Experimental Analysis in SDN Open Source Environment

Kuljaree Tantayakul, Riadh Dhaou, Béatrice Paillassa, Wasimon Panichpattanakul

Experimental Analysis in SDN Open Source Environment

Kuljaree Tantayakul*, Riadh Dhaou*, Beatrice Paillassa*,Wasimon Panichpattanakul†
*IRIT-ENSEEIHT, University of Toulouse, France
†Department of Computer Engineering, Prince of Songkla University, Phuket, Thailand
kuljaree.tantayakul@etu.enseeiht.fr, {Riadh.Dhaou, Beatrice.Paillassa}@enseeiht.fr, wasimon@mail.coe.phuket.psu.ac.th

***Abstract*—The paper focuses on the free products used to setup an Software-Defined Network by emulation and real experimentation. Because SDN networking is a challenging research area, objective of this paper is to provide assistance to researchers wishing to evaluate their SDN work in an experimental setting. In this paper, two collections of open source OpenFlow switches; Open vSwitch (OVS) and ofsoftswitch13 (CPqD), which support new version of OpenWRT firmware and the testing switches in our laboratory, have been emulated and implemented in real networks. The performances of OVS and CPqD in terms of UDP throughput, TCP throughput and percentage of the packet loss have been analyzed. The results indicate the consistency of the results obtained by emulation and experimentation and highlight the blocking parameter of the switch processing capacity.**

***Index Terms*—Software-defined Network, OVS, CPqD.**

I. Introduction

In legacy networks, each network device must be controlled and managed individually. In addition, devices of different vendors usually have different firmwares and the forwarding and control planes are coupled within one box. Thus, it is inflexible and complicate to manage the network. Since the Software-Defined Network (SDN) [1][2] was proposed, a new approach to computer networking has been introduced to allow the network administrators to easily configure, update and monitor the different network devices from different manufacturers through a software application. Moreover, it is more flexible to extend network functions in future.

Thus, it is not surprising that many simulation and emulation tools, supporting SDN, have been published in both commercial and open source tools e.g. Network Simulator 3 (NS3) [3][4], EstiNet [5], Mininet [6], and OpenNet [7]. These tools can reduce the cost of designing, testing and optimizing solutions before implementating into the real networks. Moreover, the interested data can be easily collected. The comparison of these simulators [7] can be shown in TABLE I. The main challenge of evaluation tools is the performance assessment and OpenNet emulator is the best tool because it links Mininet and NS3 together in order to use the Mininet advantage in term of compatibility and the NS3 ability in term of the wireless/mobility modeling. However, since this paper made its experimental network based on wired network, Mininet emulator has been selected as a sufficient tool.

To test the performance of SDN by deployment in real networks, whether there are various commercial products and testing platforms supporting the SDN such as GEANT OpenFlow Facility (GOFF) [8] from GEANT network, these products and platforms are expensive and not easily accessible. Thus, the use of open source products is a right solution for many researchers. However, since there are many choices of open source switches; it motivates us to make some experiments and measure the performance of open source OpenFlow switches. In this paper, we focused on two collection of open source OpenFlow switches; Open vSwitch (OVS) and ofsoftswitch13 (CPqD). The experiments and analysis of this paper will be a guidance for the network administrators and researchers, who are interested in deploying SDN to their networks.

TABLE I
Comparison between Existing Simulators/Emulators [7]

Simulators	Mininet	NS3	EstiNet	OpenNet
OpenFlow Version	1.3.1	0.8.9	1.3.1	1.3.1
Simulation (S)/Emulation (E)	E	S/E	S/E	S/E
Wireless Functionality	No	No Scan	Yes	Yes
Controller Compatibility	Yes	No	Yes	Yes
Extendibility	Yes	Yes	No	Yes

The outline of the paper is as follows: the concept of SDN, controller and software OpenFlow switch are presented in Section II. Then, the results of performance analysis on experimental topology are shown in Section III and Section IV. Finally, the conclusion and future works will be presented in Section V.

II. Related Works

A. Software-defined Networking (SDN)

The concept of SDN is based on a centralized intelligence by dividing the operation into two parts: the control plane and the data plane. The operation methods require some protocols such as NETCONF, Border Gateway Protocol (BGP), Open vSwitch Database Management Protocol (OVSDB), MPLS Transport Profile (MPLS-TP), and the OpenFlow protocol. The key component of SDN architecture is the controller, which operates as a brain since it coordinates and manages all network devices in the SDN.

OpenFlow protocol [9] is the first standard protocol that was defined by Open Network Foundation (ONF). It is a Layer 2 protocol, which provides the communication between the centralized controller and the network devices in SDN

architecture. It works on top of the Transmission Control Protocol (TCP) [10] and was released in many versions: OpenFlow1.5.1 [11] (march 2015) is the current version.

B. SDN Controllers

The controller is one of the main components in the SDN network. There are many different available SDN controllers such as POX [12], RYU [13], Pyretic [14], Trema [15], FloodLight [16], ONOS [17] and OpenDaylight [18]. In this paper, RYU controller has been chosen because it is widely used, provides well document and defines API for creating various SDN application [19] [20]. It also is very fast python based OpenFlow controller [21] as compare the performance of application with POX and Pyretic controller.

C. Software OpenFlow Switch

There are many software switches [22][23] such as Open vSwitch [24], CPqD OpenFlow switch [25], LINC switch [26] and Pantou [27]. These different switches can be installed in the network. However, from our review LINC supports OpenFlow 1.3 but cannot be implemented on a small Linux router such as OpenWRT while Pantou supports only OpenFlow 1.0.

1) Open vSwitch (OVS): is an open source software switch implemented on Linux distribution. It is compatible with OpenFlow1.0 and OpenFlow1.3. OVS can be used with flexible controller in User-Space and perform fast Datapath in Kernel.

2) CPqD software Switch (CPqD): was originally supported by Ericsson Innovation Center in Brazil in a partnership with CPqD. CPqD switch is the first open source software kit, which supports OpenFlow1.2 and the latest version supports OpenFlow 1.3.

3) LINC Switch: is a pure OpenFlow software written in Erlang, which supports for OpenFlow 1.2 to 1.4. It is not the most efficient one but it gives a lot of flexibility and allows users to quickly develop and test the new OpenFlow features.

4) Pantou: is an OpenFlow 1.0 implementation based on the BackFire OpenWRT release (Linux 2.6.32). It turns a commercial wireless router/Access Point to an OpenFlow enabled switch by implementing an OpenFlow on top of OpenWRT. The performance of Pantou on TP-Link WR1043ND is declared that it can competently send out about 43 Mbps for a single UDP flow [27].

III. Experimental Topology

An experimental network has been setup as illustrated in Fig. 1 in order to compare the performance of OVS and CPqD switches. In this paper, the experiments have been separated into 2 parts: emulation part and real experiment part. The parameters of each case are set as shown in TABLE II.

For emulation part, Mininet has been used to generate a topology consisting of one OpenFlow switch, three hosts, one Linux Router and one RYU controller. The OpenFlow switch was set to be OVS switch and changed to be CPqD switch later. They have been configured to support OpenFlow 1.3. The RYU controller runs script for OpenFlow 1.3.

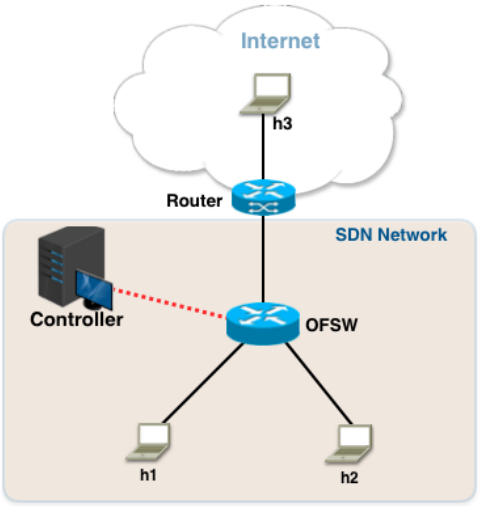

Fig. 1. SDN Network Testbed

TABLE II
Experimental Parameters

Parameter	Setting
Emulation Tool	Mininet 2.1.0p2
Link Speed	100Mbps
OpenFlow	vSwitch 2.4.0/ofsoftswitch13
Controller	RYU 3.18
OpenFlow	v1.3.0
Testing Tool	Iperf v2.0.5
UDP Datagram	1470 Bytes
TCP Window Size	85 KBytes

For our real experimental network, the device specification is shown in TABLE III. TP-Link WR1043NDv3 is set as a Router while TP-Link WR1043NDv2 is set as an OpenFlow switch. We compiled a new OpenWRT firmware and added the OpenFlow package version 1.3 from trunk source. This firmware is a CPqD OFSoftswitch 1.3 [28][29] that is called CPqD switch in this paper. The memory usage of OpenWRT CPqD switch is 308 KBytes, about 7% of overlay. The OpenWRT OVS switch is installed on TP-Link WR1043NDv2 [30] by installing OpenWRT firmware [31] first and using sysupgrade firmware to specify a version that supports Open vSwitch package. The Open vSwitch package can be installed by using opkg command. The memory usage of OpenWRT OVS switch is 2.5 MBytes about 54% of overlay. Both of the OpenFlow switches were configured to support OpenFlow 1.3.

RYU controller is used to manage the routing path of OpenFlow switch while Iperf [32] tool have been used to generate UDP and TCP traffics in order to measure thier performance. Wireshark [33] is also used to capture the traffics and tshark [34] is used to classify the data for performance analysis.

IV. Performance Analysis

For experimental results and analyses, three factors have been considered. There are the bandwidth capability, the

TABLE III
HARDWARE SPECIFICATION IN REAL EXPERIMENT

Device Name	Hardware	Firmware/OS
h1	Lenovo ideapad 100	Ubuntu 14.04.4 LTS
h2	Acer ONE D270	Ubuntu 12.04.4 LTS
h3	MacBook Pro	OS X EI Capitan
Controller	DELL Precission T5610	Ubuntu 14.04.4 LTS
OVS Switch	TP-Link WR 1043 ND v3	Linux OpenWrt 4.1.23
CPqD Switch	TP-Link WR 1043 ND v2	Linux OpenWrt 4.1.11
R1	TP-Link WR 1043 ND v3	TP-Link Firmware v.3.16.9

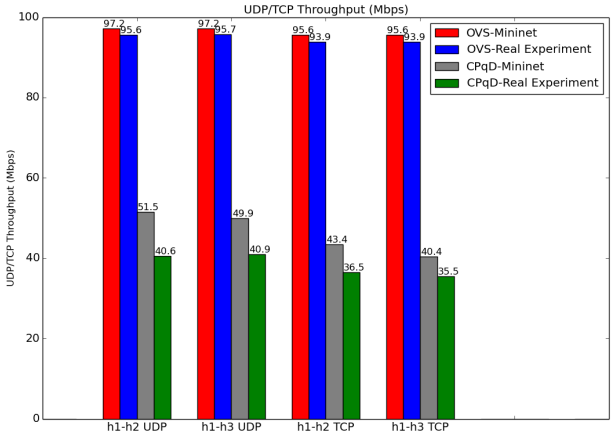

Fig. 2. UDP/TCP Throughput

percentage of packet loss and the service time of one packet in each OpenFlow switch. *1) Bandwidth Capability*

Iperf was used to generate UDP and TCP traffic for one hour. Every parameter was set as indicated in TABLE II. The UDP and TCP bandwidth capabilities have been measured in 2 scenarios separated by traffic types: internal and external traffics. Internal traffic is a traffic (or a connection) between two hosts inside SDN network, which is the h1 and h2 connection in Fig. 1. The external traffic is a traffic of a host in SDN network, which connects to another host outside the SDN network e.g. h1 and h3 connection or h2 and h3 connection as shown in Fig. 1. In this experiment, no traffic matrix has been used because the main objective is to measure the bandwidth capability of links in each OpenFlow switch. A single UDP and TCP traffic have been generated 10 times. The results of UDP/TCP throughput can be illustrated in Fig. 2. The trend of the results of each switch in Mininet is similar to those of each switch in real experiment.

Considering the UDP/TCP throughput of OVS switch, the results are close to the link bandwidth capacity (100Mbps). The UDP throughput in Mininet is higher than that of the real experiment about 1.65% and about 1.78% for TCP throughput.

For CPqD switch, the UDP/TCP throughput in is about 50 Mbps in Mininet, and it is 20% higher than that of the real experiment. The results show that OVS switch has better performance than CPqD switch.

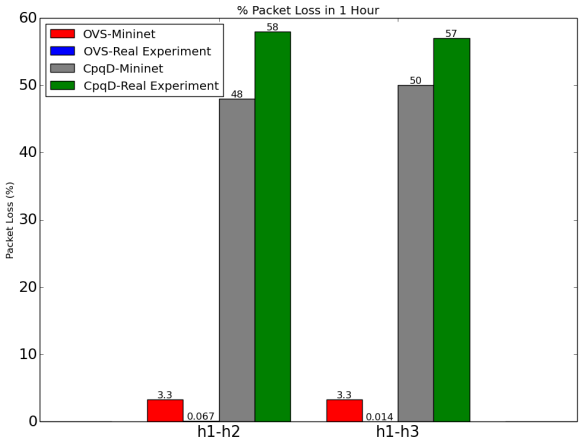

Fig. 3. Percentage of Packet Loss in 1 Hour

2) Packet Loss

Percentages of the packet loss in 1 hour of internal and external traffics are illustrated in Fig. 3. Considering OVS switch, the percentages of the packet loss in Mininet are about 3.3%, which are higher than those of the real experiment. In contrast, considering the percentages of the CPqD switch, the packet loss in the real experiments are about 57-58%, which are higher than those in Mininet. The percentage of packet loss of CPqD switch is very high because the maximum bandwidth capacity of CPqD is only about 50 Mbps referring to the results in previous sub-session(Bandwidth Capability). Thus, it cannot handle the UDP traffic, which has its data rate equal to 100 Mbps.

From Fig. 2 and Fig. 3, Mininet indicates higher throughput than experiment, meanwhile loss is higher. Thus, the number of packet that is transmited by the Mininet host emulation entity is higher than the real host, that is not a surprising result. Besides, during experiment, we have found that the throughtput is dependent of the kind of source devices (Acer One, MacBook Pro, Lenovo). Since the Iperf generates flow at the congestion limit, we can suppose that emulated host is transmitting more rapidly than real host increases loss at the emulated switch, while real hosts does not. This explains the loss and throughtput results. Note decreasing the Iperf data rate, reducing the loss. The experiment with 1Gbps shows that the throughtput is limited around 700 Mbps.

3) Service Time

In the sub-experiment, tcpdump [35] has been used for capturing the incoming packets and outgoing packages of OpenFlow switches in order to calculate the total times that a packet spends in the switch. Results are shown in TABLE IV.

When a packet arrives at a switch, the switch lookups its flow entry for forwarding. If there is no flow entry, the switch will send a PACKET_IN message to its controller. Then, the controller sends a FLOW_MOD message back to allow the switch to add a flow entry. Thus, the switch is able to forward

TABLE IV
SERVICE TIME

	Setting					
	Emulation Experiment			Real Experiment		
	OVS	CPqD	OVS/CPqD	OVS	CPqD	OVS/CPqD
First Packet (no flow entry)	2.7878	4.8472	0.5757	2.6904	2.662	1.0107
Next Packet	0.1565	0.0995	1.5729	0.3084	0.203	1.5192

packets to the right destination.

Service time is the time between packet arrival and departure at the switch. It is a processing time inside the switch plus the flow entry established time, which are a sending process of a PACKET_IN message from a switch to the controller, controller processing time and sending process of a FLOW_MOD message back to the switch. Results in TABLE IV show that OVS switch took shorter time to connect to the controller than OVS switch in Mininet experiment, but the results are similar in the real experiment.

An average switch processing time of one packet has been analyzed. It has been derived from a period of time that the switch treats each packet (except the first packet). The results show that the CPqD switches can process the packets about 1.5 times faster than OVS switches.

V. CONCLUSION AND FUTURE WORK

This paper presents an original analysis of SDN tools. A commercial router (TPLINK WR1043ND v2) has been implemented in the real networks and is set as an OVS switch and CPqD switch in order to compare the bandwidth capability of a single UDP and TCP traffic.

From our experiments, every result acquired by Mininet and the real network (such as throughput and the percentage of packet losses) presents in the same trend. In our scenarios, the OVS switch has the maximum bandwidth capability higher than the CPqD switch for both of UDP and TCP traffics. Moreover, we found that the OpenWRT OVS switch is easier to install than the OpenWRT CPqD switch in the real network.

However, please note that even the CPqD switch has its bandwidth capability limitation about 50 Mbps, it provides faster OpenFlow handshake and uses less memory space for the package installation than those of the OVS switch. For future works, our studies should be extended to wireless SDN networks and several OpenFlow switches should be implemented and analysed in real networks.

REFERENCES

[1] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," in ONF White Paper, April 2012.

[2] C. Alaettinoglu, "Software Defined Networking," in PACKET DESIGN, 2013.

[3] NS3 OpenFlow switch support in version 3.18. Available: http://www.nsnam.org/docs/release/3.18/models/html/openflow-switch.html

[4] Gustavo J.A.M Carneiro. NS-3:Network Simulator 3. Available: https://www.nsnam.org/tutorials/NS-3-LABMEETING-1.pdf

[5] S.Y. Wang, C.L. Chou, and C.M. Yang, "EstiNet 8.0 OpenFlow Network Simulator ans Emulator," in Vol.51, Issue 9, September 2013.

[6] Mininet: An Instant Virtual Network on your Laptop (or other PC). Available: http://mininet.org

[7] M. Chan, C. Chen, JX. Huang, T. Kuo, LH. Yen, and CC. Tseng, "OpenNet: A Simulator for Software-Defined Wireless Local Area Network," IEEE WCNC'14, April, 2014.

[8] C. Argyropoulos, B. Arslan, J. Aznar, K. Baumann, K. Dombek, E. Escalona, D. Guija, E. Jacob, A. Juszczyk, J. Melnkov, A. Mendiola, S. Naegele-Jackson, T. Ogrodowczyk, D. Pajin, D. Pamiewicz, R. van der Pol, M. Przwecki, "Deliverable D13.1 (DJ2.1.1) Specialised Applications' Support Utilising OpenFlow/SDN," GEANT, March, 2015.

[9] Open Networking Foundation, "OpenFlow-enabled Transport SDN (ONF Solution Brief)," May 2014.

[10] J. Postel, "Transmission Control Protocol," in RFC 793, September 1981.

[11] Open Networking Foundation, "OpenrFlow Switch Specification version 1.5.1 (Protocol version 0x06)," March 2015.

[12] POX wiki. Available: https://openflow.stanford.edu/display/ONL/POX+Wiki

[13] Ryu 3.18 documentation: WELCOME TO RYU THE NETWORK OPERATING SYSTEM (NOS). Available: http://ryu.readthedocs.org/en/latest/index.html

[14] Pyretic. Available: http://http://frenetic-lang.org/pyretic/

[15] Trema. Available: https://trema.github.io/trema/

[16] Floodlight. Available: http://www.projectfloodlight.org/floodlight/

[17] ONOS. Available: http://onosproject.org/

[18] OpenDayLight Controller. Available: http://www.opendaylight.org/

[19] A. Shalimov, D. Zuikov, D. Zimarina, V.Pashkov, and R.Smeliansky, "Advanced Study of SDN/OpenFlow Controllers," CEE-SECR '13, October 2013.

[20] R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou, "Feature-based Comparison and Selection of Software Defined Networking (SDN) Controllers," WCCAIS 2014, October 2014.

[21] K. Kaur, S. Kaur, and V. Gupta, "Perfomance Analysis of Phyton Based OpenFlow Controllers," EEECOS 2016, June 2016.

[22] Open Networking Foundation, "OpenFlow Switch Specification Version 1.2 (Wire Protocol 0x03)," December 2011.

[23] Open Networking Foundation, "OpenFlow Switch Specification Version 1.3.0 (Wire Protocol 0x04)," June 2012.

[24] B. Pfaff, J. Pettit, T. Koponen, Ethan J. Jackson, A. Zhout, J.Rajahalme, J. Gross, A.Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The Design and Implementation of Open vSwitch," NSDI'15, May, 2015.

[25] CPqD OpenFlow 1.3 Software Switch. Available: http://cpqd.github.io/ofsoftswitch13/

[26] LINC - OpenFlow software switch. Available: https://github.com/FlowForwarding/LINC-Switch

[27] Pantou: OpenFlow 1.0 for OpenWRT. Available: http://archive.openflow.org/wk/index.php/Pantou_:_OpenFlow_1.0_for_OpenWRT

[28] OpenFlow 1.3 for OpenWRT. Available: https://github.com/CPqD/ofsoftswitch13/wiki/OpenFlow-1.3-for-OpenWRT

[29] OpenFlow for OpenWRT OpenFlow Wireless Network. Available: https://github.com/Farzaneh1363/OpenFlow-for-OpenWRT-OpenFlow-wireless-network-/wiki

[30] Turning TP-LINK WR1043NDv2.1 router into OpenFlow-enabled switch. Available: http://ljdelight.com/turning-tp-link-wr1043ndv2-1-router-into-openflow-enabled-switch/.

[31] OpenWrt Wireless Freedom. Available: https://downloads.openwrt.org/snapshots/trunk/ar71xx/generic/

[32] iPerf - The network bandwidth measurement tool. Available: https://iperf.fr/

[33] Wireshark. Available: https://www.wireshark.org/#learnWS

[34] tshark - Dump and analyze network traffic. Available: https://www.wireshark.org/docs/man-pages/tshark.html

[35] tcpdump(8)-Linux man page. Available: http://linux.die.net/man/8/tcpdump