Performance Analysis of OpenFlow Forwarders Based on Routing Granularity in OpenFlow 1.0 and 1.3

Viktor Šulák, Pavol Helebrandt, Ivan Kotuliak
STU in Bratislava
Bratislava, Slovakia
{viktor.sulak, pavol.helebrandt, ivan.kotuliak}@stuba.sk

***Abstract*—Software-defined networking is alternative approach to traditional switched networks where each switch and router makes its own forwarding decisions. Software-defined networking uses centralized controller making decision with multiple forwarders applying received rules. In this paper we analyze the performance impact of using micro and macro flows in software-defined networking and performance impact of OpenFlow version used. Our results quantify overhead when matching of flows on second to fourth layer of OSI model compared to matching on second layer only.**

I. Introduction

Software-defined networking (SDN) is a relatively new approach to building medium to large networks. It replaces distributed packet forwarding logic of every device in the network with centralized one called network controller. This creates two logical layers in topology – control plane and forwarding plane. Control plane uses applications to make and affect forwarding decisions of every device in forwarding plane. Since the logic is centralized, controller (control plane device) has complete overview of the network unlike devices in traditional packet-switching networks [1]. Using this advantage, network resources can be used more efficiently, preventing jams and collisions if possible since a controller has absolute overview over of the network [2].

OpenFlow is an open standard developed by non-profit industry consortium called Open Networking Consortium (ONF). OpenFlow defines a communication interface and protocol between control and forwarding plane of network devices, as seen in Fig. 1. It allows separation of control plane and its centralized software implementation, thus being considered to be a de facto SDN enabler.

Since SDN offers fine control over data flows, it is possible to process data differently than using traditional OSI layer 2 or even layer 3 aware switches. It is possible to process packets up to OSI layer 7. The level of logic used depends solely on SDN controller and its installed northbound applications [3]. This allows us to turn SDN forwarders into more sophisticated devices. First, incoming traffic sample is sent to controller using "packet in" message when forwarder receives data flow that does not match any rules installed in flow table. A controller can make decision what to do – it can order forwarder to send packet using specific port, drop the packet or install new flow forwarding entry [4]. These rules may match data flows on different OSI layers based on routing granularity used. The finer granularity allows data flows to be matched more specifically but introduces higher processing overhead [5]. Our work quantifies the effect of routing granularity on data flow performance. We use different SDN forwarders and controllers to do so.

Additionally, we made a fork of POX OpenFlow controller and upgraded it to use OpenFlow version 1.3 instead of 1.0 [6]. Using this controller, we are able to compare performance of virtual SDN forwarders that are able to work with both versions of OpenFlow. This allows us to compare performance of different OpenFlow versions or at least performance of implementations of libraries for different OpenFlow versions for the same virtual SDN forwarders.

Our work is structured into introduction chapter offering short introduction to SDN networking and comparing our work to related ones, performance evaluation where we propose testing setup, its implementation and present testing results. Later there is conclusion of these results and the list of related works cited.

Testing results compare latency and throughput on different trivial SDN topologies using different forwarders and controllers. Different test setups have different routing granularities used.

II. Related work

Our work shares in most with work [7] in which authors address the problem of efficient forwarding and simulate performance impact of different routing granularity levels in OpenFlow network. The main difference is that we use virtualized network topology based on software forwarders, instead of ns-2 network simulator.

Our work uses similar testing scenarios to [8] but we focus on different things. Authors of related article compare a performance of SDN architecture to traditional network under various workloads. They also analyze if there SDN architecture is ready to be used with more complex infrastructures. They do not take into account different routing granularity levels.

Authors in [2] compare two routing paradigms, reactive and proactive routing, and their influence on routing performance. We use reactive routing only.

ISSN 2305-7254

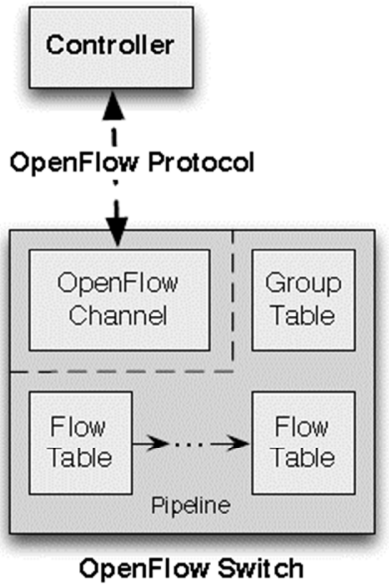

Fig. 1. Main components of an OpenFlow switch [9]

Article [10] compares OSI layer 2 switching and layer 3 routing performance using similar network parameters in SDN networks but authors compare it to tradition physical network.

None of mentioned articles measure real-life impact of SDN routing rules granularity in virtualized networks like ones used in production environment. Our work does this using some of forwarders and controllers used nowadays.

We also did not find any relevant comparison of performance differences between OpenFlow standard versions 1.0 and 1.3. This comparison is difficult to make since most of OpenFlow controllers are built for version 1.0 or are built for newer standards.

III. Performance evaluation

We compared performance of forwarding using several OpenFlow 1.3 compatible controllers and forwarders using differently grained data flows. We also added OpenFlow 1.3 support to POX controller so we can better compare performance of forwarders using different OpenFlow versions.

A. Design proposal

Testing of our solution was performed using Mininet testing platform. It provides us a simple way to test all the changes on OpenFlow virtual devices we make. It creates virtual instances of devices that allow testing of connectivity between them and tracing their communication. It enables multiple concurrent developers to work on a single complex network topology. Mininet contains a set of basic parameterized topologies for quick start of testing and is usable as-is after initial installation [11]. We used different topologies with several parameters sets to test different scenarios.

Mininet uses process-based virtualization instead of full virtualization. This provides much faster boot times, allows much larger number of virtual devices on single computer and higher bandwidth. The biggest downside of Mininet is that it supports Linux-based OpenFlow network devices only [10].

Assuming these characteristics, we consider Mininet to be the best suiting tool for our testing. The only better alternative for us is testing on real topology, but it was not possible because we did not have access to SDN-enabled hardware. Other platforms that we considered were OpenFlowVMS and Noxrepo.org VM Environment. We did not choose them because they are both deprecated and do not provide any support for OpenFlow version 1.3.

Testing was performed on single computer with no other user applications launched to provide results as relevant as possible. All tests were carried out repeatedly, usually 5 times.

B. Controller implementation

We chose to enhance POX, the open source OpenFlow 1.0 network controller, and add the support of OpenFlow 1.3 to it. Because of this, we develop the project in Python programming language. We created the OpenFlow library used by POX controller. It is the rewritten version of OpenFlow 1.0 library implementing numerous changes between specification versions. Also testing scenarios were rewritten to reflect wider possibilities and some core libraries were updated to load newer library rather than former one.

IV. Results

A. Comparison of different controllers and forwarders on the same topology

We started our proper testing with several available combinations of forwarders and controllers and we tested bandwidth between two hosts with four forwarders in linear topology between them using iperf. We can see results in the Table I and Fig. 2.

TABLE I. Average throughput for linear topology with 4 forwarders

Controller and forwarder	Average [Mbps]
Pox 1.3 + CPqD ofsoftswitch13	23,00
Ryu + CPqD ofsoftswitch13	27,00
ovs-controller 1.9.0 + Open vSwitch 1.9.0	997,00
ovs-controller 1.9.0 + Open vSwitch 2.1.0	1010,00
Pox Carp + Open vSwitch 1.9.0	942,00
Pox Carp + Open vSwitch 2.1.0	966,00
Pox 1.3 + Open vSwitch 2.1.0	1020,00
Ryu + Open vSwitch 2.1.0	1030,00

We discovered several interesting things. The first that everybody would notice is that ofsoftswitch13 performs significantly worse than Open vSwitch. This is because of their different natures. Open vSwitch runs in the kernel space and therefore has immediate access to the core of operating system. Since the Mininet creates lightweight virtual machines for every forwarder and host, we assume that it also instantiates

core functionality of software forwarder. On the other side, ofsoftswitch13 runs in the user-space and therefore has to access the kernel as any other application through system application interface [12]. These calls to the core slow down every application but it is impossible to run every application in kernel space.

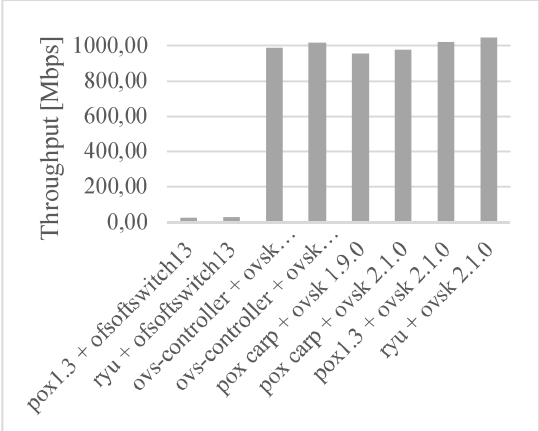

Fig. 2. Measured values for the first test on linear topology with 4 forwarders with different controllers and forwarders

Second thing we observed is the fact that our controller running OpenFlow 1.3 version of "l2_learning" behavior has performance comparable to Ryu controller running "simple_switch_13" behavior. Also, the controller performs better than original version of POX running OpenFlow 1.0 version of "l2_learning" behavior. We hoped for this result from the beginning of the work on our POX controller improvement. It means that we improved POX controller in positive way.

A surprise for us was the ovs-controller from the makers of Open vSwitch running inside the virtual machine. It performs better than original POX controller does. It is possible that the network bridge between VirtualBox-created virtual machine and main operating system lowered the performance of Ryu and both POX versions. We assume that the difference is not big since we tested it on the computer equipped with four-threaded CPU, fast memory with enough capacity and SSD hard drive.

Next, we tested the two-way delay (round-time trip) of the packets between endpoint hosts in the same topology. When compared to the previous part of test, we omitted Open vSwitch 1.9.0 that we upgraded irreversibly to version 2.1.0. We can see the results in Fig. 3 and Table II below.

We performed the second part of our test using ping command between two endpoint hosts in the linear topology with four forwarders between them. We measured the time of the first communication separately from the next ten. This is because the first communication needs to fill the MAC table of the forwarders and ARP table of the hosts. Most importantly, flow matching rules are also installed on forwarders. As we can see on the Fig. 4, we found out that CPqD ofsoftswitch13 performs significantly better than Open vSwitch during the connection setup. Since we used the same controllers with the same behavior and the same hardware and software setup, this shows us real differences between forwarders.

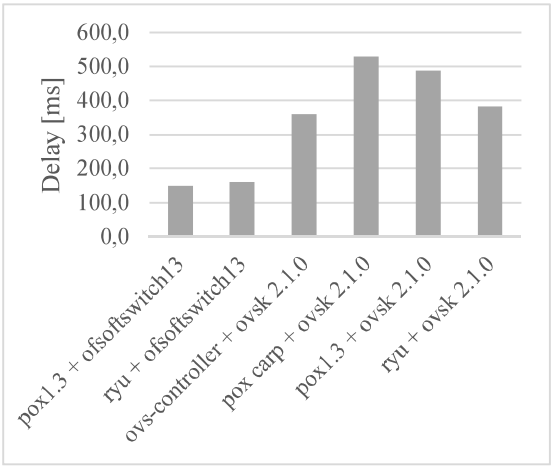

Fig. 3. Two-way delay of the first packet of new packet flow using linear topology with 4 forwarders

TABLE II. AVERAGE TWO-WAY DELAY FOR LINEAR TOPOLOGY WITH 4 FORWARDERS

Controller and forwarder	**Average delay [ms]**	
	first	next 10
Pox 1.3 + ofsoftswitch13 (OF 1.3)	150,2	2,5
Ryu + ofsoftswitch13 (OF 1.3)	160,1	2,5
ovs-controller 1.9.0 + OVS 2.1.0 (OF 1.0)	359,4	0,2
Pox Carp + OVS 2.1.0 (OF 1.0)	530,4	0,3
Pox 1.3 + OVS 2.1.0 (OF 1.3)	487,5	0,3
Ryu + OVS 2.1.0 (OF 1.3)	382,6	0,3

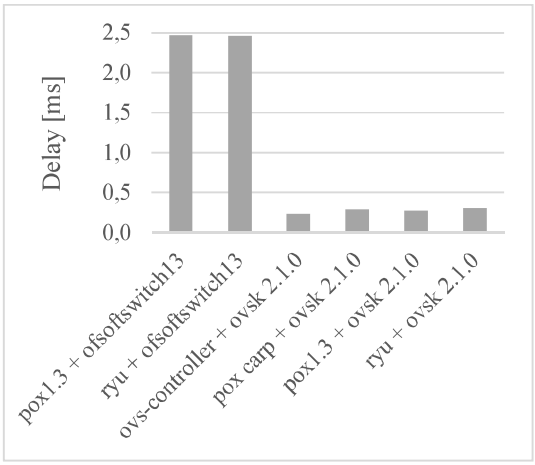

Fig. 4. Average two-way delay of the next 10 packets using linear topology with 4 forwarders

As we can see on the Fig. 4, things have changed after the initial flow setup. Open vSwitch takes advantage of its direct access to the system core and therefore fast access to the switching ports. This advantage is significant for elephant flows.

B. *Impact of the number of forwarders on their performance in Mininet*

As the next test, we measured the impact of the number of forwarders between endpoint hosts in Mininet.

We presumed that the difference in performance is not large and we tested it. Since Mininet creates the different number of forwarders with the same total processing power available, Mininet should divide available resources evenly between all of the forwards created. We can see the results of measurements in Table III and Fig. 5, Fig. 6, and Fig. 7.

TABLE III. IMPACT OF THE NUMBER OF FORWARDERS ON THEIR PERFORMANCE IN MININET

TOPOLOGY	FORWARDERS END-TO-END	THROUGH-PUT [MBPS]	TWO-WAY DELAY [MS]	
			FIRST	NEXT
single, 2	1	111,66	63,50	0,55
linear, 2	2	56,23	85,41	1,24
linear, 3	3	36,70	116,53	1,83
linear, 4	4	26,95	138,44	2,15
linear, 5	5	21,68	142,05	2,65
linear, 6	6	17,80	168,93	3,39
linear, 7	7	14,83	178,88	4,82
linear, 8	8	12,60	188,96	4,90

For the testing purposes, we used our modified controller with "l2_learning" behavior and ofsoftswitch13 forwarder. Two different topologies were used – either single forwarder with two hosts connected or linear topology with two hosts with different number of forwarders in line between endpoint hosts.

On the figure below, we can see that throughput measured by iperf tool falls exponentially. When we look at the numbers in table above, we can see that multiplication of throughput measures by the number of forwarders results in similar numbers every time. This confirms our assumption about dividing the resources evenly among multiple forwarders.

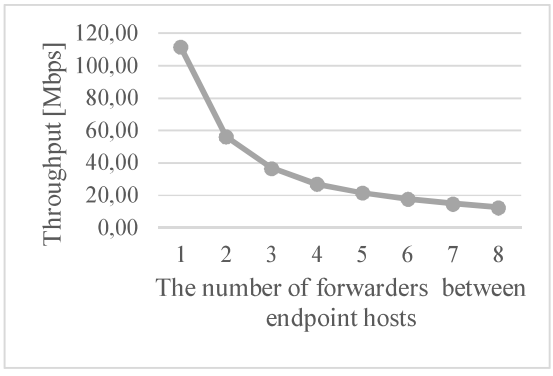

Fig. 5. End-to-end throughput dependency on the number of forwarders passed in Mininet

The next parameter we followed was the two-way delay measured using the ping command between endpoint hosts. This measurement did not deplete available resources so the measured numbers are affected only by the forwarders and not by the performance of underlying hardware. In this test, we used Open vSwitch forwarder instances instead of ofsoftswitch13.

We assumed that the round-trip delay would rise linearly with the number of forwarders passed end-to-end. When we look at the graphs on Fig. 6 and Fig. 7, we can see that we were right. This proves that each forwarder performs the same actions that take very similar time each time. Significantly longer delay times on Fig. 6 are caused by populating of ARP tables on each forwarder, by identifying of unmatched data flows on forwarders, sending their parameters to controller and installing of new rules on forwarders.

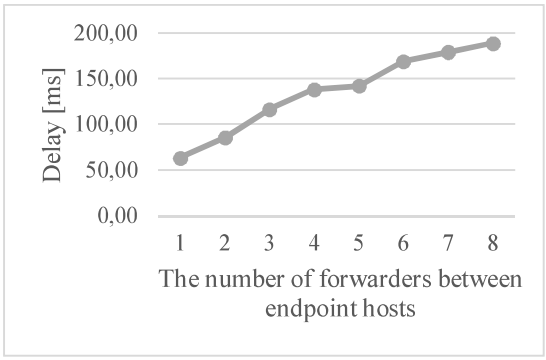

Fig. 6. Two-way delay dependency on the number of forwarders of the first communication

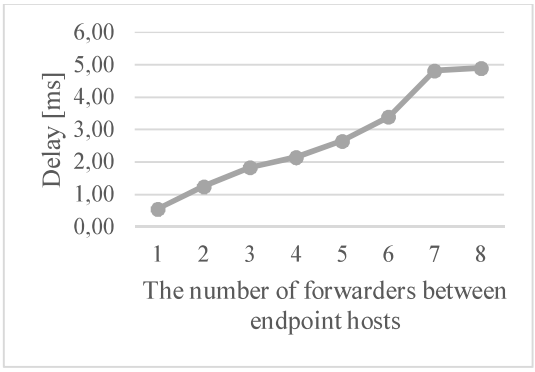

Fig. 7. Two-way delay dependency on the number of forwarders of the following 10 communications

C. *Comparison of flow matching on OSI layer 2 and OSI layers 2 to 4*

In this test, we compared the performance of the OpenFlow flow matching on OSI layer two and matching of OSI layers up to four using different amount of OpenFlow extensible match parameters sent. The wider the match is, the quicker it should take to process the traffic. This can be useful in switched environment with great amount of processed data.

These principles are similar to traditional route aggregation principles that allows us to reduce routing tables size and therefore increase scalability and reduce processing overhead. Finer grained matching of data flows also introduces larger

routing tables and higher performance requirements. This processing overhead is even bigger since software forwarders have to decapsulate packets, analyze them to higher degree, process and encapsulate them using only CPU without any aid from application specific integrated circuits. Unmatched data flows also add communication overhead to controller and processing there.

Three types of topologies were used – simple topology with central forwarder and connected hosts, linear topology with several forwarders connected serially with a host connected to each of them and tree topology in the form of rooted binary tree with multiple tiers of forwarders and hosts connected to bottom leaves. The forwarders used are instances of ofsoftswitch13. All topologies used are parametrized topologies of Mininet with two hosts communicating with maximal possible usage of bandwidth to test throughput using iperf tool.

This test is based on the usage of "l2_only" flag we added to the "from_packet" method responsible for creating the flow match from incoming packet. This flag limits the OXM fields created to OSI layer two addresses only. If it is not used, our testing environment uses addressing up to OSI layer four.

Summary throughput on all forwarders in the path is 96 to 116 Mbps with OSI layer 2 switching and is distributed on forwarders since our tests were carried out on single piece of hardware. Table IV and

Fig. 8 show improvement from 11% to 22% when the wider matches are used in different topologies. This is caused by lower communication overhead between SDN controller and forwarders and by lower processing requirements with decapsulating and encapsulating of network frames.

V. Conclusion

Our results show that usage of wider data flow matching in OpenFlow offers better throughput in network compared to finer grained matching. On the other hand, narrower matching is necessary when using quality of service techniques. This relates to traditional route aggregation benefits but software-defined networks introduce their own specific performance drawbacks that can be mitigated by data flow matching optimization.

TABLE IV. Impact of wider OXM match on switching performance

Matched OSI layers / Topology	L2 [Mbps]	L2-L4 [Mbps]	Difference [%]
Single forwarder, 4 hosts	109,00	98,03	11,19
Single forwarder, 8 hosts	108,50	94,27	15,10
Line of 4 forwarders, 4 hosts	28,88	25,00	15,53
Line of 8 forwarders, 8 hosts	12,02	10,77	11,61
Tree of 3 forwarders, 2 tiers, 4 hosts	38,67	31,47	22,88
Tree of 7 forwarders, 3 tiers, 8 hosts	21,22	18,12	17,11

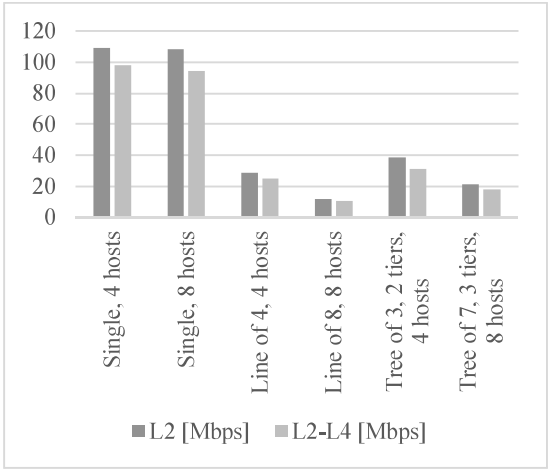

Fig. 8. Impact of wider OXM match on switching performance

As we can see in previous chapter, performance impact of narrower OpenFlow extensible matching is not dramatic but is highly dependent on network topology.

Next step would be to perform our tests in real-life network topology or simulate several topologies corresponding to typical network setups.

We also compare OpenFlow versions 1.0 and 1.3 using different controllers and forwarders. The most appropriate comparison is between POX controller in carp revision (version 0.3.0) with Open vSwitch forwarder set to support OpenFlow version 1.0 and our fork of POX controller with updated library with Open vSwitch set to support OpenFlow version 1.3. Results of this comparison shows us that OpenFlow 1.3 setup performs slightly better. Difference can be caused by implementation of OVS OpenFlow library but is closest comparison possible since cores of both forwarder and controller were the same and used only single different library.

Acknowledgment

The authors would like to thank for financial contribution from the STU Grant scheme for Support of Young Researchers. This work is a partial result of the Research and Development Operational Program for the projects Support of Center of Excellence for Smart Technologies, Systems and Services, ITMS 26240120005 and Research and Development Operational Program for the projects Support of Center of Excellence for Smart Technologies, Systems and Services II, ITMS 26240120029, co-funded by ERDF. It is also a part of projects APVV-15-0731 and VEGA 1/0836/16.

References

[1] V. Šulák, Controlling the routing in the next generation networks, Bratislava: FIIT STU, 2014.

[2] P. M. Fernandez, "Comparing OpenFlow Controller Paradigms Scalability: Reactive and Proactive," in 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), Barcelona, 2013.

[3] K. Hyojoon and N. Feamster, "Improving network management with software defined networking," IEEE Communications Magazine, vol. 51, no. 2, pp. 114 - 119, 14 February 2013.

[4] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov and R. Smeliansky, “Advanced study of SDN/OpenFlow controllers,” in Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia, Moscow, 2013.

[5] A. S. Iyer, V. Mann and N. R. Samineni, "SwitchReduce: Reducing switch state and controller involvement in OpenFlow networks," in IFIP Networking Conference, 2013, Brooklyn, NY, 2013.

[6] V. Šulák, "pox-1.3," FIIT STU, 11 June 2014. [Online]. Available: https://github.com/vsulak/pox-1.3.

[7] H. Kudou, M. Shimamura, T. Ikenaga and M. Tsuru, "Effects of routing granularity on communication performance in OpenFlow networks," in 2011 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PacRim), Victoria, BC, 2011.

[8] A. Gelberger, N. Yemini and R. Giladi, "Performance Analysis of Software-Defined Networking (SDN)," in 2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, San Francisco, CA, 2013.

[9] Open Networking Foundation, "OpenFlow Switch Specification Version 1.3.1," 6 September 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf.

[10] A. Bianco, R. Birke, L. Giraudo and M. Palacin, "OpenFlow Switching: Data Plane Performance," in 2010 IEEE International Conference on Communications (ICC), Cape Town, 2010.

[11] Mininet Team, "Mininet Overview," [Online]. Available: http://mininet.org/overview.

[12] G. Cockshull, "OpenFlow Managed Home Gateways," University of Southampton, 2013.

[13] J. Yu, "RFC 2791: Scalable Routing Design Principles," July 2000. [Online]. Available: https://tools.ietf.org/html/rfc2791.