Estimating the Flow Rule Installation Time of SDN Switches when Facing Control Plane Delay

Anh Nguyen-Ngoc, Stanislav Lange, Stefan Geissler,
Thomas Zinner, and Phuoc Tran-Gia

University of Würzburg
{anh.nguyen, stanislav.lange, stefan.geissler,
zinner, trangia}@informatik.uni-wuerzburg.de

Abstract. The software defined networking (SDN) paradigm has numerous benefits for network operators, including cost aspects, flexibility, and programmability. In the centralized SDN architecture, the controller can order the installation of flow rules in the switches it manages via FlowMod messages. Since the processing time of these messages has a direct impact on the reaction time of the network, it is a key performance indicator for switches and quantifying it in a reliable manner is required for ensuring state consistency between the control and the data plane. Furthermore, real world deployments not only consist of different data plane hardware, but may feature varying control plane delays.
Hence, in this work, we investigate the impact of such a delay on the FlowMod processing time of OpenFlow switches. Firstly, we identify a significant heterogeneity between data plane hardware in terms of processing times as well as the underlying TCP-level behavior. Secondly, we show that despite this heterogeneity, combining switch specific information with delay measurements at the controller can be used to reliably infer FlowMod processing times.
We confirm our results with measurements in a dedicated testbed that is comprised of three different hardware switches, three different SDN controllers, and several high precision measurement devices.

Keywords: OpenFlow, FlowMod, Transmission Delay, SDN.

1 Introduction

Background. Several aspects of today's networks are affected when the paradigm of software defined networking (SDN) is employed. In addition to the separation of control and data plane, the SDN architecture is characterized by a logically centralized control plane. The latter is achieved by migrating control plane functionality from the network devices to a dedicated controller, i.e., software that runs on commercial off-the-shelf (COTS) hardware. The two planes communicate via protocols like OpenFlow [1], which implement the open southbound API [2].

Goal. Prior to migrating their network to an SDN-based deployment, operators need to assert that the resulting network meets the performance requirements for the particular use case. On the one hand, such requirements include data plane aspects like packet forwarding speed. On the other hand, control plane performance characteristics such as the processing time of FlowMod messages in OpenFlow switches play an important role. In our previous work [3], we investigate different approaches for assessing these processing times during the network run time. These include measurements within the SDN controller module, packet dumps on the controller host, as well as measurements from dedicated wiretaps that are placed in the network. However, even a dedicated control plane channel might be subject to latency in a real world deployment. Consequently, the goal of this work consists of analyzing the impact of control plane delay on the FlowMod processing times and the estimation accuracy of proposed methods.

Key insights. Firstly, our experiments show that given knowledge regarding the current control plane delay and switch hardware, it is possible to accurately infer the time until FlowMods are active in the data plane of the switch. Secondly, we observe that the limited buffer size of many hardware switches leads to TCP flow control behavior that significantly reduces the throughput of FlowMod messages and thus, the time until flow rules are installed. Finally, we show that the controller implementation can also have a significant impact on the performance w.r.t. the flow setup time due to different sending and packetization behavior.

Testbed. We obtain our results in a dedicated testbed with three different hardware switches and three different SDN controller implementations. In these measurements, we use wiretaps as well as the Spirent C1 testing platform and traffic generator to ensure a reliable ground truth with high precision.

The remainder of this work is structured as follows. We discuss related work in Section 2. In Section 3, the possible communication schemes for exchanging OpenFlow FlowMod messages are presented alongside the testbed setup and the resulting measurement options. Measurement results are covered in Section 4 and Section 5 concludes the paper.

2 Related Work

This section covers two main areas of related work. On the one hand, approaches for evaluating the performance of different aspects and components of an SDN architecture are presented. On the other hand, an overview of mechanisms for identifying and addressing the heterogeneity of SDN switches is provided.

Techniques for testing SDN-based networks in a holistic fashion are discussed in [4]. Before addressing the long term goal of integrated tests, it is necessary to understand the behavior of the individual network elements, i.e., controllers and switches. In an effort to provide means to test switch behavior with respect to compliance with the OpenFlow protocol specification, the authors of [5] present the OFTest suite. In contrast to this work, they focus on functional testing rather than performance tests.

The study conducted in [6] features a dedicated hardware traffic generator in order to test the data plane performance of Linux-based OpenFlow switching. In a similar setup, the authors of [7] investigate the characteristics of virtual switches and underlying virtualization techniques. In both works, the main interest lies in the data plane performance of the different switch implementations. This work, on the other hand, investigates the control plane performance of OpenFlow-enabled switches under varying network conditions.

OFLOPS [8] is a software framework for testing OpenFlow switch performance in the data plane as well as in the control plane. Its extension, OFLOPS-Turbo [9] is capable of 10 GbE traffic generation and utilizes the open-source NetFPGA-based OSNT [10] traffic generator and capture system. In contrast, we focus on the processing time of FlowMod messages in order to assess the effects of control plane delays on the resulting performance.

Analytical approaches like [11] and [12] investigate the influence of different network parameters on the performance of an OpenFlow architecture. Since such models are often based on measurements, the accuracy of these measurements also positively affects the quality of the resulting models. Therefore, one key aspect of our analyses is the accuracy of the available measurement mechanisms. A methodology for assessing the accuracy of measurements in the SDN context is presented in [13]. In addition to measurements performed by an SDN controller module, wiretaps installed at both ends of a communication channel serve as a means of providing the ground truth. This technique is also applied in the experiments that are conducted during the course of this work.

Several previous works highlight the heterogeneity of SDN switch hardware in terms of functionality, performance, and OpenFlow protocol compliance [14,15]. Unexpected or unreliable behavior such as additional delays and inconsistency between control and data plane pose several risks with respect to security as well as correct forwarding behavior. Hence, this heterogeneity needs to be taken into account for proper planning and design of real world deployments.

Some aspects of the heterogeneity, e.g., OpenFlow protocol compliance, are addressed by approaches such as TableVisor [16] and FlowConvertor [17] that introduce abstraction layers to translate given OpenFlow messages to device specific directives that take into account the behavior of individual switch hardware. While their focus is on maintaining functional homogeneity, we address the performance aspect. Finally, methods for data plane verification and consistency checks between data and control plane are proposed in [18]. However, rather than focusing on the identification of faulty switches, we are interested in performance prediction.

3 Methodology

In this section, two mechanisms for sending FlowMod messages from an SDN controller to OpenFlow switches are presented. Furthermore, we provide an overview of the experimental setup alongside measurement parameters and the configuration of the hardware that is used.

3.1 Mechanisms for Sending FlowMod messages

There are two types of methods that are used for installing OpenFlow rules in a switch: asynchronous and synchronous or *addFlow* and *addFlowAsync*, respectively. In the first case, every FlowMod is followed by a BarrierRequest, and the next FlowMod is sent to the switch if and only if the corresponding BarierReply has already been received and thus, the controller is informed that the previous FlowMod is successfully installed. On the other hand, the *addFlowAsync* mechanism generates a set of FlowMod messages and sends only one BarrierRequest afterwards. The difference between the two mechanisms is illustrated in Figure 1, together with the measurement parameters that are considered in this work.

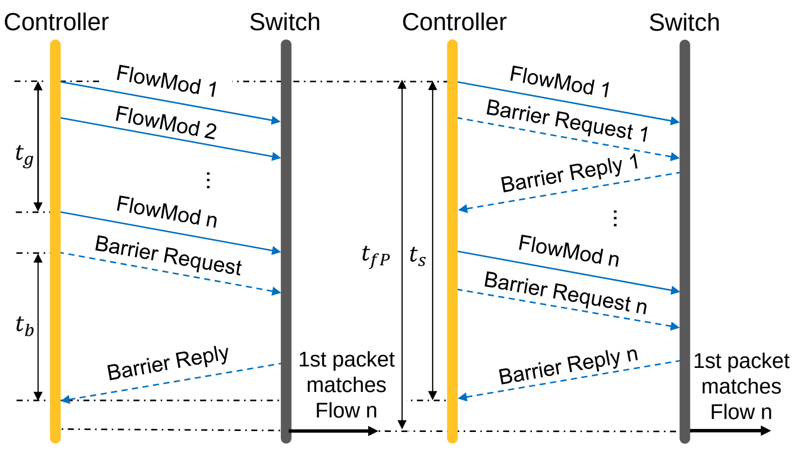


Fig. 1. Asynchronous and synchronous methods for adding flows to an OpenFlow switch.

On the left side of Figure 1, t_g represents the time that the controller needs to generate n FlowMod messages in case of *addFlowAsync* and t_b is the duration between BarrierRequest and BarrierReply. The time between the first FlowMod and the last BarrierReply indicates how long it takes the switch to finish setting up n rules and is denoted as t_s in both cases. Finally, t_{fP} denotes the time difference between the first FlowMod message and the first data plane packet that is forwarded by the switch according to the last FlowMod it received. This verifies that the corresponding flow entry is actually installed in the data plane of the switch.

3.2 Testbed Setup

In order to investigate the impact of transmission delay on the estimation of FlowMod message processing times, experiments are performed in a testbed that is set up according to Figure 2. In addition to a computer[1] which runs the SDN controller that is connected to an OpenFlow switch, two dedicated hosts[2] act as traffic source and traffic sink.

Furthermore, a computer with two 1 Gbps network interface cards (NICs) runs Ubuntu 16.04 and emulates the transmission delay in both directions, i.e., from the switch to the controller and vice versa. The red lines indicate links with delay, which is set via the tc command[3]. A Net Optics tap device[4] is inserted between the controller and Netem PC with the purpose of mirroring all traffic that passes through the corresponding link to the monitoring machine, an HP Proliant DL32 server. This server is equipped with an Endace DAG (Data Acquisition and Generation) 7.5G2 card, which has 2 Gigabit Ethernet ports, to capture every incoming packet.

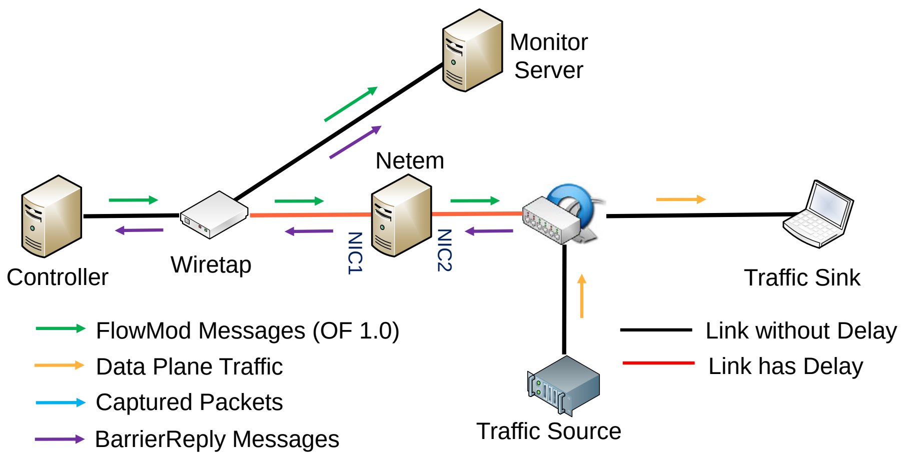


Fig. 2. Logical testbed setup.

Three different controllers are utilized in this work. Firstly, the latest version of the Python-based Ryu controller[5] is used in conjunction with an additional module that allows the generation of FlowMod messages according to the two aforementioned methods. Secondly, a module with similar function is built for the Java-based OpenDaylight controller[6]. Finally, the Spirent C1 testing plat-

[1] Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz/16GB RAM
[2] Intel(R) Core(TM)2 Duo CPU E8500/4G RAM
[3] `sudo tc qdisc add dev [interface] root netem delay [delayValue]`
[4] `http://www.ixiacom.com/products/ixia-gig-zero-delay-tap/`
[5] `http:https://osrg.github.io/ryu/`, v4.18
[6] `https://www.opendaylight.org/software/downloads/hydrogen-base-10`, Hydrogen release

form[7] with the OpenFlow Testing Package allows emulating an SDN controller. Furthermore, the Spirent C1 is capable of emulating the traffic source and sink, simplifying the testbed setup.

In this work, a total of three OpenFlow switches are used. Their specifications are displayed in Table 1.

Table 1. Switches used in this work.

Switch	**CPU**	**Memory**	**Software**
Pronto 3290	MPC8541 825 MHz	512MB	PicOs 2.0.14 (Open vSwitch v1.10.0)
Quanta T1048	MPC8541 825 MHz	1024MB	PicOs 2.6 (Open vSwitch v2.3.0)
NEC PF5240	PowerPC 667 MHz	1024 MB	OS-F3PA v5.0.0.1

3.3 Experiment Procedure

At the beginning of each measurement, the OpenFlow table in the switch is guaranteed to be empty. This is achieved by sending corresponding FlowMod messages to the switch before starting an experiment. Then, the controller sets up basic rules for exchanging ARP packets between the traffic source and sink. Later, these rules allow the traffic to be forwarded correctly to the destination without additional interaction with the controller. After that, another rule for dropping all packets that do not match any entry in the OpenFlow table of the switch is installed. Doing this prevents interference with the controller's performance due to an enormous number of PACKET_IN messages being forwarded to it. Meanwhile, the traffic source sends UDP traffic to the specific UDP port of the traffic sink using the Iperf[8] software. However, the packets can not reach the traffic sink due to the lack of a matching entry in the switch and are dropped. Afterwards, the controller generates either a batch of FlowMods followed by a BarrierRequest message or a series of alternating FlowMod and BarrierRequest messages. In both cases, the last FlowMod matches the aforementioned UDP traffic.

The results are collected by means of several approaches. Firstly, capture files are obtained by running a packet analyzer in the controller during the experiment, such as *tcpdump* or the Spirent C1's capture tool. The second approach relies on reports that are generated by the controller modules. Finally, the combination of wiretap devices and the DAG card offers the capability to capture and analyze packet timestamps at nanosecond precision.

[7] `http://www.spirent.com/Test-solutions_datasheets/Broadband/PAB/Spirent_TestCenter/STC_C1-Appliance_Datasheet`, Spirent TestCenter Application v4.69.986

[8] `https://iperf.fr/`

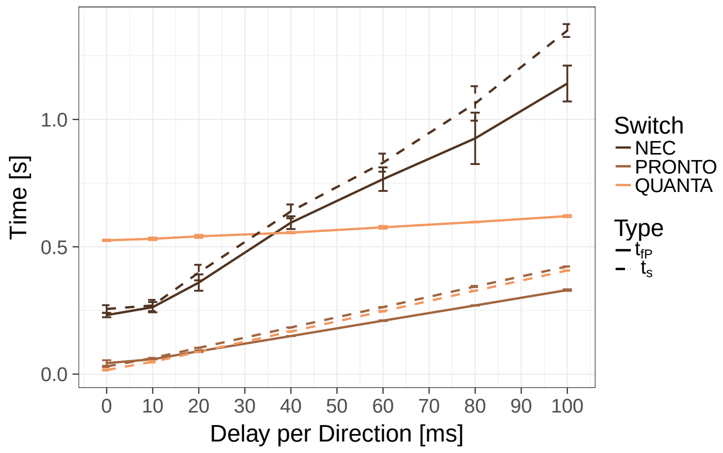
4 Results and Discussion

In this section, we present the results of the experiments that are described in Section 3. First, we demonstrate the heterogeneity of the switch hardware. This is achieved by comparing the FlowMod processing times of different switches when installing different numbers of flows and applying different amounts of control plane delay. Afterwards, we show that using prior information on the hardware specific characteristics and controller-based delay measurements, it is possible to achieve a high degree of correlation between the flow setup time, t_s, and the time until flow rules are active in the data plane, t_{fP}. This outcome highlights that reliable estimations of t_{fP} are possible at run time. Finally, we present results regarding the impact of the controller implementation on the FlowMod processing time.

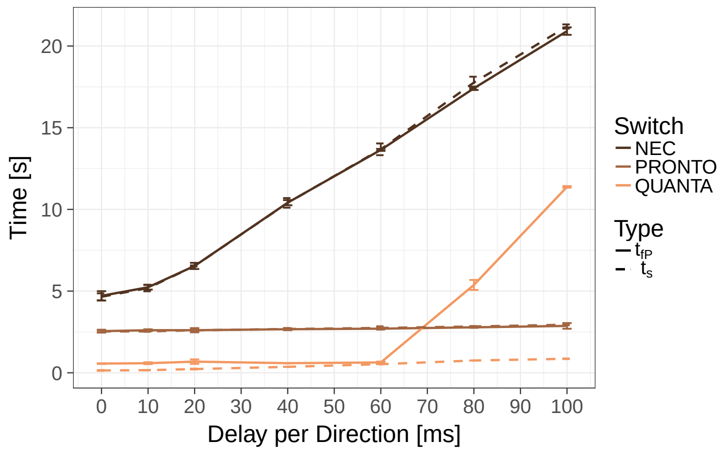
4.1 Impact of Switch Hardware

The two graphs of Figure 3 highlight the individual behavior of the three switches that are used in this work with respect to their sensitivity to parameters such as the amount of control plane delay and the number of flows that are installed. Their x-axes represent the control plane delay that is set in each direction between switch and controller, i.e., a value of 10 ms corresponds to a round trip time of 20 ms. The y-axes denote the flow setup times t_s and t_{fP} that are recorded by means of the wiretap devices and are represented by dashed and solid lines, respectively. Finally, differently colored curves correspond to different switches. For each parameter combination, five experiment runs are performed in order to construct 95 % confidence intervals that are indicated with error bars. The results in the figures are based on measurements with the OpenDaylight controller. Experiments with the other two controllers yield qualitatively similar results and are discussed in Section 4.2.

Figure 3a displays results from experiments in which a total of 100 FlowMod messages are sent to the switch via the *addFlowAsync* mechanism, i.e., 100 FlowMods are followed by one pair of BarrierRequest and BarrierReply messages. Three observations can be made. First, the three switches operate at different time scales. With processing times that are lower than 500 ms for all delay values, the Pronto switch consistently outperforms the other two switches in this scenario. Second, the sensitivity of the switches towards the control plane delay varies significantly, as indicated by the different slopes of the individual curves. Consequently, the NEC switch achieves lower values of t_{fP} than Quanta in scenarios with a low delay, whereas the Quanta switch is least affected by the increasing delay and gives better results for delays that are larger than 40 ms. Third, while the NEC and Pronto switch send their barrier reply after having installed all flow rules into the data plane, i.e., $t_s > t_{fP}$, the Quanta switch sends out the confirmation before the rules are active. Hence, a window of inconsistency of up to half a second can occur if the controller is unaware of this behavior.



(a) 100 flows.



(b) 1800 flows.

Fig. 3. Influence of the control plane delay on the FlowMod processing time when using different switches and different numbers of flows. Scenario details: OpenDaylight controller and *addFlowAsync* mechanism.

Increasing the number of installed flows to 1800 exposes additional differences between the switches. The corresponding results are shown in Figure 3b. For all switches, the increased number of FlowMod messages that need to be processed results in larger setup times. Furthermore, the high delay sensitivity of the NEC switch is even more pronounced in this scenario, with setup times ranging from 5 to over 20 seconds. In the case of the Quanta switch, a significant increase of the installation time is observed for delay values larger than 60 ms. Combined with the premature barrier reply message, this can be a major threat to state consistency. Only the Pronto switch is able to maintain nearly constant t_s and t_{fP} values for all delay settings.

While the wiretap-based measurements that are presented in the previous figures demonstrate the differences between the hardware switches, there is a high pairwise similarity between t_s and t_{fP} values. We use this relationship to derive a mechanism that can be used to infer t_{fP} from information regarding the particular switch model that is in use and measurements at the controller. These measurements include tcpdump on the controller machine to obtain t_s and a simple round trip time measurement like ping to determine the control plane delay.

For each of the three switches, the graphs in Figure 4 show the measurement of the flow setup time t_s at the controller on the x-axis and the actual time until the first data plane packet t_{fP} at the wiretap on the y-axis. Differently colored dots denote different control plane delays.

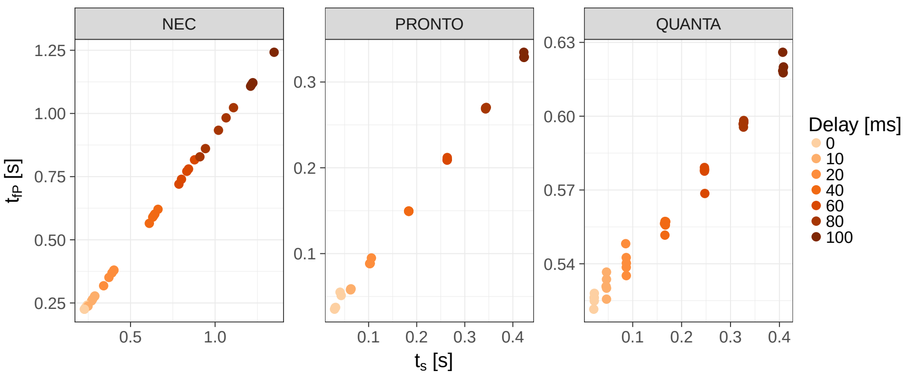
In Figure 4a, results that are obtained when installing 100 flow rules are displayed. Although the times that are recorded for the three switches have significantly different ranges, a high linear correlation between t_s and t_{fP} can be observed. Hence, using switch-specific information regarding its sensitivity towards control plane delay in conjunction with round trip time and t_s measurements is sufficient for an accurate estimation of the flow installation time in the data plane.

Similar results are obtained in case of the installation of 1800 flow rules, as presented in Figure 4b. While the NEC switch has the highest setup and processing times, it also has the most consistent behavior and an almost perfect linear correlation. Except for few outliers, the Pronto switch also shows a high degree of correlation, even with the increased number of flows. Finally, the Quanta switch produces outliers for high control plane delays. Nevertheless, this behavior is observed consistently - qualitatively as well as quantitatively - in multiple repetitions of our experiments, as indicated by clusters of dots in the scatter plot. Therefore, this switch-specific characteristic can also be taken into account by the controller when making predictions regarding the data plane state.

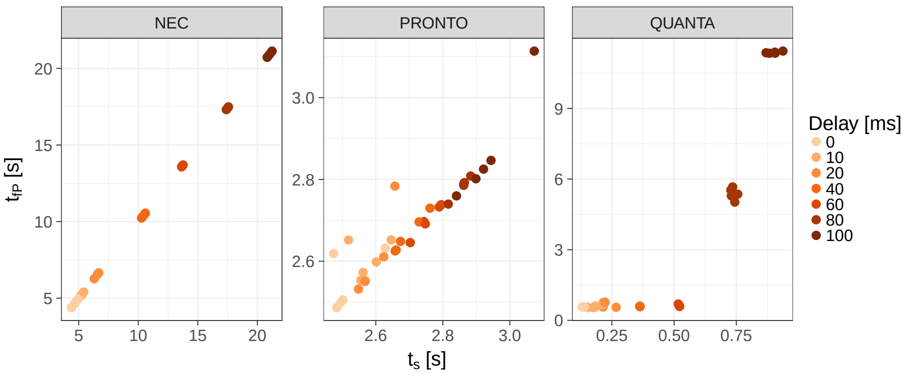
Summarizing, our findings show that using simple controller-based measurements in combination with switch properties that can be determined prior to deployment can be used for performing accurate prediction of the FlowMod installation time in the data plane of OpenFlow switches.

4.2 Impact of Controller Choice

While the previously shown results focus on the peculiarities of different data plane hardware, this section is devoted to the influence of the controller implementation on the performance. To this end, experiments with the NEC switch are conducted with three different controllers. These include the Java-based OpenDaylight controller, the Python-based Ryu controller, as well as the controller implementation that is provided by the OpenFlow Testing Package of the Spirent C1. In case of the OpenDaylight and Ryu controller, the same host machine is



(a) 100 flows.



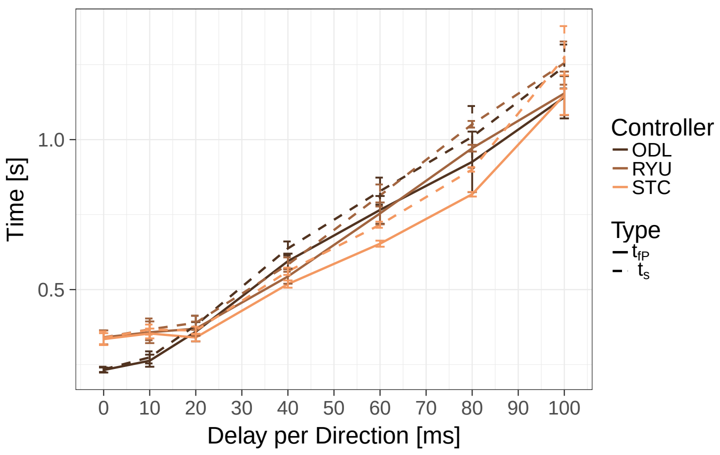
(b) 1800 flows.

Fig. 4. Flow setup time t_s recorded at the controller and time to first data plane packet t_{fP} recorded via the wiretap for the three different switches. Scenario details: OpenDaylight controller and *addFlowAsync* mechanism.

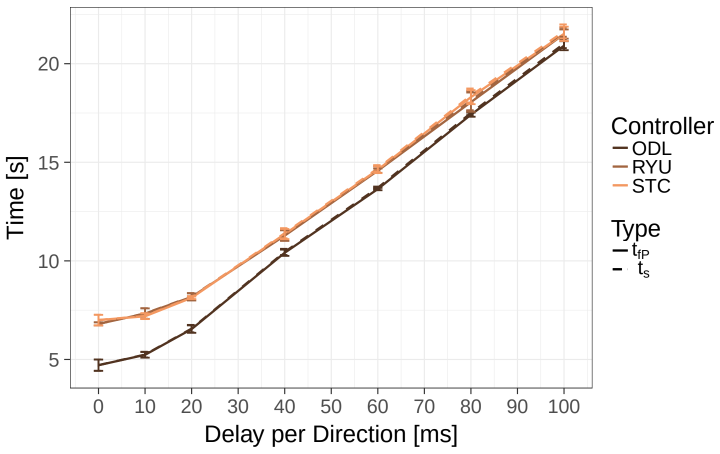
used to ensure that the results are not affected by a heterogeneity of the underlying hardware. The graphics in Figure 5 show t_s and t_{fP} values for different numbers of flows on the y-axis and the control plane delay on the x-axis. Differently colored lines correspond to the three controllers.

When 100 flows are installed, the majority of confidence intervals in Figure 5a overlap. This indicates that for control plane delays that are larger than 10 ms, no statistically significant difference between the three controllers can be identified. In the case of control plane delays that are lower than 20 ms, using the OpenDaylight controller leads to setup times of roughly 0.23 seconds as opposed to setup times of roughly 0.34 seconds that are observed for Ryu and the Spirent-based controller.

These phenomena are even more pronounced in the case of 1800 flows. Figure 5b shows that using the OpenDaylight controller leads to consistently faster



(a) 100 flows.



(b) 1800 flows.

Fig. 5. Impact of the controller choice on flow setup times for different numbers of installed flows. Configuration details: NEC switch and *addFlowAsync* mechanism.

flow setup times than Ryu and Spirent. Differences between 1 and 2 seconds are observed for setup times that range between 4.7 and 21.6 seconds.

The aforementioned results indicate that the controller is not merely a generator of FlowMod messages but can also affect the performance. In-depth analyses of the corresponding packet dumps show that the sending behavior of the OpenDaylight controller and the corresponding packetization of OpenFlow messages differs from the other two controllers. Hence, controller developers should be aware of such mechanisms in order to adapt to switch capabilities and opportunities to improve the overall performance.

5 Conclusion

In this work, we investigate the influence of control plane delay on the performance of OpenFlow switches in terms of their FlowMod processing time. Our testbed setup features hardware from NEC, Quanta, and Pronto as well as three different SDN controller implementations. These include the Java-based OpenDaylight controller, the Python-based Ryu controller, and the controller implementation that is available in the OpenFlow Testing Package of the Spirent C1 platform. Additionally, we use wiretap devices in order to obtain highly precise measurements.

The contribution of this work is threefold. Firstly, we confirm the heterogeneity of OpenFlow switching hardware. This includes not only varying processing times but also different degrees of sensitivity towards the control plane delay between controller and switch. Secondly, we demonstrate that switch-specific characteristics that can be extracted prior to deployment can be used in conjunction with simple measurements at the controller in order to accurately predict the data plane state and performance of switches. Such a prediction mechanism can significantly reduce the window of inconsistency between an SDN controller and the switches it manages. Finally, we show that implementation details of the SDN controller can also have an impact on the overall FlowMod processing performance due to sender-side behavior. This leads to optimization potential that can be taken into account by both, controller developers who want to improve the general performance of their controller as well as network operators who want to maximize compatibility and reliability of the components in their particular network.

Several directions for future work are available. On the one hand, the impact of dynamically fluctuating control plane delays can be analyzed. Depending on the amount and frequency of the fluctuation, the measurement frequency at the controller needs to be adapted. On the other hand, more in-depth analyses of the packet dumps might reveal different classes of switch-side behavior that can be used to infer more generic and robust models.

Acknowledgments

This work has been performed in the framework of the CELTIC EUREKA project SENDATE-PLANETS (Project ID C2015/3-1), and it is partly funded by the German BMBF (Project ID 16KIS0474). The authors alone are responsible for the content of the paper.

References

1. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM CCR*, 2008.
2. M. Jarschel, T. Zinner, T. Hossfeld, P. Tran-Gia, and W. Kellerer, "Interfaces, Attributes, and Use Cases: A Compass for SDN," *Communications Magazine, IEEE*, 2014.
3. A. Nguyen-Ngoc, S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, and M. Jarschel, "Performance evaluation mechanisms for flowmod message processing in openflow switches," in *Communications and Electronics (ICCE), 2016 IEEE Sixth International Conference on*, 2016.
4. M. Kuźniar, M. Canini, and D. Kostić, "OFTEN testing OpenFlow networks," in *European Workshop on Software Defined Networking (EWSDN)*, 2012.
5. "OFTest – Validating OpenFlow Switches," Big Switch Networks. [Online]. Available: http://www.projectfloodlight.org/oftest/
6. A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "Openflow switching: Data plane performance," in *IEEE International Conference on Communications (ICC)*, 2010.
7. P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle, "Performance characteristics of virtual switching," in *IEEE 3rd International Conference on Cloud Networking (CloudNet)*, 2014.
8. 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*, 2012.
9. C. Rotsos, G. Antichi, M. Bruyere, P. Owezarski, and A. Moore, "OFLOPS-Turbo: Testing the Next-Generation OpenFlow switch," in *European Workshop on Software Defined Networks (EWSDN)*, 2014.
10. M. Shahbaz, G. Antichi, Y. Geng, N. Zilberman, A. Covington, M. Bruyere, N. Feamster, N. McKeown, B. Felderman, M. Blott *et al.*, "Architecture for an open source network tester," in *Proceedings of the ninth ACM/IEEE symposium on Architectures for networking and communications systems*, 2013.
11. 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*, 2011.
12. S. Azodolmolky, R. Nejabati, M. Pazouki, P. Wieder, R. Yahyapour, and D. Simeonidou, "An analytical model for software defined networking: A network calculus-based approach," in *IEEE Global Communications Conference (GLOBECOM)*, 2013.
13. M. Jarschel, T. Zinner, T. Höhn, and P. Tran-Gia, "On the Accuracy of Leveraging SDN for Passive Network Measurements," in *Australasian Telecommunication Networks & Applications Conference (ATNAC)*, 2013.
14. M. Kuźniar, P. Perešíni, and D. Kostić, "What you need to know about sdn flow tables," in *International Conference on Passive and Active Network Measurement*, 2015.
15. D. Y. Huang, K. Yocum, and A. C. Snoeren, "High-fidelity switch models for software-defined network emulation," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013.
16. S. Geissler, S. Herrnleben, R. Bauer, S. Gebert, T. Zinner, and M. Jarschel, "Tablevisor 2.0: Towards full-featured, scalable and hardware-independent multi table processing," in *Network Softwarization (NetSoft), 2017 IEEE Conference on*, 2017.

17. H. Pan, G. Xie, Z. Li, P. He, and L. Mathy, "FlowConvertor: Enabling Portability of SDN Applications," in *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*, 2017.
18. P. Zhang, H. Li, C. Hu, L. Hu, L. Xiong, R. Wang, and Y. Zhang, "Mind the gap: Monitoring the control-data plane consistency in software defined networks," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, 2016.