Author(s): Asghar, Muhammad; Habib, M. Ahsan; Hämäläinen, Timo

Title: Performance evaluation of OpenFlow enabled Commodity and Raspberry-pi Wireless Routers

Year: 2017

Version:

Please cite the original version:

Asghar, M., Habib, M. A., & Hämäläinen, T. (2017). Performance evaluation of OpenFlow enabled Commodity and Raspberry-pi Wireless Routers. In O. Galinina, S. Andreev, S. Balandin, & Y. Koucheryavy (Eds.), NEW2AN 2017, ruSMART 2017, NsCC 2017 : Internet of Things, Smart Spaces, and Next Generation Networks and Systems (pp. 132-141). Springer International Publishing. Lecture Notes in Computer Science, 10531. https://doi.org/10.1007/978-3-319-67380-6_12

Performance evaluation of OpenFlow enabled Commodity and Raspberry-pi Wireless Routers

Muhammad Zeeshan Asghar, M. Ahsan Habib, and Timo Hämäläinen

Department of Mathematical Information Technology,
PL 35, 40014, University of Jyväskylä, Jyväskylä, Finland
{muhammad.z.asghar,mahsan.habib,timo.hamalainen}@jyu.fi
http://www.jyu.fi

Abstract. *Software defined network (SDN) allows the decoupling of data and control plane for dynamic and scalable network management. SDN is usually associated with OpenFlow protocol which is a standard interface that enables the network controllers to determine the path of network packets across a network of switches. In this paper, we evaluate openflow performance using commodity wireless router and raspberry pi with two different SDN controllers. Our test setup consists of wired and wireless client devices connected to openflow enabled commodity wireless router and raspberry pi. All clients used traffic generator tool to transmits data to a sink server host. The results are promising and paves the way for further research on using software defined wireless network*

Keywords: Software defined network (SDN), Wireless SDN, OpenFlow, performance evaluation, Raspberry-pi

1 Introduction

The modern computing environment are dynamic and requires scalable computing and advanced storing needs that are not achievable by traditional network architecture. In order to address this challenge, the existing network architecture needs to be upgraded with additional features. This situation pushes operators to upgrade their network management from static to dynamic and scalable computing and opens doors for Software Defined Networking (SDN) research.

The SDN is an umbrella term for various ways to use software to manage and manipulate networks. There are usually three views on SDN including, open SDN using openflow protocol, SDN via Application Programmable Interfaces (APIs) model where the functionality on networking devices is exposed using a rich API, and sdn via overlays.

The SDN has potential to offer high flexibility in network management . The key idea is to split the network forwarding function from the network control function. This can be achieved by separating the control and data planes. This allows a simpler and more flexible network control and management. SDN is usually associated with OpenFlow protocol which is a standard interface that enables the network controllers to determine the path of network packets across a network of switches. The network controller communicates with OpenFlow switches using the OpenFlow protocol through a secure channel. Using this connection, the controller is able to configure the forwarding tables of the switch. OpenFlow-based SDN technologies enable us to address the high-bandwidth, dynamic nature of computer networks; adapt the network functions to different business needs easily; and reduce network operations and management complexity significantly. As the cost involved to replace all legacy devices by new ones is high, the possibility of using commodity wireless routers with the OpenFlow might be a smart strategy to accelerate the deployment of SDN technologies.

This work aims at performance evaluation of Open cSwitch (OVS) running in commodity wireless router and raspberry pi as a linux kernel module. The results show that Openflow router provides good performance with low cost, therefore, the adapted wireless router with openflow compatibility can be widely utilized in home networks and small organizations. The rest of this paper is organized as follows. Section 2 describes the related work. Section 3 presents the evaluation strategy and proposed testbed architecture.Section 4 shows the experimental results and analysis. We conclude by proposing future research directions.

2 Related Work

In this section we briefly describe the related work. There have been few studies about the performance evaluation of SDN architectures and openflow. Tootoonchian et al. [6] studied the SDN controller performance by focusing on the control plane only. Rotsos et al. [7] proposed a tool for evaluating performance of SDN architecture consisting of several OpenFlow implementations, and measured raw performance of OpenFlow without comparison to other SDN solutions authors presented an architecture in [4] to improve the loopup performance of teh OpenFlow Linux kernerl module implementation. A cost effective alternative of implementing SDN testbed with Open vSwithc (OVS) is proposed in [5]. They used raspberry-pi as an alternate choice for the SDN switch. They implemented the SDN testbed with thte OpenFlow specification 1.0. The performance of net-FPGA and OpenFlow based software swicthed is compared in terms of maximum throughput. The results shows the similar performance with 1Gbps net-FGPA device. The shortcoming of this work is that the testbed is implementing SDN functionalities for wired network only and therefore it is not scalable.

A comparative study of the performance evaluation of a commodity wireless router and different SDN controllers is presented in [6]. The performance evaluation is conducted using the performance metrics such as throughput, delay, jitter and packet loss. The results show bottleneck when using UDP protocol with high rates and therefore, the commodity wireless router can be only deployed in reduced size networks. Other recent research work on SDN includes [7–12]. Some recent works on SDN testbed with raspberry-pi are [5, 8]. The performance of OpenFlow in commodity wireless networks are given in [6].

3 Proposed Testbed Architecture

In this section we describe our proposed testbed architecture and methodology to collect the performance measurements such as average bitrate, packet dropped, average delay and average jitter. The architecture overview of SDN testbed is shown in figure 1.

3.1 Network Achitecture

Our test setup consists of wired and wireless client devices and these devices connected to openflow enable commodity wireless router and raspberry pi. All the clients used traffic generator tool to transmits data to a server. The network design shown in figure 2. There are four clients and a server connected to the wired ports and one client connected through the wireless link. The SDN controller connected to wan port. In our testbed, we have selected TP-LINK WR1043ND ver 2.1 [13] and raspberry pi 3 [14] devices. The TP-LINK wireless router has a Qualcomm Atheros QCA9558 processor with 720MHz clock speed, 64MB memory, which is decent amount memory for a wireless router. The router has 4 LAN and 1 WAN Gigabit Ethernet interfaces and also has a 802.11n/g/b wireless network interface. On the other hand, raspberry PI 3 has a 4 core ARM Cortex-A53, 1.2GHz processor, 1 GB LPDDR2 memory and a Gigabit Ethernet interface. The device has also 4 USB 2 interfaces which allow for up to 4 additional Ethernet interfaces via USB-to-Ethernet adapters. We used 10/100Mbps USB to Ethernet adapters to connected clients to the raspberry pi OpenvSwitch. All our wired and wireless clients and the SDN controller are real machines have been running the Ubuntu 14.04 operation system. The SDN controller has an Intel dual core i5-5300U with 2.30 GHz processor and 4 gigabyte of memory.

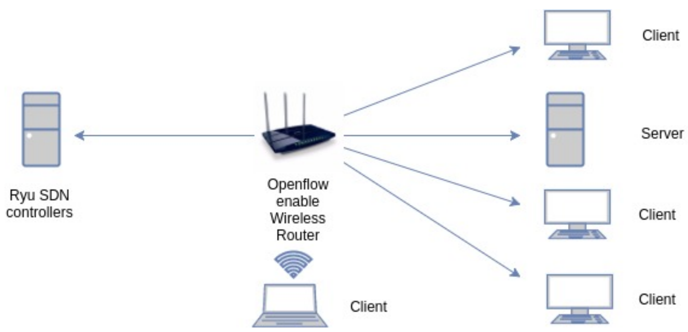

Fig. 1. Network topology used in the experiments. Wired and wireless client devices are generate traffic to a sink server. The wireless router interconnects all devices and is controlled by Ryu SDN controller

3.2 Software Design

We have used popular open source traffic generate tool called Distributed Internet Traffic generator `D-ITG` [15] which is a platform capable to produce IPV4 and IPV6 traffic by precisely reproducing traffic from many popular internet applications. `D-ITG` is also a network measurement tool able to generate most useful information from packet level such as throughput, delay, jitter, packet loss. In our experiment, Using `D-ITG` tools we were able restrict the packet size and control the total number of packets to be transmit and the duration of the generation experiment. In order to correctly measure the packet delay, we have implemented Network Time Protocol (NTP) [16] in our testbed environment so that all the clients and the server can be synchronization between them.

To ensure all clients and server can generate and receive rate up to 10 Gbps, we have replaced the default Libcap module in linux kernel with the `PF_RING` library [17]. `PF_RING` is a linux kernel module and user-space framework that allows us to process packets in high rates. The main idea is to bypass any and all notion of what is actually data on the wire and transmits as quickly as possible.

Raspberry-pi linux kernel is 3.7.11+ version based on debian Jessie. We installed OVS 2.3.1 to create an OpenFlow enabled switch which supports OpenFlow 1.3. The wireless router running the original equipment manufacturer (OEM) firmware The firmware OEM substitued by the OpenWRT 14.0 without OpenFlow module The OpenFlow 1.3 support in the OpenWRT firmware, using the OVS kernel module, was enabled.

3.3 SDN Controller

For this study, we have used some most popular controller in research area such as RYU [18] and Floodlight [19] to perform our experiment. Both controllers run simple switch learning application. The OpenFlow ver 1.3 protocol used to communication between OVS and SDN controller.

3.4 OpenvSwitch

In our experiment, we have installed and configured OpenvSwitch on OpenWrt [20] to showcase openflow capabilities. We complied OVS with openWrt running in linux kernel module. We have used OpenvSwitch ver 2.3.1 and OpenWrt ver 14.07 Barrier breaker image on TP-Link wireless router. The OpenFlow ver 1.3 protocol support was enabled in OVS on OpenWrt firmware. OpenVswitch is also run on Raspberry Pi as a kernel module. We have download the openvSwitch from the source code and its dependencies and complied with kernel header.

The raspberry Pi is driven by the raspbian Jessie with pixel (raspbian is a Debian based operating system) with kernel version 4.4.34. We have been running OVS version 2.7.90. OVS fail mode can be set to either standalone or secure mode.

In standalone mode, OVS will take the responsibility for forwarding the packets if the controller fails. In secure mode, only the controller responsible for forwarding the packets, and if the connect between OVS and controller lost all the packets are going to be dropped. In our test case scenarios, both OpenWrt and Raspberry Pi OVS switch set standalone as a fail mode, in order to eliminate the impact of any SDN controller in the process.

4 Experimental Results

In this section we describe the experimental results with different scenarios. The performance factors for the experiments were traffic generator clients, packet size, packet sending rate and transport protocols. In the experiments we used two different controllers i.e., ryu and floodlight. We evaluated average bitrate, average packet dropped, average delay and average jitter.

Figure 2 compares the average bitrate between SDN controller ryu and floodlight over openwrt and raspberry pi. The comparison shows that both sdn controllers have similar performance. Both controllers with openwrt shows high average bitrate around 40 - 50 kpps over opernwrt. However, the avergae bitrate in case of raspberry pi stays lower than the openwrt. Openwrt and raspberry-pi average bitrate almost same from 4kpps to 6kpps but raspberry-pi bitrate does not increase much between 7kpps and 30kpps, it starts to increase from 30kpps to 50kpps. On the other hands, openwrt shows similar pattern but openwrt bitrate increases significantly from 30 kpps and both controller shows almost similar bitrate. Raspberry-pi shows lower bitrate then openwrt because raspberry-pi has 4 (usb 2) ports which has limited data transfer rate.

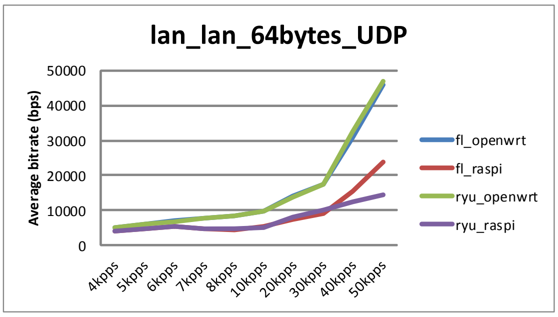

Fig. 2. *Average bitrate for experiments with UDP 64-bytes packets, using different OpenFlow controllers*

Figure 3 shows packet dropped for different SDN controller ryu and floodlight over openwrt and raspberry pi. The comparison shows that both sdn controllers have similar performance. There is no packet dropped in the scenario with openwrt. however in case of raspberry-pi there is significantly high. As can be seen in the figure 3, the ryu controller performance is slightly better than floodlight controller. There are not many packets drop until 40 kpps in openwrt for both controller but the packets dropped starts around 50kpps. If we look at the raspberry-pi openVswitch, we can see huge amount of packets drop using both controller. As mentioned previously raspberry-pi has limited data transfer rate so packets dropped increase significantly for huge number of packets specially 6kpps.

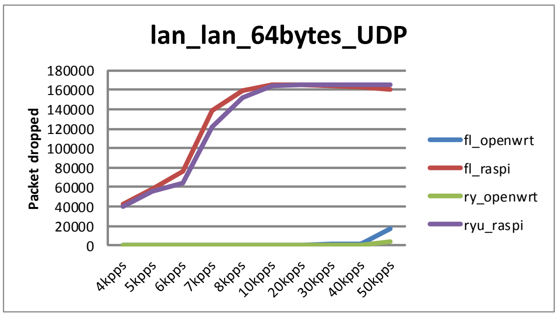

Fig. 3. *Average Packet dropped for experiments with UDP 64-bytes packets, using different OpenFlow controllers*

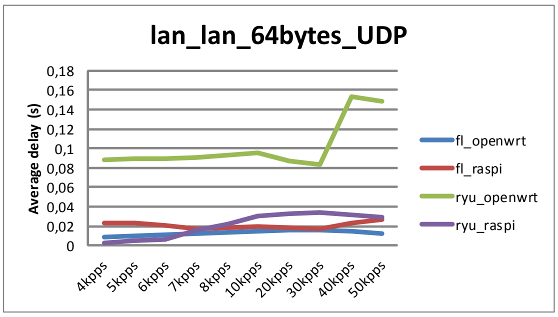

Fig. 4. *Average delay for experiments with UDP 64-bytes packets, using different OpenFlow controllers over openwrt and raspberry pi*

In figure 4 the average delay is not much when we used OpenWrt or raspberry-pi but average delay is high when we used openwrt with ryu controller. There were similar kinds of delay until 30 kpps but it increase significantly around 40kpps and 50kpps. In figure 5, similar to previous figure average Jitter are sim-

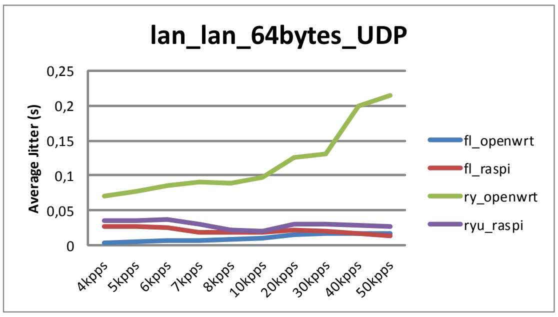

Fig. 5. *Average Jitter for experiments with UDP 64-bytes packets, using ryu controller over openwrt and raspberry pi*

ilar in both openwrt and raspberry-pi but average jitter incrementally increases in the case of ryu controller. These results show that there is a bottleneck around 30 kpps. In figure 6, When wifi client is used to transmit data to LAN client

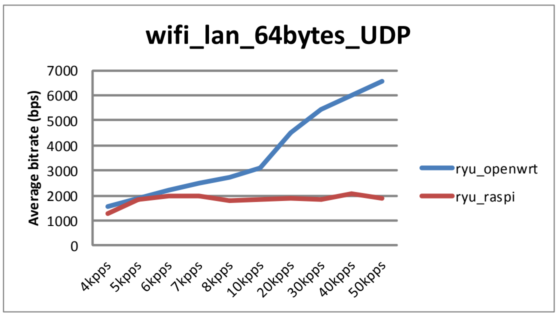

Fig. 6. *Average bitrate for experiments with UDP 64-bytes packets, using ryu controller over openwrt and raspberry pi*

raspberry-pi bitrate does not increase because of wifi adapter data transfer rate limitation but for the openwrt it is totally opposite, bitrate increases as more packets are transmitted from wifi client to lan client. As we can see from the

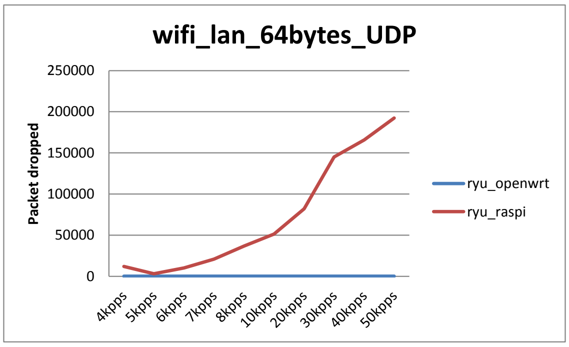

Fig. 7. *Average packet dropped for experiments with UDP 64-bytes packets, using ryu controller over openwrt and raspberry pi*

figure 7, that openwrt shows stable results and no packet dropped at all however, raspberry-pi shows huge number of packets dropped as more packets are transmitted on from wifi client to lan client. In figure 8,The openwrt shows high

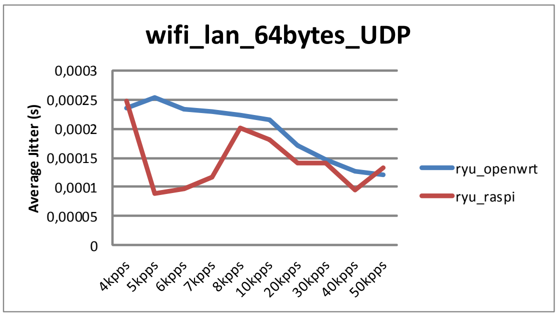

Fig. 8. *Average jitter for experiments with UDP 64-bytes packets, using ryu controller over openwrt and raspberry pi*

jitter on small number of packet and it started to decrease as more packets are transmitted. In case of raspberry-pi average jitter is high at 4kpps but it is low between 5kpps and 7kpps. The average jitter increases again at 8kpps and show a decaying pattern between 10kpps and 40kpps. In figure 9, results for TCP transport protocol are shown traffic generated from wifi client to lan client.

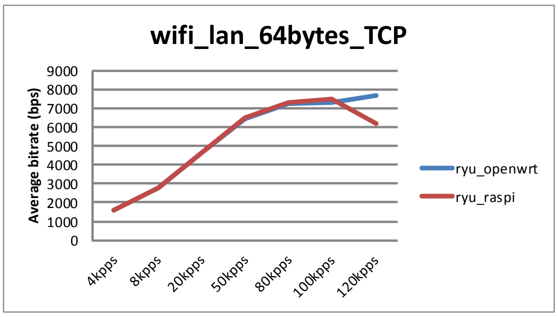

Fig. 9. *Average bitrate for experiments with TCP 64-bytes packets, using ryu controller*

Both openwrt and raspberry-pi showed similar increasing trend. The openwrt bitrate increased around 120kpps but average bitrate of raspberry-pi started to decrease because of external wifi adapter limitation.

5 Conclusion

In this paper we evaluated the performance of a commodity wireless router and raspberry-pi used as an OVS linux kernel module. Our proposed testbed is built on latest versions of controllers. The experiments are conducted with two different OpenFlow controllers and we observed the performance variations when using with different packet size, variable packet rate generation, traffic generating clients and transport protocols. It is found out that the performance of two different controllers is similar which indicates that the performance bottleneck is at OVS module. The experiments with UDP transport protocol show that the performance bottleneck exists when traffic rate exceed 40 kpps. The results show that the usage of the SDN architecture and OpenFlow standard introduces benefits in terms of maximum throughput, delay and jitter. The performance obtained suggests that a commoduty wireless router and raspberry pi can be used in scenarios of small networks. The results shows that adapted wireless router and raspberry-pi with Openflow can be widely used in small networks such as home networks, university campus, and small organizations. The proposed testbed is low-cost, less complex and scalable. The future work involves running experiments with complex scenarios using multiple controllers, thus allowing to evaluate the performance differences and bottlenecks.

References

1. Open Networking Foundation, Available: `http://opennetworking.org`
2. Tootoonchian, A., Gorbunov, S.: On controller performance in software-defined networks. In: Proc. USENIC Hot-ICE, (2012)

3. Rotsos, C., Sarrar, N., Uhlig, S., Sherwood, R., Moore, A. W.: An Open Framework for OpenFlow Switching Evaluation: Composing a Complex Biological Workflow through Web Services. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 1148–1158. Springer, Heidelberg (2006)
4. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)
5. Kim, H., Kim, J. and Ko, Y.B.: Developing a cost-effective OpenFlow testbed for small-scale Software Defined Networking. In Advanced Communication Technology (ICACT), 2014 16th International Conference on (pp. 758-761). IEEE.(2014)
6. Lima, L., Azevedo, D. and Fernandes, S.: Performance evaluation of OpenFlow in commodity wireless routers. In Network Operations and Management Symposium (LANOMS), Latin American (pp. 17-22). IEEE. (2015)
7. Araniti, G., Cosmas, J., Iera, A., Molinaro, A., Morabito, R. and Orsino, A.: OpenFlow over wireless networks: Performance analysis. In Broadband Multimedia Systems and Broadcasting (BMSB), 2014 IEEE International Symposium on (pp. 1-5). IEEE (2014)
8. Ariman, M., Seinti, G., Erel, M. and Canberk, B., 2015, November. Software defined wireless network testbed using Raspberry Pi of switches with routing add-on. In Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on (pp. 20-21). IEEE. (2015)
9. Brock, J.D., Bruce, R.F. and Cameron, M.E.: Changing the world with a Raspberry Pi. Journal of Computing Sciences in Colleges, 29(2), pp.151-153. (2013)
10. Sezer, S., Scott-Hayward, S., Chouhan, P.K., Fraser, B., Lake, D., Finnegan, J., Viljoen, N., Miller, M. and Rao, N.: Are we ready for SDN? Implementation challenges for software-defined networks. IEEE Communications Magazine, 51(7), pp.36-43.(2013)
11. Shalimov, A., Zuikov, D., Zimarina, D., Pashkov, V. and Smeliansky, R., 2013, October. Advanced study of SDN/OpenFlow controllers. In Proceedings of the 9th central & eastern european software engineering conference in russia (p. 1). ACM. (2013)
12. Vissicchio, S., Vanbever, L. and Bonaventure, O.: Opportunities and research challenges of hybrid software defined networks. ACM SIGCOMM Computer Communication Review, 44(2), pp.70-75. (2014)
13. 300Mbps Wireless N Gigabit Router TL-WR1043ND, http://static.tp-link.com/resources/document/TL-WR1043ND_V2_datasheet.pdf
14. Raspberry Pi Foundation, https://www.raspberrypi.org/
15. Distributed Internet Traffic Generator (D-ITG), http://www.grid.unina.it/software/ITG/
16. The Network Time Protocol (NTP), http://www.ntp.org/
17. pf_ring, http://www.ntop.org/products/packet-capture/pf_ring/
18. Ryu, https://osrg.github.io/ryu/
19. Floodlight, http://www.projectfloodlight.org/
20. Openwrt, https://openwrt.org/