where the last two ones are virtual ports. Therefore, OpenFlow defines a number of available ports for processing, which does not necessarily match the number of physical interfaces in the switch.

An OpenFlow switch has at least one flow table, where an action is associated with each entry in the table, and a secure channel that allows communications with the controller [19]. These tables outline the OpenFlow flows that define the data plane. For a switch to achieve high performance at low cost, the entries are based on flows, instead of packets. Each flow entry has a series of associated instructions. These may involve the modification of the actual packet or the list of actions. Some of the instructions can redirect the packet (including new metadata and updated fields) to another table where it will be treated in the same way. If the searching process in the flow table does not result in a hit, the actions of the switch will depend on the instructions defined in case of failure. This entry specifies the series of actions to perform when the incoming packet does not meet any of the entries in the table [20].

Packets are accompanied by a group of actions, initially empty. The flow entries modify this group with instructions that can add actions such as Write-Action. This group of actions follows the packet through tables and can be modified in each table until the end of the navigation. When the navigation is finished, all the actions of the group will be executed.

The OpenFlow messages that allow the communication between the SDN switches and the controller are exchanged on a secure channel. The messages start with the OpenFlow header. This header specifies the version of the OpenFlow protocol, the type of message, the length of the message, and the transaction ID of the message [21]. The OpenFlow protocol defines three message types, each with multiple subtypes:

- Controller-to-switch: These messages are initiated by the controller to handle or inspect the state of a switch and may or may not require a switch response.
- Symmetric: They are messages sent without prior request, in both directions. Hello messages are typically sent back and forth between the controller and switches when the connection is first established. Echo request and reply messages can be used by either the switches or controller to measure the latency or bandwidth of a controller-switch connection.
- Asynchronous: These are messages sent to the controller without having been previously requested, since they communicate the arrival of packets, changes of states, or errors. A typical example is the Packet-in messages, which may be used by a switch to send a packet to the controller when there is no flow-table match.

Although it is usual to talk about OpenFlow in IP networks, it is not necessary for the packets to have any special format as long as the flow tables can compare the fields of the packet header. This allows the experimentation with new addresses, routings, and naming schemes. For example, a flow can be identified through the MAC addresses using a new value in the EtherType field, or at the IP level with a new IP version[22].

V. SDN Controllers

The controller is the main element of an SDN network and is considered as its operating system. The controller centralizes the decisions for all the communication that passes through the devices and provides an overview of the network. A general description of an SDN controller would be: software system or collection of systems that together provide [23]:

- A management of the state of the network
- A high-level data model that captures the relationship between managed resources, policies, and other services provided by the controller.
- An application programming interface which exposes the services offered by the controller for applications.

To validate our performance measurement tool, we use several SDN controllers: OpenDaylight, Floodlight, Ryu, and OpenMUL. The choice was based on the supported features, the commercial competitiveness, the current use of the controller in projects in development or networks already established for production, and its future projection.

A. OpenDaylight

OpenDaylight is an open source collaborative project, carried out by the Linux Foundation in collaboration with numerous companies in the area of computing and networking such as Cisco Systems, Dell, Ericsson, HP, Intel, etc. OpenDaylight can be the central component of any SDN architecture, allowing users to reduce operational complexity, extend the lifetime of existing networks, and enable new services and capabilities only available in SDN.

The OpenDaylight project is one of the most recent projects to develop an open-source SDN controller. OpenDaylight is developed in Java, has several distributions, and a series of complements whose objective is to allow a more transparent interaction with the controller. For example, OpenDaylight can work with Karaf [24,25], a generic platform that provides high-level functions and services designed specifically for the creation of servers based on OSGi [26,27] (Open Service Gateway Initiative). OSGi is a modular system and a service platform for the Java programming language. It

allows Java packages to be remotely installed, started, stopped, updated, and uninstalled without requiring a reboot, in a simple way.

By default, OpenDaylight does not install specific functionalities. After the installation of the controller, the additional functionalities can be installed using the Karaf console. Through Karaf, users can select the required functionalities from a list that is comprehensive enough to suit most network environments.

OpenDaylight is an infrastructure in which the southbound, located on the border with network elements, can work with multiple protocols, such as OpenFlow 1.0 and 1.3, BGP-LS, etc. Another important feature of its architecture is the SAL (Service Abstraction Layer), which exposes services to the superior modules, and complies with the requested services, independently of the underlying protocol that is used between the controller and the network devices. This provides protection against protocol changes or versions changes of the same protocol, over time. Fig. 4 [28] depicts the structure of the OpenDaylight controller.

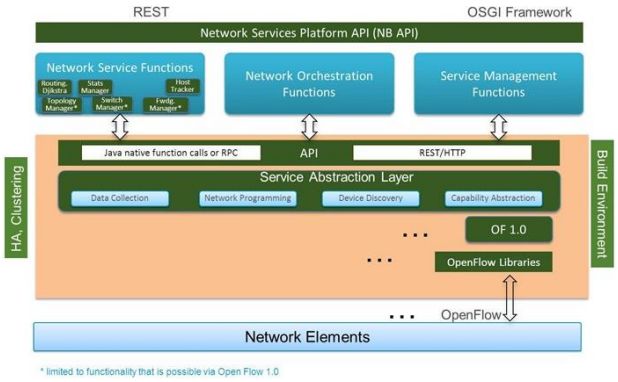

Fig.4. Architecture of OpenDaylight.

B. Ryu

Ryu is used for managing flow control to enable intelligent networking [29]. On the one hand, it is known as an open source framework for building SDN applications. On the other hand, it is also known as a NOS (Network Operating System) that not only supports the management of OpenFlow devices, but also allows users to work with devices through NETCONF [30], OVSDB (Open vSwitch Database Management Protocol), sFlow [31], NetFlow [32,33], VRRP [34] (Virtual Router Redundancy Protocol), and SNMP [16] (Simple Network Management Protocol).

Ryu has a design inspired by the construction of monolithic controllers, with a component-based architecture that offers greater scalability at the cost of efficiency, since it is implemented in the Python programming language. Ryu supports a wide variety of OpenFlow versions in the range [1.0, 1.4]. Fig. 5 shows the general architecture of Ryu.

Ryu applications are entities that implement multiple functions of Ryu based on components and libraries, such as the OpenFlow controller, the topology viewer, the L2 switch, the firewall, etc. The framework offers a useful and comprehensive group of these components, that can be modified and grouped, with the ability to communicate with other components. The creation of new components does not tie the component to a single language, since the communication is done with standard messages.

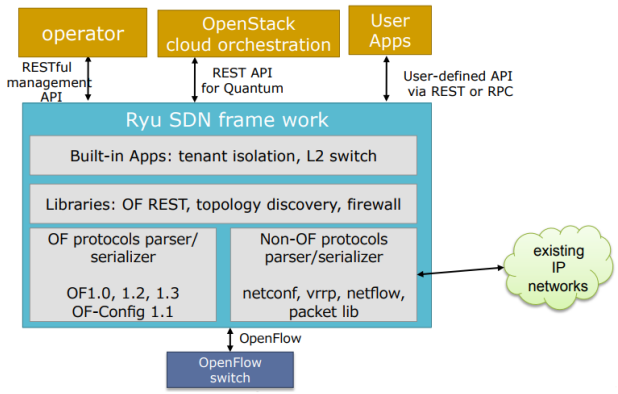

Fig.5. Architecture of Ryu.

C Floodlight

Floodlight is an OpenFlow controller written in Java under the Apache license. It is supported by a community of developers, including some Big Switch Networks engineers. Floodlight is a collection of applications built on top of the controller itself. The controller performs a series of common functionalities to monitor and control the OpenFlow network, while the applications are aimed for different activities to solve user requirements over the network.

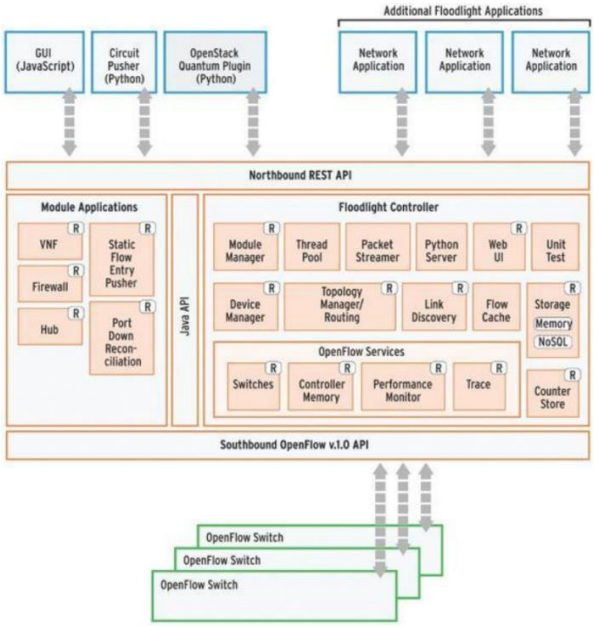

Fig.6. Architecture of Floodlight.

Since it is written in Java, it runs on top of a JVM (Java Virtual Machine). The Floodlight architecture is modular, as shown in Fig. 6. Like other controllers, Floodlight offers its services through southbound and northbound. In the northbound, the modules are exposed

through APIs that are executed by using a REST port. Any application can interact with the controller, by sending HTTP-REST commands, allowing the request of information or the invocation of services. It has numerous specific purpose modules that provide the basis for network functions, such as Topology Manager, Link Discovery, Device Manager, Performance Monitor, etc. These are necessary to offer services to applications and take actions in the underlying network.

On the other hand, in the southbound, Floodlight listens on the TCP port specific for the OpenFlow protocol (versions 1.0, 1.3, and 1.4), to initiate connections with SDN switches.

In the westbound, the Java API allows the development of modules in Java and their rapid interaction with the central controller. The modules are loaded by the system during the start of the controller and integrated into it, thus allowing communications with the basic network functions and providing a rapid response to network events, such as the appearance of new packets or flows.

D. OpenMUL

OpenMUL is an open source SDN controller, developed mainly in C by the OpenMUL Foundation, with a multi-threaded infrastructure that promises high performance, reliability, and capacity to host modular applications. It provides a wide variety of applications in its northbound interfaces and is capable of working with multiple southbound protocols such as OpenFlow 1.0/1.3/1.4, OVSDB, NETCONF [30], etc.

OpenMUL has a modular architecture with the aim of providing flexible functionalities for the orchestration of networks through an easy interface with multiple access points. It has multiple APIs for several languages and RESTful APIs for web applications in the northbound. The groups of functionalities can be observed in Fig. 7 [35].

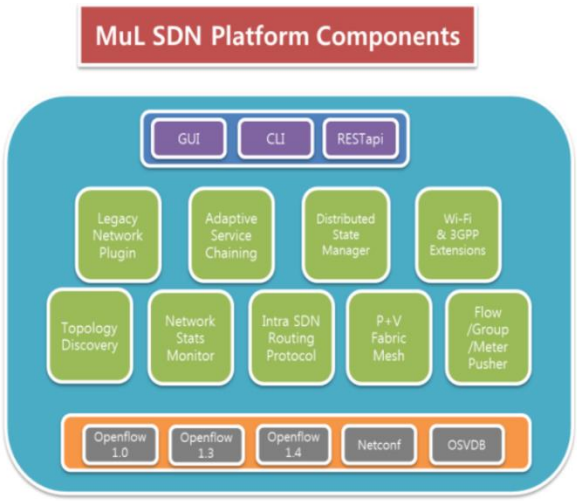

Fig.7. Architecture of OpenMUL.

VI. Proposed Benchmark Tool

We conducted a study of existing performance evaluation tools for OpenFlow controllers and found improvement opportunities. In the open-source world, the main tool is "Cbench." It works with OpenFlow version 1.0 and provides multiple parameters for achieving throughput and latency tests. Its operational model consists in the simulation of SDN switches that send requests (Packet-In) to the controller through the OpenFlow protocol, with the goal of measuring the throughput or the latency. Unfortunately, Cbench seems to be a dead project. It has not been updated for several years. Hence, it only supports an old version of OpenFlow (version 1.0). For these reasons, we decided to propose a more versatile tool, called OFC-BenchTool (OpenFlow Controller-Benchmark).

OFC-BenchTool supports OpenFlow version 1.3, and its architecture is based on a distributed master-slave model, with message passing. Message passing allows parallel processes to run synchronized tests against the same controller, creating a large number of requests, thus increasing the stress capacity during tests. Additionally, with the goal of reporting hardware characteristics during the performance assessment, we included the usage of SNMP [16]. Through SNMP, we collect information to strengthen the evaluation process. Our tool also has a module to facilitate the analysis of information, incorporating the generation of graphics to represent the results obtained. The architecture of OFC-BenchTool is represented in Fig. 8.

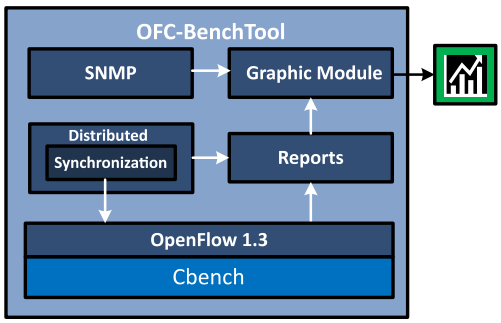

Fig.8. Architecture of OFC-BenchTool.

We chose the C programming language to develop OFC-BenchTool, due to its robustness, numerous libraries, and extensive documentation. In order to facilitate specific functionalities, we used some development libraries: gnuplot_i for the graphical module in charge of displaying a graphical representation of the results; libpcap for the capture and handling of network packets; and Net-SNMP for the process of SNMP packets.

OFC-BenchTool permits three types of tests: (1) latency, (2) throughout in isolated mode, and (3) throughput in distributed mode. The three tests are based on the client/server model, where a virtual switch is a

client and the controller is the server. In the latency test, a unique slave process of the benchmark is involved which creates one virtual switch. Before the test, the benchmark obtains a timestamp. Then, the virtual switch sends a Packet-In message to the controller for a new flow, and waits for the Packet-Out response. As soon as the switch receives the answer, it sends a new Packet-In message to the controller for a new flow, and waits for the answer. This process is repeated n times. At the end of the n exchanges (Packet-In/Packet-Out), a second timestamp is taken. The difference between the two timestamps is divided by n, to get the latency.

In the throughput test in isolated mode, there is only one slave process of the proposed benchmark to assess the controller. This slave process creates several virtual switches (a parameter specified by the users). Basically, each virtual switch tries to overwhelm the controller by sending Packet-In messages for new flows, as fast as they can, and counts the number of Packet-Out responses received. This process is repeated for 50 seconds. The total number of Packet-Out messages received in one second is reported as the throughput and is expressed as flow/second, that is, the benchmark reports the number of new flows that were successfully created.

The throughput test in distributed mode is similar to the throughput test in isolated mode, but several slave processes of the benchmark are created to assess the SDN controller (the idea is to create more stress over the controller). Through parameters, users can specify the number of slave nodes, the number of virtual switches to be created in each slave node, and the IP address of the master. Basically, and similarly to the isolated mode, each virtual switch tries to overwhelm the controller with Packet-In messages. The master node is in charge of maintaining the synchronization between the slave nodes, performing SNMP requests, and handling the reports generated by slaves. To keep the slaves synchronized, a topology connection in star between the master and the slaves is established. Initially, the master waits for the slaves to be operational, and when all the slaves are connected, the master sends a start command to each slave. At the end of the test, the slaves send their reports to the master. The master receives the reports and formats them adequately for the graphic module, where they can be presented in the form of lines, points, and bars. Simultaneously to the synchronization of the slaves during the stress tests, the master process sends SNMP queries to the SDN controller, to monitor its state.

VII. Results

This section describes the tests that were performed with OFC-BenchTool, in order to evaluate the fulfillment of the proposed objectives.

For all the tests done in this paper, 16 simulated switches were created in each slave node. Also, we performed 50 cycles, where 1 cycle is equivalent to 1 second.

A. Throughput Test in Isolated Mode

In this mode, we used the throughput test with 1 slave node that created 16 virtual switches (Virtual Switch 0, ..., Virtual Switch 15). Fig. 9 shows the throughput recorded in each virtual switch, per second, when using OpenMUL. The x-axis represents the time, in second, where 0 sec is the beginning of the experiment. The y-axis has the throughput, that is, the number of flows established by second. As we can observe, there is no preference of responses for a specific switch, that is, the treatment of OpenMUL is uniform among the 16 virtual switches. Also, it is worth mentioning that the throughput is low at the beginning of the experiment, and then tends to fluctuate around 275 flow/sec in each virtual switch.

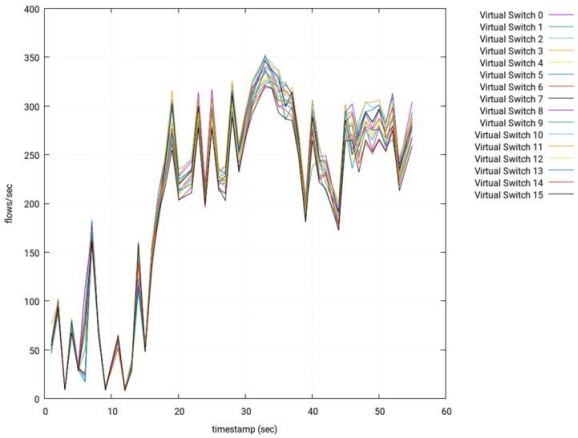

Fig.9. Throughput per Switch – 1 Slave Node – OpenMUL.

Fig. 10 depicts the memory and CPU consumption in the computer running the SDN controller (OpenMUL), during this test. Both parameters are expressed as percentages of the total memory and CPU.

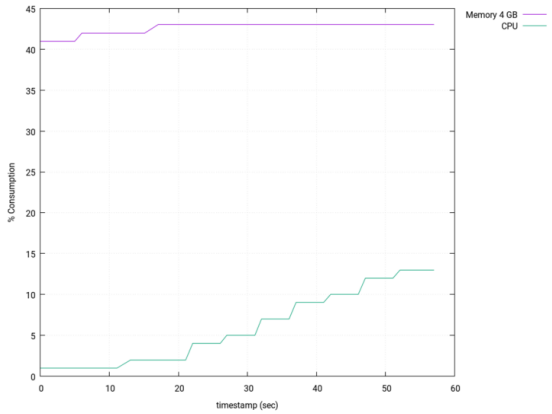

Fig.10. Memory and CPU Consumption – 1 Slave Node – OpenMUL.

Fig. 11 shows the usage of the network interface of the SDN controller (OpenMUL), during this test (I/O traffic). This usage is represented as the accumulated amount of bytes sent and received by the controller.

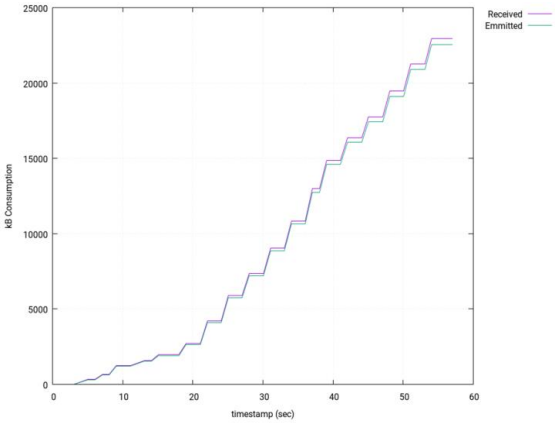

Fig.11. Input/Output Network Traffic through the Interface of the SDN Controller – 1 Slave Node – OpenMUL.

B. Throughput Test in Distributed Mode

For the distributed mode, we made tests with 4 nodes (1 master and 3 slaves). The master was in charge of the synchronization of the slave nodes, the measurement of parameters through SNMP, and the recollection of the results from the slaves. The slave nodes were responsible for carrying out the tests and sending their final results to the master. Each slave node created 16 virtual switches (Virtual Switch 0, ..., Virtual Switch 15), with a total of 48 virtual switches ($3 \times 16 = 48$).

Figs. 12, 13, and 14 show the throughput recorded in each virtual switch, per second, when using OpenMUL. The x-axis represents the time, in second, where 0 sec is the beginning of the experiment. The y-axis has the throughput, that is, the number of flows established by second.

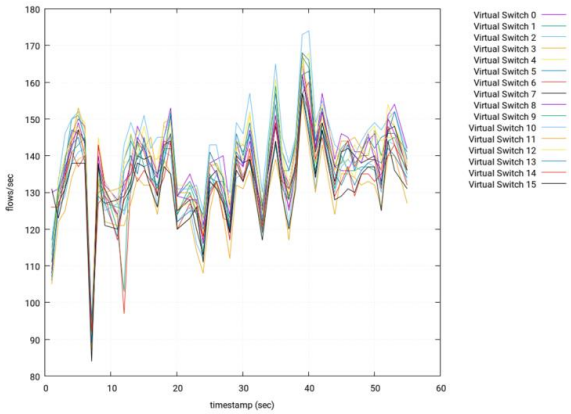

Fig.12. Throughput per Switch – 3 Slave Nodes – OpenMUL – Slave 1.

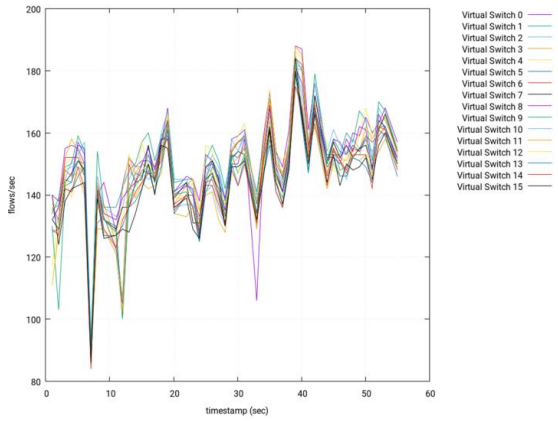

Fig.13. Throughput per Switch – 3 Slave Nodes – OpenMUL – Slave 2.

In contrast to Fig. 9 (throughput in isolated mode) where the throughput tends to fluctuates around 275 flow/sec for each virtual switch, the throughput in distributed mode is lower (around 150 flow/sec per virtual switch). This is due to the major stress suffered by the controller, since it serves 16 switches in the isolated mode, vs 48 switches in the distributed mode.

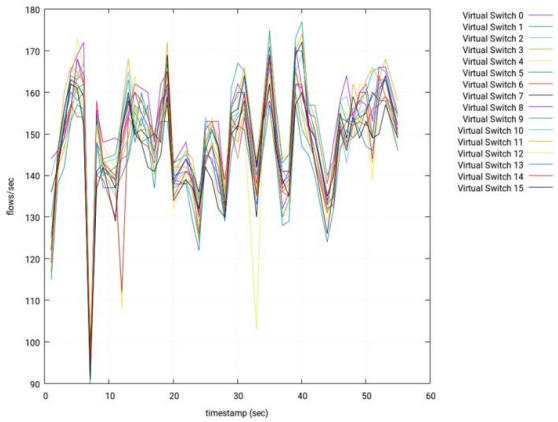

Fig.14. Throughput per Switch – 3 Slave Nodes – OpenMUL – Slave 3.

Fig. 15 depicts the memory and CPU consumption in the computer running the SDN controller (OpenMUL), during this test. Both parameters are expressed as percentages of the total memory and CPU. When compared with the Isolated Mode (see Fig. 10), we can conclude that the memory consumption is similar in both experiments. However, the CPU processing in much higher in the Distributed Mode (30% vs 12%, at the end of the experiment), and it is due to the major stress suffered by the controller.

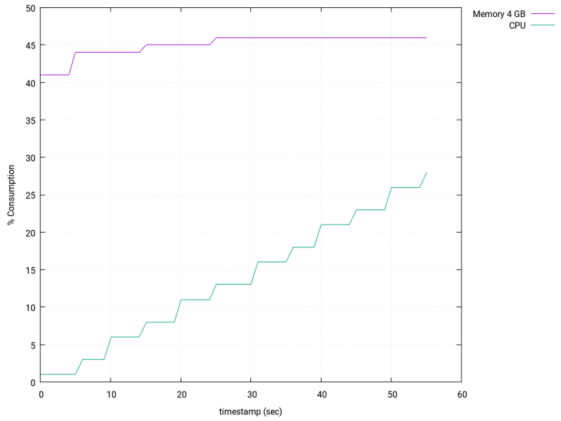

Fig.15. Memory and CPU Consumption – 3 Slave Nodes – OpenMUL.

Fig. 16 shows the usage of the network interface of the SDN controller (OpenMUL), during the test (I/O traffic). This usage is represented as the accumulated amount of bytes sent and received by the controller. When compared with the Isolated Mode (see Fig. 11), we can conclude that the network usage in much higher in the Distributed Mode (50000 kB vs 22000 kB, at the end of the experiment), and it is due to the major stress suffered by the controller.

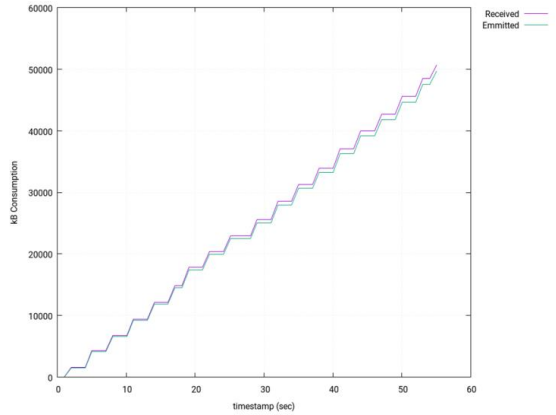

Fig.16. Input/Output Network Traffic through the Interface of the SDN Controller – 3 Slave Nodes – OpenMUL.

C. Throughput Test for Different Controllers

To compare different controller, the test of throughput in Distributed Mode (see Section VII.B) was repeated with other SDN controllers (OpenDaylight, Floodlight, and Ryu). In order to make a fair comparison of these controllers, the installation of each controller was performed in the same environment, with the same characteristics. That is, a computer with an Intel i7 CPU and 4 GB of RAM, that was running Ubuntu 17.04 as the operating system.

1) OpenMUL: The tests that were performed on the OpenMUL controller were successful, the connection was achieved with the controller in the default port (port 6653), the OpenFlow messages were answered correctly, and at an expected rate. The results of these tests are shown from Fig. 12 to Fig. 16.

2) Floodlight: Similarly to OpenMUL, the tests made with the Floodlight controller were successful, and the results are shown from Fig. 17 to Fig. 21. As can be observed in our experiments, the throughput of OpenMUL is a little over the throughput of Floodlight.

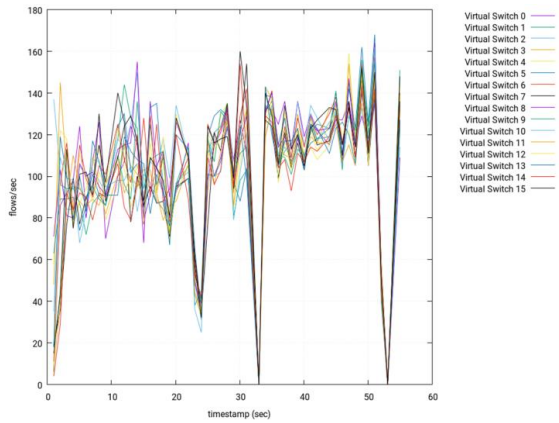

Fig.17. Throughput per Switch – 3 Slave Nodes – Floodlight – Slave 1.

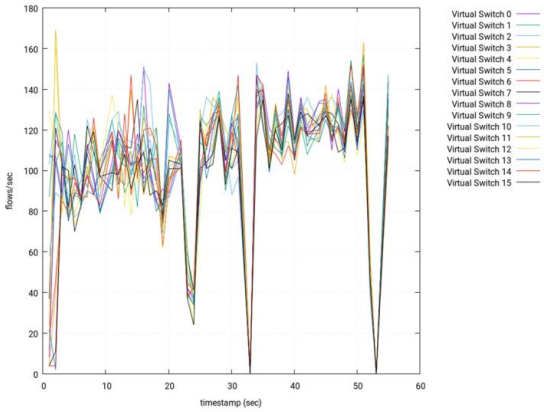

Fig.18. Throughput per Switch – 3 Slave Nodes – Floodlight – Slave 2.

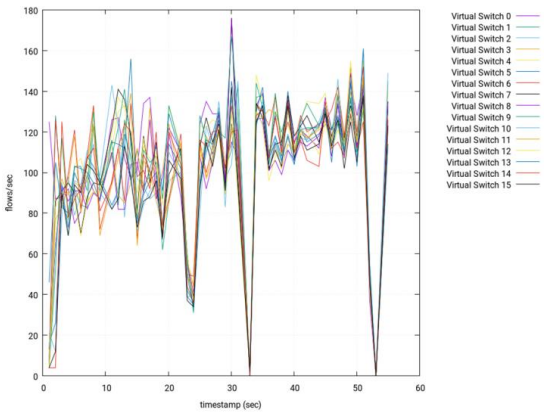

Fig.19. Throughput per Switch – 3 Slave Nodes – Floodlight – Slave 3.

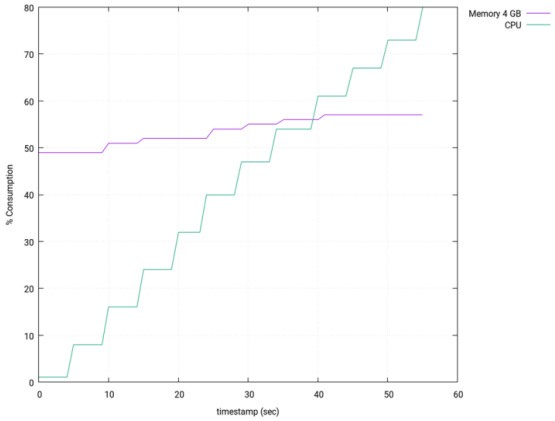

Fig.20. Memory and CPU Consumption – 3 Slave Nodes – Floodlight.

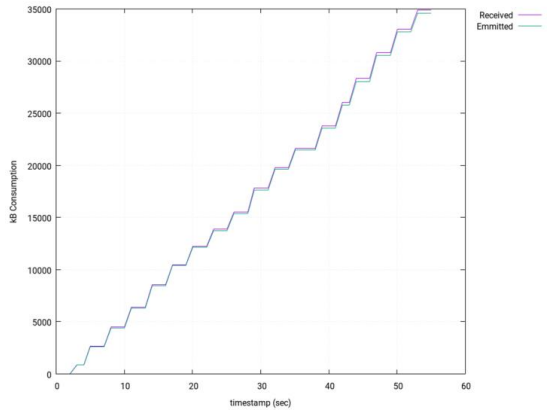

Fig.21. Input/Output Network Traffic through the Interface of the SDN Controller – 3 Slave Nodes – Floodlight.

1) Ryu: For the Ryu controller, the tests done showed a much lower performance than the other controllers, possibly due to the fact that Ryu is completely written in Python, an interpreted language, which is much slower than C or Java. The results are shown from Fig. 22 to Fig.26.

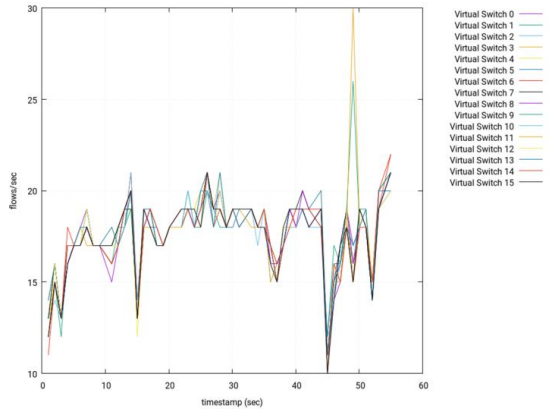

Fig.22. Throughput per Switch – 3 Slave Nodes – Ryu – Node 1.

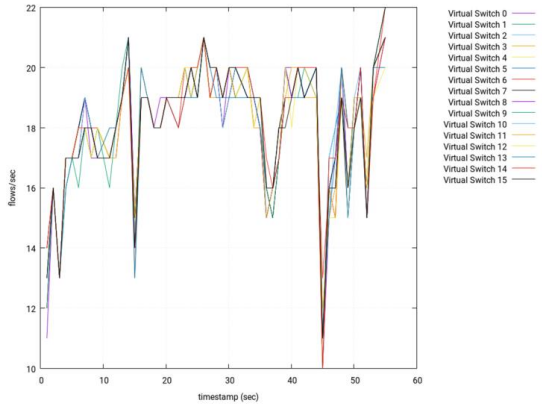

Fig.23. Throughput per Switch – 3 Slave Nodes – Ryu – Node 2.

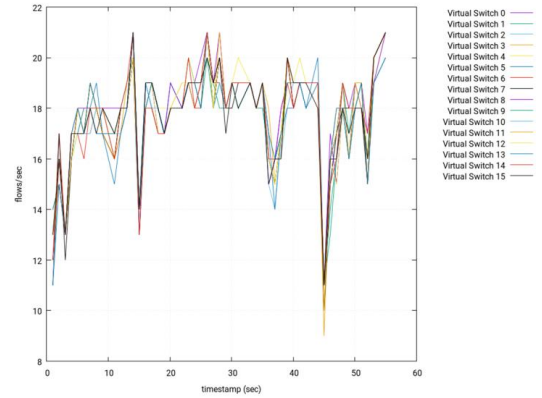

Fig.24. Throughput per Switch – 3 Slave Nodes – Ryu – Node 3.

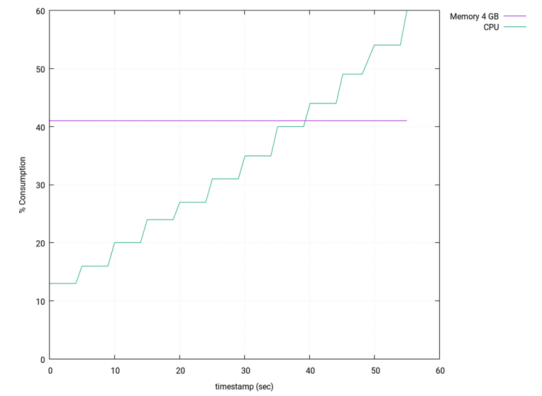

Fig.25. Memory and CPU Consumption – 3 Slave Nodes – Ryu.

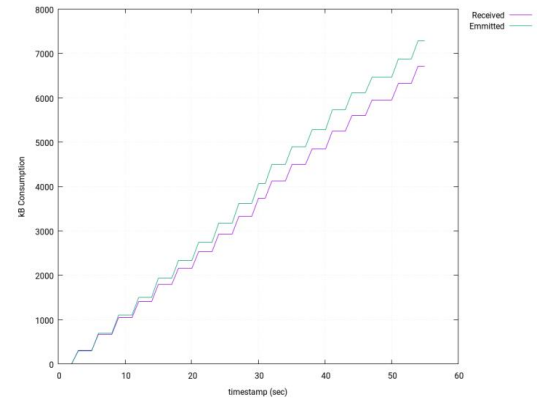

Fig.26. Input/Output Network Traffic through the Interface of the SDN Controller – 3 Slave Nodes – Ryu.

2) OpenDaylight: We faced problems with the OpenDaylight controller during our tests. It worked well during the first iterations, before reaching a point where it looks like that it does not process any petition anymore. Despite our effort, we could not find any reason for the malfunction of the controller. Hence, the test was repeated with a smaller number of cycles. That is, we used 16 cycles (instead of 50 cycles) for OpenDaylight and the obtained results are shown from Fig. 27 to Fig. 31.

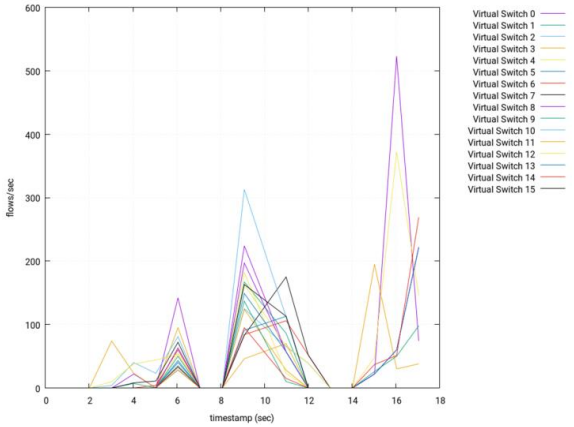

Fig.27. Throughput per Switch – 3 Slave Nodes – ODL – Node 1.

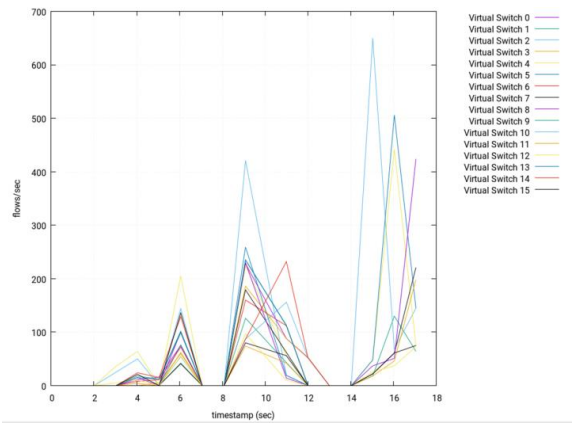

Fig.28. Throughput per Switch – 3 Slave Nodes – ODL – Node 2.

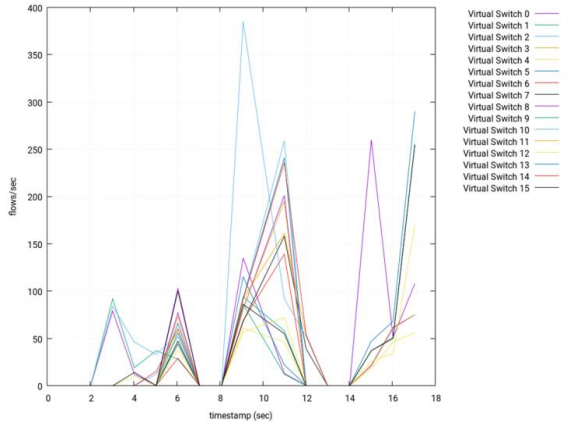

Fig.29. Throughput per Switch – 3 Slave Nodes – ODL – Node 3.

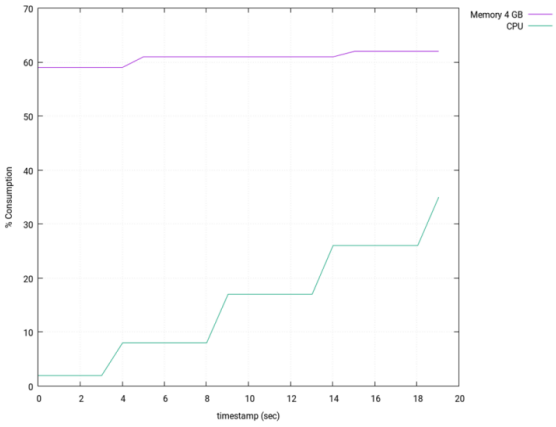

Fig.30. Memory and CPU Consumption – 3 Slave Nodes – ODL.

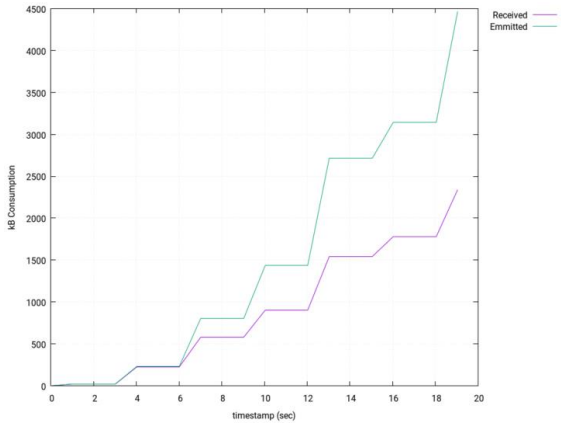

Fig.31. Input/Output Network Traffic through the Interface of the SDN Controller – 3 Slave Nodes – ODL.

D. Comparison with Other Performance Tools

Tests with similar tools were carried out with difficulties, since there are only a few related tools that are open-source. The stronger open-source option is Cbench [7]. The throughput test of Cbench was performed against an OpenMUL controller, and the obtained results were compared with the ones that we got with our benchmarking tool.

It is very important to note that there is a significant difference in both tests, which is the OpenFlow version used. In the test performed with Cbench, the OpenFlow version is 1.0, since it is the only supported version by this tool. In the test performed to OpenMUL with our benchmarking tool, the version of the protocol is 1.3, which has significant changes. The results of the test performed with Cbench are shown in Fig. 32. We can see that this test has similar results to the ones obtained in the isolated mode (see Fig. 9), confirming the correct operation of our benchmarking tool.

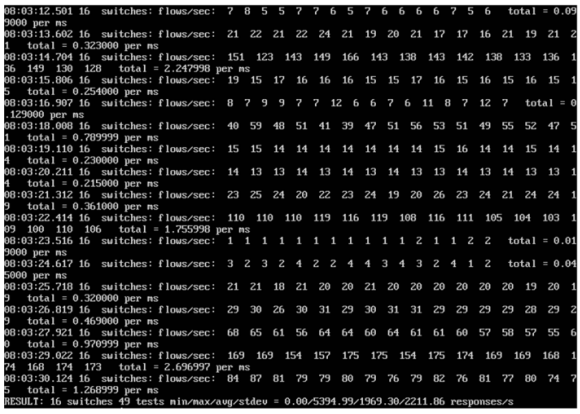

Fig.32. Results of Cbench when Evaluating OpenMUL.

VIII. Conclusions and Future Work

In this paper, we proposed a performance evaluation tool to assess SDN controllers. Our tool was developed in the C programming language, with libraries that facilitate the graphical representation of results and the handling/ construction of packets for OpenFlow and SNMP. The tool has two modes (isolated and distributed). In the isolated mode, a single test node is used, that emulates virtual SDN switches. In the distributed mode, several distributed slave nodes are created and synchronized by a master node, to achieve a higher level of stress against the SDN controller. To validate the tool, we carried out tests with several popular SDN controllers.

With our tool, users can assess the throughput and latency of SDN controllers, under different levels of stress. Additionally, users can get information about memory and CPU utilization of the controller during the test. Our tool can significantly help network administrators in the selection of an SDN controller, since they can study the behavior of the controllers when they reach their limits.

As future work, we plan to develop new functionalities and modules to support:

- newest versions of the OpenFlow protocol
- graphic interfaces that improve the usability and user experience
- error management based on logs
- tests in more complex networks with fault-tolerant architecture.

References

[1] P. Goransson, C. Black, and T. Culver. Software Defined Networks: A Comprehensive Approach, Second Edition. November 2016.

[2] J. Doherty. SDN and NFV Simplified: A Visual Guide to Understanding Software Defined Networks and Network Function Virtualization. Addison-Wesley Professional; First Edition, March 2016.

[3] Open Networking Foundation. OpenFlow Switch Specification. Version 1.5.1 (Protocol Version 0x06). March 2015.

[4] O. Coker and S. Azodolmolky. Software Defined Networking with OpenFlow, Packt Publishing, Second Edition. October 2017.

[5] G. Blokdyk, OpenFlow: The Definitive Handbook. CreateSpace Independent Publishing Platform. October 2017.

[6] M. Basheer Al-Somaidai and E. Bassam Yahya, Survey of Software Components to Emulate OpenFlow Protocol as an SDN Implementation, American Journal of Software Engineering and Applications, Vol. 3, No. 6, pp. 74-82.

[7] A. Hassan and S. Ahmed, "Performance Comparison of the State of the Art OpenFlow Controllers". Halmstad, September 2014.

[8] M. Jarschel, F. Lehrieder, Z. Magyari, and R. Pries, A Flexible OpenFlow-Controller Benchmark, in Proceedings of the 2012 European Workshop on Software Defined Networking (EWSDN 2012), Darmstadt, Germany, October 2012.

[9] M. Darianian, C. Williamson, and I. Haque, Experimental Evaluation of Two OpenFlow Controllers, in Proceedings of the 2017 IEEE 25th International Conference on Network Protocols (ICNP), Toronto, ON, Canada, November 2017.

[10] B. Xiong, K. Yang, J. Zhao, W. Li, and K. Li. Performance Evaluation of OpenFlow-based Software-defined Networks based on Queueing Model, Computer Networks, Vol. 102, pp. 172-185, 2016.

[11] I. Bholebawa and U. Dalal, Performance Analysis of SDN/OpenFlow Controllers: POX Versus Floodlight, Wireless Personal Communications: An International Journal, Vol. 98, No. 2, pp. 1679-1699, January 2018.

[12] B. Lantz, B. Heller, and N. McKeown. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In Proceedings of the 9th ACM Workshop on Hot Topics in Networks, Monterey, CA, USA. October 2010.

[13] B. Lantz, B. O'Connor, A Mininet-based Virtual Testbed for Distributed SDN Development. In Proceedings of SIGCOMM 2015, London, UK. August 2015.

[14] S. Rowshanrad, V. Abdi, and M. Keshtgari, Performance Evaluation of SDN Controllers: Floodlight and OpenDaylight, IIUM Engineering Journal, Vol. 17, No. 2, 2016.

[15] S.-Y. Wang, H.-W. Chiu, and C.-L. Chou, Comparisons of SDN OpenFlow Controllers over EstiNet: Ryu vs. NOX, in Proceedings of the Fourteenth International Conference on Networks (ICN 2015), Barcelona, Spain, April 2015.

[16] G. Blokdyk, SNMP Simple Network Management Protocol: Amazing Projects from Scratch, CreateSpace Independent Publishing Platform, October 2017.

[17] N. Feamster, J. Rexford, and E. Zegura, "The Road to SDN: An Intellectual History". ACM SIGCOMM Computer Communication Review, vol. 44, no. 2, pp. 87-98, April 2014.

[18] N. Figuerola, "SDN – Redes Definidas por Software". October 2013. https://articulosit.files.wordpress.com/2013/10/sdn.pdf.

[19] I. Gavilán, "Fundamentos de SDN (Software Defined Networking)". August 2013. http://es.slideshare.net/ igrgavilan/20130805-introduccion-sdn.

[20] S. Azodolmolky, "Software Defined Networking with OpenFlow", Packt Publishing Ltd., October 2013.

[21] P. Göransson and C. Black, "Software Defined Networks: A Comprehensive Approach", Morgan Kaufmann, May 2014.

[22] M. Palacin Mateo, "OpenFlow Switching Performance". July 2009.

[23] T. D. Nadeau and K. Gray, "SDN: Software Defined Networks". O'Reilly, August 2013.
[24] A. Nierbeck, J. Goodyear, J. Edstrom, and H. Kesler, Apache Karaf Cookbook, Packt Publishing, August 2014.
[25] J. Edstrom, J. Goodyear, and H. Kesler, Learning Apache Karaf, Packt Publishing, October 2013.
[26] G. Blokdyk, OSGi: Upgrader's Guide, CreateSpace Independent Publishing Platform, November 2017.
[27] H. Cummins and T. Ward, Enterprise OSGi in Action: With Examples using Apache Aries, Manning Publications, 1st edition, April 2013.
[28] OpenDaylight, "Wiki OpenDaylight", https://wiki.opendaylight.org/view/OpenDaylight_Controller:Architectural_Framework.
[29] K. Ohmura, "OpenStack/Quantum SDN-based Network Virtualization". May 2013. http://osrg.github.io/ryu/slides/LinuxConJapan2013.pdf.
[30] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, Network Configuration Protocol (NETCONF), RFC 6241, June 2011.
[31] P. Phaal, S. Panchen, and N. McKee, InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks, RFC 3176, September 2001.
[32] O. Santos, Network Security with NetFlow and IPFIX: Big Data Analytics for Information Security, Cisco Press, 1st edition, October 2015.
[33] M. Allen Patterson, D. Robb, and A. Akhter, Unleashing the Power of NetFlow and IPFIX, Amazon Digital Services LLC, September 2013.
[34] S. Nadas, Virtual Router Redundancy Protocol (VRRP) Version 3 for IPv4 and IPv6, RFC 5798, March 2010.
[35] N. Malik and D. Saikia, "An Introduction to OpenMUL SDN Suite". September 2014.

Authors' Profiles

Eric Gamess received an M.S. in Industrial Computing from the National Institute of Applied Sciences of Toulouse (INSA de Toulouse), France, in 1989, and a Ph.D. in Computer Science from the Central University of Venezuela, Venezuela, in 2000. He is currently working as a professor at Jacksonville State University, Jacksonville, Alabama, USA. Previously, he worked as a professor at the Central University of Venezuela, Venezuela, University of Puerto Rico, Puerto Rico, and "Universidad del Valle," Colombia. His research interests include Vehicular Adhoc Networks, Network Performance Evaluation, IPv6, and Network Protocol Specifications. He is a member of the Venezuelan Society of Computing and has been in the organization committee and the technical program committee of several national and international conferences.

Daniel Tovar received a B.S. in Computer Science from the Central University of Venezuela, Venezuela, in 2017. He is currently working at CGTS Corp, Caracas, Venezuela, as a network and system administrator. His research interest includes: Network Simulations, Network Performance Evaluation, Software Defined Networks, and Web Development.

Alberto Cavadia received a B.S. in Computer Science from the Central University of Venezuela, Venezuela, in 2017. His research interest includes: Computer Networking, High Performance Computing, and Web Development.

How to cite this paper: Eric Gamess, Daniel Tovar, Alberto Cavadia, "Design and Implementation of a Benchmarking Tool for OpenFlow Controllers", International Journal of Information Technology and Computer Science(IJITCS), Vol.10, No.11, pp.1-13, 2018. DOI: 10.5815/ijitcs.2018.11.01