Investigating the Impact of Network Topology on the Processing Times of SDN Controllers

Christopher Metter*, Steffen Gebert*, Stanislav Lange*, Thomas Zinner*, Phuoc Tran-Gia*, Michael Jarschel§
*University of Wuerzburg, Germany
Email: {christopher.metter|steffen.gebert|stanislav.lange|zinner|trangia}@informatik.uni-wuerzburg.de
§Nokia, Munich, Germany
Email: michael.jarschel@nsn.com

***Abstract*—Software Defined Networking (SDN) introduces the concept of logically-centralized controllers in charge of managing the forwarding behavior of network elements. The new possibilities enabled through the centralization of control logic come with a certain risk: The controller might become a performance bottleneck. Therefore, ensuring sufficient controller performance is one of the crucial tasks prior to a successful SDN deployment. Furthermore, fine-grained traffic engineering, e.g., to achieve higher link utilization, results in a higher frequency of requests that are sent to the controller, which leads to an increased controller load. It is therefore important to analyze the capabilities of SDN controllers prior to deployment. This paper investigates two software implementations, the OpenDaylight and Ryu controllers. The control message throughput of different controllers has been studied several times already; however, it is not yet known what influence the number and topology of connected switches have. This paper investigates this influence in detail for a fat-tree data center topology and a WAN topology as well as 261 topologies with varying characteristics from the *Internet Topology Zoo*.**

I. Introduction

The new networking paradigm Software Defined Networking (SDN) introduces a separation of data plane and control plane. While the data plane resides in the network elements, i.e., white-box switches, the control plane is logically centralized and implemented as an SDN controller software running on one or more servers. In order to allow an exchange of information between these two planes, a communication protocol for this *Southbound API* [1] is required. Today's widely used OpenFlow [2] protocol is such a communication protocol, which enables vendor-independence on both planes.

This not only allows hardware manufacturers to offer OpenFlow-compatible switches that can be used in conjunction with existing OpenFlow hardware and controller software, but also fosters innovation. Numerous SDN controller implementations have been developed, starting with the reference implementation provided by the NOX controller [3], each new controller adding additional features. They focus on particular use cases as well as scalability improvements.

Scalability and performance are crucial for successful production deployments of OpenFlow-based SDN networks. To evaluate an SDN controller's performance, several solutions have been introduced over the past years. The evaluation of processing rates as provided by Cbench [4] has the downside that it is not detailed enough to uncover unfairness between the switches in terms of their response times. In our previous works, we introduced OFCProbe [5, 6], an open-source controller benchmarking software that emulates a number of OpenFlow switches in order to test the controller's behavior under given circumstances, e.g., a number of connected switches with a defined rate of new flow arrivals. With per-switch statistics and by investigating the number of yet unanswered messages, unfair processing priorities depending on from which switch a message was received, have been shown.

The current work extends these investigations and examines the influence of the managed network topology on the processing times of the controller, as well as the processing times of different OpenFlow message types. Therefore, the behavior of two SDN controller software implementations (OpenDaylight (ODL) [7] and Ryu [8]) are investigated in the following. For this purpose, OFCProbe is used to emulate different network topologies and expose them to the controller software, which is tested.

The remainder of this work is structured as follows: Section II covers the technical background of OpenFlow, as well as related work. Section III describes the measurement setup, followed by Section IV, where results are given. Finally, Section V summarizes this work and gives an outlook towards further extensions.

II. Background and Related Work

This section introduces the technologies used in this paper. First, a brief summary of Software Defined Networking and OpenFlow is provided. Then, related work on controller benchmarking and performance of the SDN control plane is highlighted.

A. Software Defined Networking (SDN)

To remedy the limitations of modern-day networks, SDN introduces the decoupling of the control and data plane of networking components and establishes well-defined interfaces between these components [1]. The Southbound API specifies communication between the data plane, usually implemented in hardware, and the control plane, implemented as software. This decoupling allows a much faster evolution of networks, independent of long release cycles of network devices. In contrast to most traditional networking devices, *Northbound APIs* allow an integration with other infrastructure components, e.g., to allow better automation or improved network utilization.

This work focuses on the Southbound API and the currently most-widely used OpenFlow protocol [2].

B. OpenFlow

As the control logic is shifted from the switching hardware into a controller software running on a server, the hardware representing the data plane lacks any functionality for making forwarding decisions. All of this information has to be set by the OpenFlow controller using the OpenFlow protocol.

The information, which packets should be sent out on which port is stored in the switch's *flow table*. For fast processing, this is implemented in the hardware, as well as matching of incoming packets against the entries in the flow table. Each of these entries, the *flow rules*, specifies a set of *header fields*, i.e., Ethernet, IP, or TCP headers, based on which it is decided, which action should be executed for each packet.

After a switch starts with an empty flow table, information about where to forward incoming packets is required. Therefore, the switch sends a copy of the packet in a *packet-in* message towards the connected controller and awaits instructions through a *packet-out* message. As forwarding of every packet to the controller will drastically lower network performance, the usual behavior is that the controller also reacts with a *flow-mod* message including match criteria and an action to the switch, which is then installed in the flow table. All consequent incoming packets matching such criteria can now be processed solely by the switch–until the rule times out and the next packet again triggers a packet-in message towards the controller. With this behavior, the so-called *reactive* behavior allows the OpenFlow controller to react upon changes in the network, e.g., changes in the link utilization, to instruct the switches to use a different forwarding path.

Given the varying granularity of match criteria available in flow rules, a huge number of requests towards the SDN controller might be triggered, e.g., if rules match particular IP addresses as well as TCP ports. This requires a high throughput rate as well as short processing times by the SDN controller. Therefore, the performance of the SDN controller is of great interest and thus will be also covered in this work.

C. Related Work

A model of forwarding speed and blocking probability of an OpenFlow switch was established in [9]. The estimation of packet sojourn times and the packet loss probability in an OpenFlow-enabled environment are the benefits of this model. Accordingly, these estimations enable the performance approximation of an OpenFlow architecture with certain given parameters. The results show that the OpenFlow controller processing speed has the largest effect on packet sojourn times.

Similar effects have been described [10]. The impact of transmission delays between switches and their connected controller for different flow setup strategies is investigated. The authors compare three OpenFlow controllers having network virtualization support, namely *Trema*, *Floodlight*, and the *NOX* controller with their own *LibNetVirt* extension. The authors studied how an emulated delay of 5 or 10 ms on the control path affects ICMP round-trip times, TCP-based download times, as well as packet loss of UDP transfers. In contrast to [10], the studies presented here explicitly ignore the transmission delays by placing the measurement point inside the SDN controller, as it will be explained in Section III-A.

Previously, OpenFlow controller benchmarking tools such as *Cbench* [11] and *OFCBench* [12] were introduced, however offering only rudimentary options and measurement data or limited scalability. The authors previously introduced *OFCProbe* [5], a platform independent OpenFlow controller analysis tool, available as open-source software [6]. OFCProbe implements the switch-side of the OpenFlow protocol and emulates many of these while connecting to a controller. By avoiding to use full virtual switch implementations like Open vSwitch and directly creating control plane traffic instead of first generating data plane traffic, the overhead is reduced compared to other approaches. OFCProbe allows to measure performance on a per switch level based on multiple attributes, e.g., packet throughput, response times, or CPU utilization of the controller host, while the number and topology of emulated switches is specified in the program code. One contribution of this work is an extension of OFCProbe's functionality which allows for loading of topologies from GraphML files, e.g., from the *Internet Topology Zoo* [13].

III. Measurement Setup

This section describes the processing chain of OpenFlow messages within the controllers, while at the same time providing an insight on which measurement points are used to gain performance metrics from inside the controller software where they are defined. Furthermore, the test setup used to conduct the measurements is described as well.

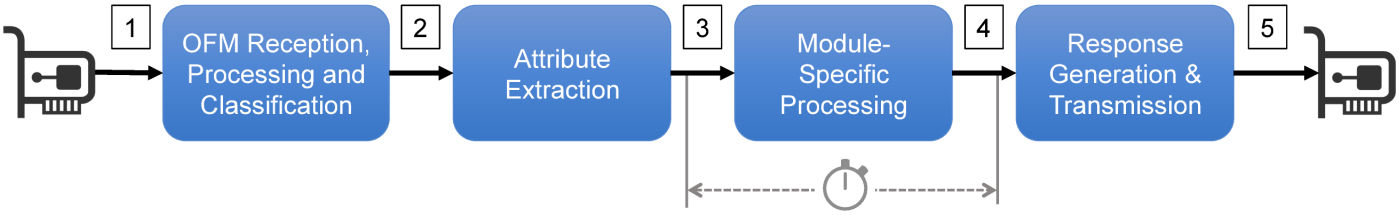
A. Controller Architecture and Sensor Placement

To understand the influence of the controller's architecture on its performance, a thorough investigation is performed. The abstract flow graph with modules involved in the OpenFlow (OF) message processing progress is shown in Figure 1: First, the OF packets are received by the controller's network card (1), before the byte stream is decoded into an OF message, processed, and then classified according to its message type (2). According to this classification, attributes are extracted from the message and delegated to the responsible application module (3). In the case of packet-in messages, the module computes a packet-out or a flow-mod message (4). This response message is filled with computed values, encoded into a byte stream, and sent back to the switch (5).

As can be observed in the table of Figure 1, the controller implementations vary in their particular implementation: ODL divides the process into smaller steps distributed over multiple modules. In Ryu, a lot of functionality is put into the central controller module, which is able to receive messages and start their processing. After the decision has been made by the launched controller application, the central controller module takes over again, and generates the outgoing message, encodes it into a byte stream and transmits it.

To conduct the following measurements, including only the processing times of the packets corresponding to packet-out and flow-mod messages, sensors are inserted immediately before (3) and after (4) [1].

[1]The modified source code for ODL and RYU modules can be found on https://github.com/lsinfo3/manfi2015-perf-ODL and https://github.com/lsinfo3/manfi2015-perf-ryu



Controller	OFM RX, Processing, Classification	Attribute Extraction	App-specific Processing	Response Generation, TX
ODL	Message Decoding, Internal Switch TX/RX Interface, Classification	Packet Attribute Extraction, Arphandler	Layer 2 Forwarding Switch Module	Packet-out or flow-mod Generator, Internal Switch TX/RX Interface, Message Encoding
Ryu	Central Controller Module, Message Decoding, Event Generation, Event Propagation, Classification	Layer 2 Switch Module		Central Controller Module Handles Generation, Encoding and Transmission

Fig. 1. Control packet flow and involved modules within the SDN controllers.

B. Testbed Setup and Used Software Versions

The used testbed consists of two directly connected servers[2], one running the OpenFlow controllers, the other running OFCProbe in version 1.0.4, both using a Linux operating system[3]. The used controller versions are the Hydrogen release of ODL as well as Ryu 3.14. Both controllers are started with a Layer 2 forwarding switch application.

C. Experiment Process

The course of all experiments is as follows: The controller software is started on one server and OFCProbe on the other. OFCProbe will now emulate a configured switch topology and start generating control plane traffic. At all times during the experiment, only one switch causes packet-in requests, while all other virtual 4-port switches are idle. A virtual host is emulated at every port not connected to another switch. The generated traffic consists of packet-in messages originating from each directly connected virtual host trying to communicate with all other virtual hosts in the topology, including those connected to the same switch. After the switch finishes transmitting its requests, the next switch can begin generating traffic. To mitigate influences of multiple switches trying to communicate with the controller, a waiting time of 2 seconds is introduced when changing switches.

IV. Evaluation

In this section, the results of the conducted experiments are presented, starting with a detailed investigation of three different switch topologies: In order to get a performance baseline, 20 switches without any interconnection and only with one connected host each are evaluated. Then, a fat-tree topology with 20 switches as well as the Viatel topology with 87 switches taken from the Internet Topology Zoo [13] are investigated in detail. Finally, a study of 261 more topologies is presented in aggregated form.

[2]Sun Fire X4150, 2x Intel Xeon L5420 QC 2.5 GHz (2 x 4 Cores), 16 GB RAM, 4-Port Gigabit Ethernet NIC

[3]OFCProbe host: Debian Squeeze x64; OF controller: Ubuntu 14.04 LTS x64

The following figures show one particular time series graph with processing times of the controllers. To obtain statistically significant results, each test is repeated five times and the following figures also present 95% confidence intervals.

A. Unconnected Switches (Baseline)

This first scenario investigates a network of 20 switches without links between them and only one connected host per switch. When the OF packet-in for an emulated data plane packet of the host to a yet unknown host is sent, the controller will return a flood instruction due to a lack of knowledge about the destination. Flow-mod messages will not appear in this context and thus, only processing times for pairs of packet-in / packet-out messages are considered in this scenario.

Figure 2 shows the processing times for the ODL and the Ryu controllers. Two main observations are: First, the measurements for both controllers show a high degree of periodicity. This behavior can be caused by multiple phenomena: On the one hand, the experiment procedure itself is periodic, featuring bursts of 16 packets followed by pauses between successive bursts. The number of observed peaks is 18 resp. 19 for ODL and Ryu. This is in line with 300 measured values, i.e., $\frac{300}{19} \approx 16$ indicates one peak each 16 measured values. On the other hand, sleep mechanisms may be triggered after an idle interval in both controllers in order to save CPU resources. Such a behavior would imply that the first packet in a burst needs to interrupt the sleep operation or even reinitialize the process, which in turn leads to a longer total processing time. Packets belonging to the same burst, however, arrive fast enough and encounter an active controller. Thus, their processing time is lower. After the pause between bursts, controllers will return to their sleep mode and the first packet of the next burst requires a higher processing time. The second observation is that Ryu is outperformed by ODL with respect to the processing times of packet-out messages in this case. While ODL achieves mean values of around 0.11 ms, Ryu takes significantly longer, resulting in a mean processing time of around 0.16 ms. However, this it is not the main objective of this paper, to compare controllers with each other, but instead to study the influence of the managed topology.

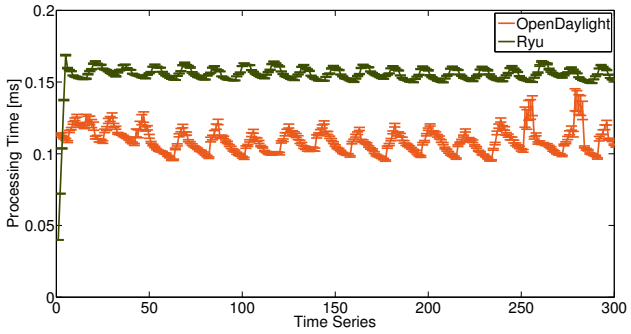


Fig. 2. Packet-out processing times for 20 unconnected switches.

Additionally, there is a difference in peak-to-peak amplitudes for the two controllers, with ODL's amplitude being almost twice as big as Ryu's. This indicates a difference in the implementation of the sleep mechanism. Furthermore, it might be possible that garbage collection of the particular programming language influences this behavior.

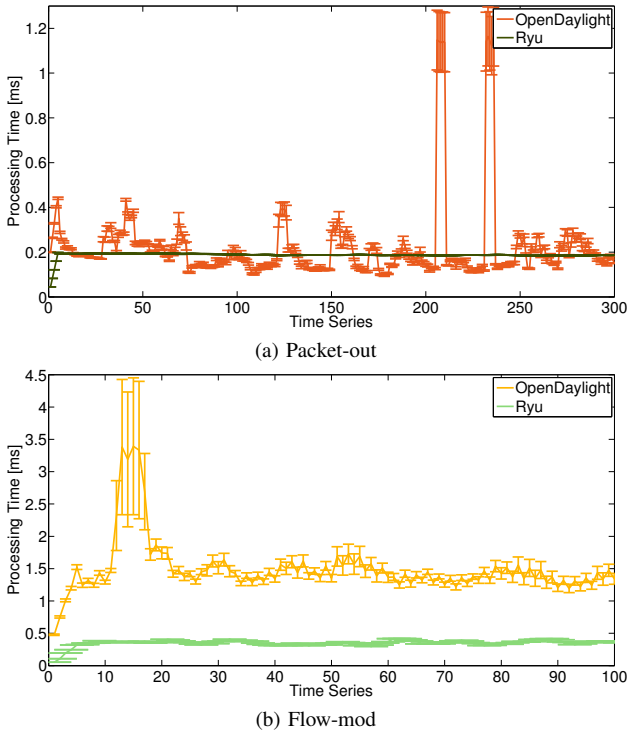
B. Fat-Tree Topology

The fat-tree topology can be thought of as a reference data center topology. This section deals with the results obtained by emulating a fat-tree topology with 20 switches, where the 8 edge switches have two emulated hosts each. Again, 16 packets emulating a data plane packet from one host to another are sent per burst per host. Once the controller has learned the location of a destination host, it will now also send flow-mod messages that, however, will be ignored by OFCProbe.

Figure 3 shows the packet-out and flow-mod processing times for both controllers. While the processing times for packet-out messages reach an almost constant value of around 0.2 ms in the case of Ryu, corresponding values of the ODL controller are less predictable. In addition to processing times being in the range between 0.1 and 0.4 ms, there are outliers that are even beyond 1 ms. While processing times of the ODL controller are often lower than those of the Ryu controller, the outliers may not be acceptable in certain scenarios. Thus, no definite statement about controller preference with regard to the processing times of packet-out messages can be made in the case of the presented fat-tree topology.

Results for flow-mod processing times in the fat-tree topology are displayed in Figure 3b. As opposed to 300 distinct values for the packet-out message processing times, only 100 values are recorded for the flow-mod messages. This difference is explained by the context in which each message type is generated. While packet-out messages often correspond to the processing of a single packet which triggers a flood command, flow-mod messages are produced only when the destination host is known to the controller which, in turn, allows computing the forwarding paths.

Initially, processing times of ODL rise from 0.5 ms to around 3.5 ms. However, the they stabilize after 20 processed flow-mod messages. While the mean values slightly oscillate around the 1.5 ms mark, overlapping confidence intervals of the 5 runs indicate no statistically significant variation. Ryu shows similar behavior, where flow-mods processing constantly rises to a value of 0.4 ms during the first 8 measurements and barely



(a) Packet-out

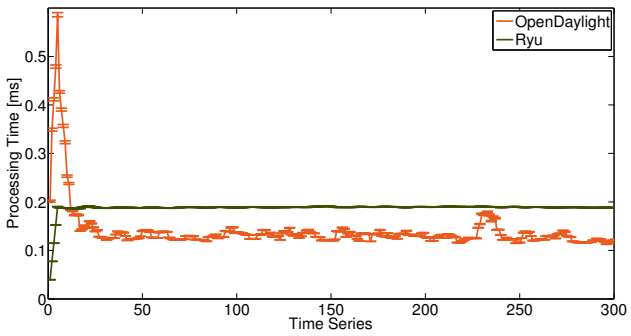
(b) Flow-mod

Fig. 3. Processing times for the fat-tree topology emulation.

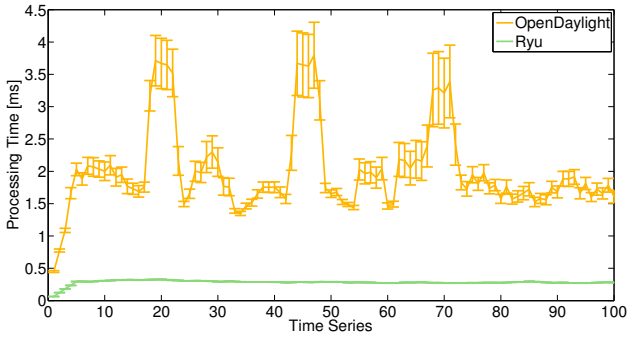
changes in the remaining time. In contrast to the packet-out processing times, the Ryu controller clearly outperforms the ODL controller with respect to flow-mod processing times in the fat-tree topology. In addition to the fact that the mean processing times for Ryu are almost four times lower, the variation of the values is also significantly reduced. The latter can be derived from the smaller width of confidence intervals as well as from a lack of outliers. For both controllers, the flow-mod processing times are higher than those for packet-out messages. The processing speed of the latter is lower by a factor of around 2 and 8 in the case of Ryu and ODL, respectively.

C. Viatel Topology

The Viatel topology is a WAN topology and consists of 87 switches. Figure 4 shows the behavior of the two controllers using the Viatel topology. The processing times for packet-out messages indicate short start-up phases for both controller implementations, after which relatively stable values are attained. In case of ODL, values begin at 0.2 ms, rise to almost 0.6 ms, and even out at around 0.12 ms after 20 messages. One slight deviation can be observed during the last third of the experiment. A more detailed analysis is required in order to explain this phenomenon. Ryu, however, shows processing times which rise from around 0.05 ms and reach an almost constant value of 0.2 ms after 5 packet-out messages. While the processing time of the Ryu controller behaves almost exactly the same as in with the fat-tree topology, a significant decrease in delay and variance can be observed for the ODL controller. When considering only values obtained after the initial start up phase, ODL strictly outperforms Ryu by a margin of almost 0.8 ms with respect to processing times of packet-out messages.



(a) Packet-out



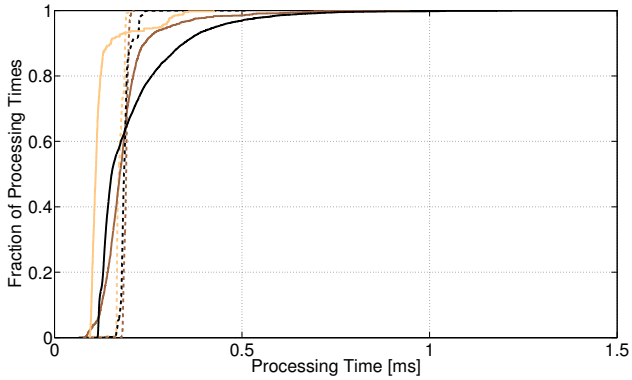
(b) Flow-mod

Fig. 4. Processing times for the Viatel topology emulation.

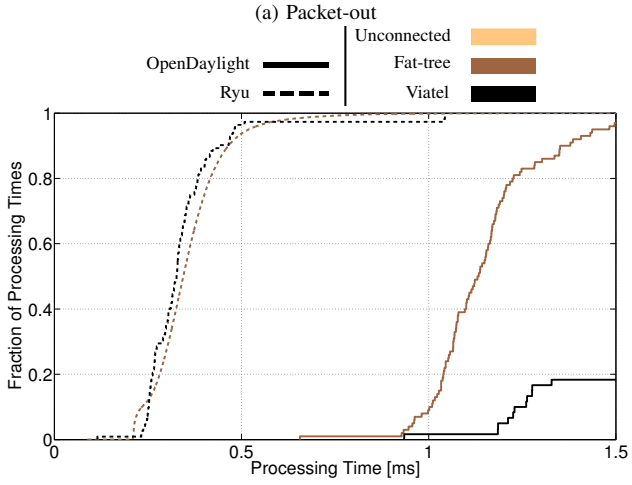
When considering processing times for flow-mod messages, on the other hand, Ryu achieves significantly lower values compared to ODL. Figure 4b illustrates this relationship. Furthermore, the shape of the curve and the absolute values reached by the Ryu controller are barely distinguishable from those observed in the context of the fat-tree topology. In the case of ODL, however, the processing times increase from around 1.5 ms in the fat-tree topology to about 2 ms for the Viatel topology. Additionally, a higher frequency of peaks and larger confidence intervals are present. This leads to the assumption of a correlation between the topology, particularly of the number of nodes, and the resulting processing times.

Figures 5a and 5b provide an aggregated view on all measurements presented so far by showing the cumulative distribution function (CDF) of processing times for packet-out and flow-mod messages. Each line corresponds to a combination of message type, topology, and controller. On the x-axis, observed processing times are displayed, while the y-axis shows the fraction of experiments in which a particular configuration led to processing times below the corresponding value.

The first observation is that the lines form two groups, i.e., one group with 95% quantiles below 0.5 ms and two curves with response times of at least 0.7 ms. In general, packet-out messages are processed faster than flow-mod messages and Ryu is faster than ODL, especially with respect to flow-mod messages. Once again, a large gap between the curves corresponding to flow-mod processing times of Ryu and ODL is observed. Furthermore, Ryu's response times appear to be more stable as their interquantile range is lower than for ODL.



(a) Packet-out



(b) Flow-mod

Fig. 5. Processing time distribution per topology, controller and message.

D. Aggregated Results for 261 Topologies

Previous sections indicated a possible relationship between the network topology and the controllers' performance in terms of processing times. To further investigate this phenomenon, the study continued with an evaluation of all topologies available in the Internet Topology Zoo [13]. The aggregated results show the performance of the controllers with increasing topology complexity. For this analysis, topology complexity is quantified by the sum of number of edges and vertices. This choice is based on the previous investigations, where graph size appeared to have an impact on controller performance.

Figure 6 shows the results for all 261 Internet Topology Zoo topologies. On the x-axis, the sum of vertices and edges (resp. emulated switches plus links) is shown, while the mean processing time in milliseconds is depicted on the y-axis. To provide a better view of the results, the *Kentucky Datalink* topology comprising 754 switches and 899 links had to be omitted from the figure. However, even for this large topology, processing times matched with the ones shown.

Ryu shows the most reliable performance, indicated by almost constant processing times for each message type, independent of the topology. For packet-out messages, a mean of 0.15 ms is observed. For flow-mods, the mean is slightly higher with 0.22 ms. For ODL, the message type has an influence on the processing time: Whereas the packet-out message values all are found at around 0.2 ms, the flow-mod messages are highly

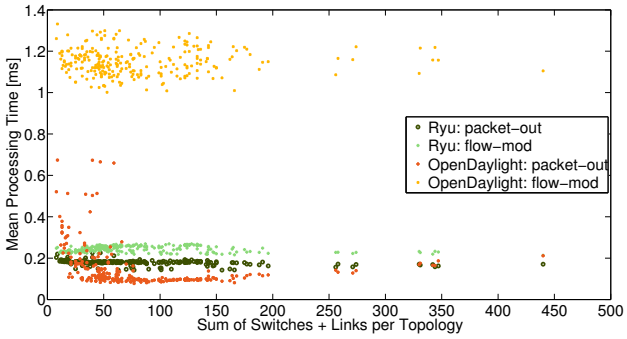


Fig. 6. Dependency of processing times on topology size.

scattered around 1.2 ms. In contrast to previous assumptions, the topology complexity has no influence on the performance of any of the two controllers, regardless of the message type.

The CDFs shown in Figure 7 show the same behavior as observed before. Ryu's response times are relatively constant between 0.2 and 0.25 ms, no matter which type of message. For ODL's packet-out messages, the interval of response times is spread between 0.1 ms to almost 0.7 ms. The corresponding flow-mod processing times are even higher and in the range of 1 ms to 1.3 ms. The CDF of the flow-mod messages shows the behavior of the ODL controller. Here, the response times are spread over an interval, ranging from values larger than 1 ms and consistently increasing to values beyond 1.3 ms. However, as shown in Figure 6, no correlation between response time and topology complexity can be found. Furthermore, no indication for a correlation of processing times with various other graph complexity measures like edge density, diameter, and betweenness could be found during the experiments. Detailed results had to be omitted due to brevity reasons.

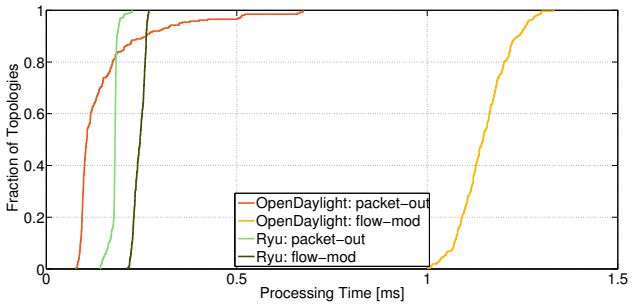


Fig. 7. Distribution of processing times for all Topology Zoo topologies.

V. Conclusion

This paper investigates processing times of OpenDaylight and Ryu SDN controllers. The evaluation focused on the processing times of packet-in/packet-out and flow-mod messages that are generated when a switch receives a packet of an unknown flow or when a flow rule should be installed in the switch, respectively. As it was yet unclear, how the switch topology affects the controller performance, three different topologies were studied in detail (20 unconnected switches, a fat-tree topology consisting of 20 switches, and a WAN topology consisting of 87 switches). The initial observation, namely that the number of switches influences the processing times, is disproved through an evaluation of 261 topologies from the Topology Zoo. These topologies vary with respect to their link density as well as with the amount of switches present in the network, the largest one comprising 754 switches. In total, the Ryu controller was found to outperform OpenDaylight in most of the scenarios, especially regarding the time until a flow-mod message is sent to the switch. Other than that, no correlation of processing times with any of the investigated graph metrics including the number of switches and links, link density, and diameter could be observed. This might be caused by the simplistic learning switch implementation used for the investigations. More sophisticated mechanisms, e.g., proactive installation of rules on the whole path that the packet will take, might lead to a different behavior here. Additionally the results of the controllers running with a more sophisticated Layer 3 switching module are of interest, too. These tasks, however, remain for future work. To summarize, as of yet there is no indication that the number of connected switches has any performance implication.

Acknowledgment

This work has been carried out in the framework of the CELTIC EUREKA project SASER-SIEGFRIED (Project ID CPP2011/2-5), and it is partly funded by the BMBF (Project ID 16BP12308). The authors alone are responsible for the content of the paper.

References

[1] M. Jarschel, T. Zinner, T. Hossfeld, P. Tran-Gia, and W. Kellerer, "Interfaces, attributes, and use cases: A compass for SDN," *Communications Magazine, IEEE*, vol. 52, no. 6, pp. 210–217, June 2014.

[2] B. Heller, "Openflow switch specification, version 1.0.0," http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf, OpenFlow Consortium, Tech. Rep., 2009.

[3] Nicira, "Noxrepo," http://www.noxrepo.org/, 2013.

[4] R. Sherwood and K.-K. Yap. Cbench controller benchmarker. [Online]. Available: http://sourceforge.net/projects/cbench/

[5] M. Jarschel, C. Metter, T. Zinner, S. Gebert, and P. Tran-Gia, "OFCProbe: A Platform-Independent Tool for OpenFlow Controller Analysis," in *5th IEEE International Conference on Communications and Electronics (IEEE ICCE 2014)*, Da Nang, Vietnam, Aug. 2014.

[6] C. Metter. (2014) OFCProbe. [Online]. Available: https://github.com/lsinfo3/ofcprobe

[7] The Linux Foundation. OpenDaylight. [Online]. Available: http://www.opendaylight.org/

[8] R. S. F. Community, "Ryu SDN framework," http://osrg.github.io/ryu/, 2014.

[9] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and performance evaluation of an openflow architecture," in *Proceedings of the 23rd international teletraffic congress*. International Teletraffic Congress, 2011, pp. 1–7.

[10] D. Turull, M. Hidell, and P. Sjodin, "Performance evaluation of openflow controllers for network virtualization," in *High Performance Switching and Routing (HPSR), 2014 IEEE 15th International Conference on*. IEEE, 2014, pp. 50–56.

[11] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "Oflops: An open framework for openflow switch evaluation," in *Passive and Active Measurement*. Springer, 2012, pp. 85–95.

[12] M. Jarschel, F. Lehrieder, Z. Magyari, and R. Pries, "A Flexible OpenFlow-Controller Benchmark," in *European Workshop on Software Defined Networks*, Darmstadt, Germany, Oct. 2012.

[13] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *Selected Areas in Communications, IEEE Journal on*, vol. 29, no. 9, pp. 1765–1775, 2011.