Modeling and Performance Evaluation of an OpenFlow Architecture

Michael Jarschel, Simon Oechsner, Daniel Schlosser, Rastin Pries, Sebastian Goll, Phuoc Tran-Gia
University of Würzburg, Institute of Computer Science, Würzburg, Germany.
Email: {michael.jarschel,oechsner,schlosser,pries,goll,trangia}@informatik.uni-wuerzburg.de

***Abstract*—The OpenFlow concept of flow-based forwarding and separation of the control plane from the data plane provides a new flexibility in network innovation. While initially used solely in the research domain, OpenFlow is now finding its way into commercial applications. However, this creates new challenges, as questions of OpenFlow scalability and performance have not yet been answered. This paper is a first step towards that goal. Based on measurements of switching times of current OpenFlow hardware, we derive a basic model for the forwarding speed and blocking probability of an OpenFlow switch combined with an OpenFlow controller and validate it using a simulation. This model can be used to estimate the packet sojourn time and the probability of lost packets in such a system and can give hints to developers and researchers on questions how an OpenFlow architecture will perform given certain parameters.**

***Index Terms*—OpenFlow, switch, performance evaluation**

I. Introduction

In the discussion about the emerging Future Internet, OpenFlow is currently seen as one of the promising approaches that may pave the way towards that goal. OpenFlow was first proposed in [1] as a way to enable researchers to conduct experiments in production networks. However, its advantages may lead to its use beyond research, e.g. in the context of network virtualization. At its core, OpenFlow offers a higher flexibility in the routing of network flows and the freedom to change the behavior of a part of the network without influencing other traffic. It achieves this by separating the control plane in network switches from the data plane, and allowing for a separate controller entity that may change the forwarding rules in modern switches. This enables the implementation of, e.g., virtual networks, user mobility, or new network and transport layer protocols.

Generally, only software changes should be necessary to enable OpenFlow on state-of-the-art Ethernet switches. It is also not necessary for hardware vendors to open up their systems to support OpenFlow, since only a quite well defined interface has to be provided to the control plane. Consequently, an OpenFlow extension can be introduced as firmware for existing hardware, significantly lowering the entry barriers for new technologies. Therefore, it is believed that OpenFlow may circumvent the technological inflexibility, also known as the 'Internet ossification'. New technologies or protocols that are tested using OpenFlow-enabled hardware in large-scale networks can have a significantly reduced development time as performance data can be gathered under realistic circumstances instead of an isolated test environment.

This work was partially funded by the Federal Ministry of Education and Research of the Federal Republic of Germany (Förderkennzeichen 01BK0917, GLab). The authors alone are responsible for the content of the paper.

Currently, first OpenFlow implementations from hardware vendors are available and being deployed in networks. As a result, we can expect a growing number of works conducting experiments in OpenFlow-enabled networks. Here, OpenFlow serves as the basis for the evaluation of new virtualization techniques or new routing protocols. However, the basic technology itself, i.e., the use of an OpenFlow controller to modify the flow table of an Ethernet router via a secure channel, is still new and few performance evaluations of the OpenFlow architecture exist.

Understanding the performance and limitations of the basic OpenFlow concept is a prerequisite for using it for experiments with new protocols and mechanisms. Therefore, we aim to provide a performance model of an OpenFlow system with this paper. The model is based on results from queuing theory and is verified by simulations and measurement experiments with a real OpenFlow switch and controller. The advantage of this preliminary analytical model over the simulation is the fact that it can provide results in a few seconds' time whereas the simulation may require several hours to complete depending on the computing hardware. Additionally, the M/M/1-S feedback queue is already a good approximation of the actual controller performance. The model captures the delay experienced by packets that have to be processed by the controller in contrast to be processed just by the switch, as well as the probability to drop packets if the controller is under high load. Using this model, we derive conclusions about the importance of the performance of the OpenFlow controller in different realistic scenarios, and its effect on the traffic flowing through the OpenFlow-enabled switch.

The remainder of the paper is structured as follows. In Section II, we explain the OpenFlow architecture in more detail and give an overview on related work. Section III holds details about the setup of the measurements conducted. Our model is introduced in Section IV. The results from the performance evaluation are described in Section V. We conclude the paper by summarizing the main contributions in Section VI.

II. Background and Related Work

To better understand the model for the OpenFlow performance evaluation, we first give a brief overview of OpenFlow. More details on OpenFlow can be found in the white paper [1] as well as in the OpenFlow specification [2].

OpenFlow was designed as a new network paradigm, which enables researchers to test new ideas under realistic conditions on an existing network infrastructure. To be able to take action in the switching, OpenFlow separates the control plane from the data plane and connects them by an open interface, the OpenFlow protocol. The control plane is implemented in software in form of a controller on an external PC. For the communication between the switch and the controller, a secure channel is used. This allows researchers to be flexible with their work, while at the same time using high-performance hardware.

The OpenFlow switch itself holds a flow table which stores flow entries consisting of three components. First, a set of 12 fields with information found in a packet header that is used to match incoming packets. Second, a list of actions that dictates how to handle matched packets. Third, a collection of statistics for the particular flow, like number of bytes, number of packets, and the time passed since the last match.

When a packet arrives at the OpenFlow switch, its header information is extracted and then matched against the header portion of the flow table entries. If checking against entries in each of the switches' tables does not result in a match, the packet is forwarded to the controller, which determines how the packet should be handled. In the case of a match, the switch applies the appropriate actions to the packet and updates statistics for the flow table entry. This process is visualized in Figure 1. In this paper, we want to analyze the influence of the switch-controller interaction on the normal switching performance. Before, we provide an overview on the literature related to this paper.

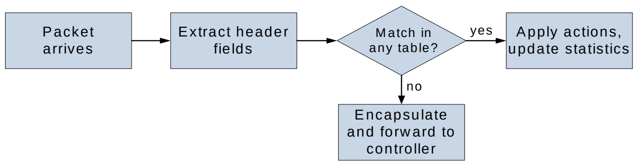
Packet arrives
Extract header fields
Match in any table?
yes
Apply actions, update statistics
no
Encapsulate and forward to controller

Fig. 1. Handling of incoming packets in an OpenFlow switch.

Recently, several papers have been published indicating possible uses for OpenFlow [3]–[6]. All these papers demonstrate that the concept of splitting the control plane from the data plane is useful in a variety of fields, like data center routing, energy saving, and network virtualization. However, none of these papers addresses performance issues of the OpenFlow concept.

Tanyingyong et al. [7] and Luo et al. [8] introduce mechanisms to increase the performance of OpenFlow. They propose an architectural design to improve the lookup performance of OpenFlow switching in Linux using a standard commodity network interface card. They show a packet switching throughput increase of up to 25 percent compared to the throughput of regular software-based OpenFlow switching. Luo et al. instead, apply network processor based acceleration cards to perform OpenFlow switching. They show a 20 percent reduction on packet delay compared to conventional designs.

Currently, only one paper focuses on performance measurements of OpenFlow [9]. In the paper, the OpenFlow switch performance is measured for different types of rules and packet sizes. However, the performance of the switch-controller interaction is not taken into account.

In contrast to the above mentioned papers, we did not only measure the performance of the switch-controller interaction but also provide a simple performance model of an OpenFlow system.

III. OpenFlow Measurements

Before introducing the analytical model in Section IV, we first derive the performance parameters. Therefore, we measured the forwarding performance of various OpenFlow implementations.

A. Measurements

In order to measure the parameters we need for the analytical model and the simulation, we set up an OpenFlow test bed as depicted in Figure 2. We connect several servers as load generators to a typical gigabit switch. It has to be noted that this switch is only used to multiplex the traffic of several traffic generators to increase the bandwidth being produced using small packet sizes. This switch has been tested to work with maximum link rate and does not limit the transmission speed between the traffic generators and the traffic sink.

The packets leaving this switch towards the OpenFlow switch are cloned by a wire tap. This device ensures that packets arriving at the wire tap are forwarded to the OpenFlow switch and the measurement server at exactly the same time. Next, the packets are processed by the OpenFlow switch. For

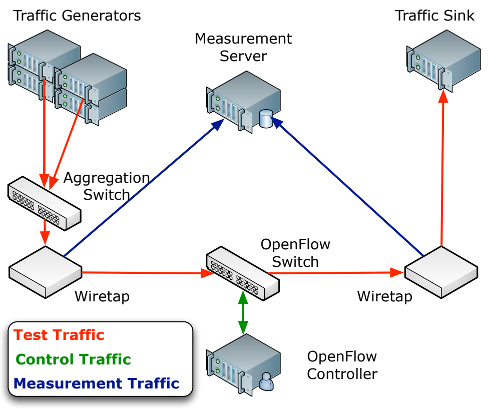

Fig. 2. Test bed set up to measure model parameters.

our experiments, we used three different OpenFlow implementations, a Pronto 3290 24 port gigabit switch, a NetFPGA OpenFlow switch as well as the Open vSwitch software switch.

If the switch contains a rule matching the packet received, the forwarding decision is instantaneously executed on the switch. If there is no matching rule in the switch, the switch either extracts some parts of the packet or sends the complete packet to the controller requesting for an action to execute. The controller will answer to the switch's request with an action to perform on all packets of this flow. Furthermore, the packet triggering the request is forwarded by the switch or the controller itself, depending on the implementation and the cabling of the controller.

After leaving the switch, each packet is replicated by another wire tap to the measurement server as well as to the traffic sink. Replicating the packet before and after the OpenFlow switch makes it possible to precisely monitor the sojourn time. We use an Endace DAG card in our measurement server to keep the measurement error below nine nanoseconds [10].

In order to measure the time the OpenFlow switch needs to forward a packet without controller interaction, we send packet bursts of two million identical packets through the OpenFlow switch. A rule matching these packets is preinstalled into the switch so that no controller access is involved. We measured the forwarding time for different packet sizes between 64 byte and 1514 byte Ethernet frame length and estimated the sojourn time μ_S based on the results. Figure 3 shows the measured forwarding delays in microseconds for the Open vSwitch software switch, the Pronto 3290 switch, and the NetFPGA OpenFlow switch depending on the packet payload. The error bars indicate the standard deviation from the measured mean over 10 identically performed measurements for each payload size. The y-axis is in logarithmic scale to be able to display the curves for all switches in one figure. We observe an almost linear increase of the mean forwarding delay from about $4\mu s$ to about $16.5\mu s$ with the increase in payload size for both hardware-based implementations with the NetFPGA

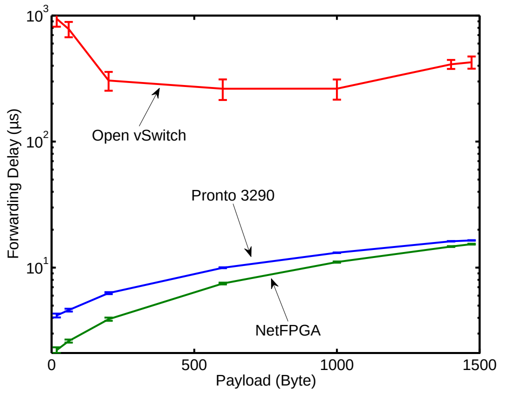

Fig. 3. Measured switch delay.

performing slightly better. The Open vSwitch as a software implementation is two orders of magnitude slower and suffers from frequent memory accesses especially at small packet sizes. From these results we decided to use the performance values of the Pronto 3290 as input parameters for our model as the difference between Pronto and NetFPGA is marginal and the technical specifications of the Pronto are much closer to those of commercial hardware switches than those of the NetFPGA.

The measurements which provided the data for the response times of the OpenFlow controller Nox 0.9 were performed in a different scenario. We installed the Cbench [11] tool on the measurement server and attached the controller directly. The Cbench tool measures the rate in which flow requests are handled by the controller. Unfortunately, we were not able to match requests and responses to the OpenFlow controller. Hence, we need to rely on the number of answers per second the Cbench tool measured. These showed a mean value of 4175 responses per second with a standard deviation of 101.43. From this value, we calculated the mean values of the controllers sojourn times and use these in our analytical model and in the simulation.

Finally, we also require the inter-arrival times of new packets and flows. These are based on the measurements published in [12], where we analyzed the traffic of a residential wireless Internet access for over 30 days. The packet size distribution and the probability of new flows used in our performance evaluation have been extracted from this study.

IV. A Simple Model of an OpenFlow Architecture

We abstract the OpenFlow architecture as a feedback-oriented queuing system model, divided into a forward queuing system of the type $M/GI/1$ and a feedback queuing system of the delay-loss type $M/GI/1-S$. We deliberately start by assuming Markovian servers for both systems, i.e. an $M/M/1$ for the forward model and an $M/M/1-S$ for the feedback model, to test the robustness of the modeling approach. The forward queue has an average service time of 9.8 microseconds. The queue size of the forward system is assumed to be infinite. In contrast, the buffer for the packets waiting on a controller response is assumed to have a finite capacity of 512, which models the queue of the feedback system. The arrival process at the switch, i.e. of the forward system, is a combination of the arrival process of packets received from the line cards with rate λ and of packets being forwarded from the switch buffer after the controller has determined the appropriate action and the corresponding entry in the flow table was created.

The OpenFlow controller is thus modeled by the feedback $M/M/1-S$ queuing system with an exponential service time with a mean value $E[B_C] \in \{31\mu s, 240\mu s, 5.2ms\}$. The high as well as the low mean service time were chosen arbitrarily one order of magnitude larger respectively smaller than the measured mean service time of $240\mu s$. We assume that all merging traffic flows again form Poisson streams. The smallest

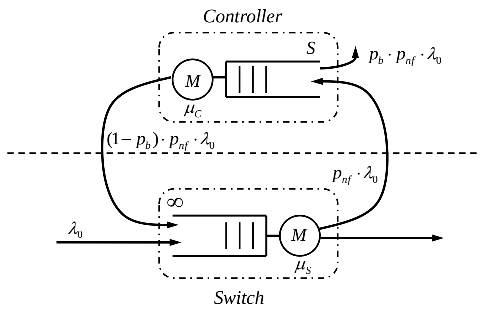

Fig. 4. A simple model of an OpenFlow switch.

value for $E[B_C]$ is taken from our controller benchmark, the other values have been chosen arbitrarily to reflect the impact of controller applications one or two order of magnitudes slower. The queue length S of the controller system is limited in order to model the possibility of dropped packets under high load conditions. The arrival rate in this system is a fraction of the arrival rate of packets from line cards in the switch, governed by the probability p_{nf} of a flow being seen by the switch which has no flow table entry yet.

The two main performance indicators of interest for an evaluation are the total sojourn time of a packet through the system and the probability of dropping a packet. A packet has to traverse the switch system at least once. With a probability of p_{nf}, the switch has no entry in its flow table for that packet and forwards it to the controller. A packet can be blocked at the controller with probability p_b. After the controller sojourn time, it is again queued in the switch and traverses it for a second time. The complete model of the forward and feedback queuing systems with both components and all traffic flows is shown in Figure 4.

It is important to note that a single packet cannot be forwarded to the controller twice, i.e., p_{nf} is only applied to the initial packet flow with rate λ_0. In our analysis, we ensure that a packet does not experience more than the sojourn time of the controller plus twice the sojourn time of the switch. Figure 5 illustrates the way a packet takes through the system in a more sequential manner. W_S and W_C represent the waiting queues at the switch and the controller respectively, whereas B_S and B_C represent the corresponding service units.

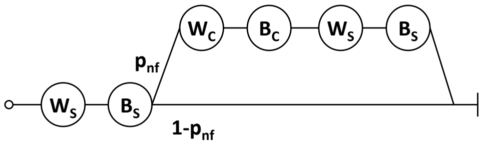

Fig. 5. Phase diagram of the packet sojourn time.

A. Assumptions

In the forward queuing system, we take the simplifying assumption that the overall arrival process at the switch (forward), as well as the arrival process at the controller (feedback) are Poisson. This can be justified as the state processes on the forward and the feedback paths are on very different time scales, which allows the decomposition of the two queuing systems. Moreover, we queue all packet arrivals in a single queue at the switch, instead of a separate queue per line card. The feedback queue used to model the controller actually comprises out of the line-out card of the switch towards the controller, the buffer, and processing at the controller itself. However, the Nox controller we used as a reference for this model controls the traffic rate it receives from the switch in order to prevent overload. Therefore, no additional queuing happens at the controller itself, and the line-out buffer at the switch is the only place where packet loss may happen. The transmission time of packets from the switch to the controller is encapsulated in the service time of the controller.

B. Limitations of the Model

In its current form, the model does not capture the fact that incoming traffic at the switch is queued first per line card, i.e., one queue per port. As well, it is currently limited to a single switch per controller, whereas OpenFlow allows the same controller to be responsible for a number of different switches. Furthermore, the model assumes TCP traffic as opposed to UDP traffic. For TCP traffic only the first packet header of a new flow is sent to the controller, while for UDP all packets are relayed until a flow rule is in place. These limitations will be addressed in future work by refining the model described here. Refinements may contain replacing the forward input queue by a polling system or replacing the $M/M/1-S$ system of the controller with a more general $M/GI/1-S$ system that allows the use of a measured service time distribution as an input for the model.

Although it might be desirable to have the complete distribution for the model, we focus on the first two moments only as they provide us the most important performance indicators required by a service provider for dimensioning the network.

V. Performance Evaluation

In this section, we discuss the output of our model and validate them by means of a simulation. We use a set of scenarios depicting different use-cases for an OpenFlow-enabled switch. This is mainly reflected in the forwarding probability p_{nf}. A value of $p_{nf} = 0.04$ represents a normal productive network carrying end user traffic, where [12] showed that this is the probability for new flows being observed at the switch. Values of $p_{nf} = 0.2$ or 0.5 model a network where a part of the traffic is routed via the controller, e.g., if a portion of the traffic is using a virtualized network. Finally, $p_{nf} = 1$ depicts the case where the controller handles the complete traffic going through the switch, e.g., in an experiment testing new protocols, such as described in [1].

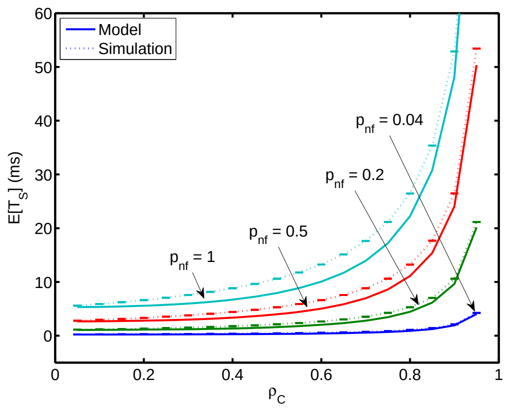

Fig. 6. Impact of the forwarding probability on the mean packet sojourn time for $E[B_C] = 5.3ms$

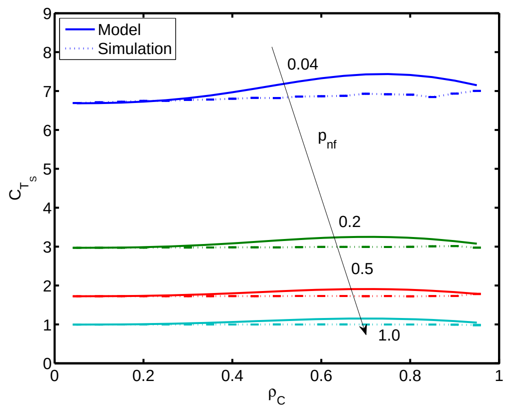

Fig. 7. Impact of the forwarding probability on the coefficient of variation for $E[B_C] = 5.3ms$.

We also vary the mean service time of the controller. The slowest value is two orders of magnitude larger than the service time of the switch, which may be the case if commodity hardware is used to run the controller. We provide results for faster controller systems as well to allow to predict the system behavior if dedicated hardware is used. The buffer size at the controller queue is set to $S = 512$ packets in all experiments, which was chosen arbitrarily as a middle ground between experimental and commercial switches.

To be able to validate the results from the analytical model, we implemented a packet based simulation in OMNeT++. The simulation model also reflects the structure of the analytical model, cf. Figure 4. Verification through simulation was chosen over measurements as simulations results are faster to obtain and do not require as many repetitions to stabilize. The process time of the controller in the simulation uses the same distribution as the analytical model. In contrast, we are able to study the feedback generated by packets looping back over the controller in the simulation, which we are not able to fully reflect in the analytical model. The assumptions about the arrival and departure process of the controller are also relaxed in the simulation, i.e., no Poisson process is assumed. Instead, the actual inter-departure times of packets after traversing the switch and the controller, respectively, are used.

The simulation results shown in this section are based on six simulation runs per parameter and different seeds. The 95 percent confidence intervals are given for the simulation results in all figures, but are only represented by dashes as they are very small compared to the scale.

Figure 6 illustrates the modeled as well as the simulated mean sojourn time of a packet depending on the controller load for an expected controller service time value of $E[B_C] = 5.3ms$. Graphs are shown for several p_{nf}, i.e., the probability that a received packet at the switch represents a new flow and subsequently causes the switch to send an OpenFlow packet, which needs to be answered by the controller. Values are given for controller loads from 5 to 95 percent in 5 percent steps.

As a first general observation, we see an increase in the mean sojourn time which is caused by the influence of the controller. The more packets need to wait for a controller response, the more packets have a longer sojourn time caused by the controller service time and additional waiting time imposed by queuing. Since the controller reaches a high utilization sooner than the switch with an increasing λ_0, i.e., an increasing raw traffic rate, it contributes a much longer waiting time to the total packet sojourn time.

In case of $p_{nf} = 0.04$, the mean sojourn time of the system barely deviates from that of the switch, since only a small fraction of traffic has to be handled by the controller. Only at a controller load of 75 percent and above we observe the start of an exponentially rising gradient. If we increase p_{nf} to 0.2, we see this increase at a controller load of about 45 percent. For $p_{nf} = 0.5$ it already starts at a load of 30 percent and at the maximum value $p_{nf} = 1.0$ we detect the increase already at 10 percent controller load. In all cases, the simulation curve is located slightly above the curve for the model. For low values of p_{nf} this deviation appears to be marginal. Here, simulation and model show a nearly identical progression. With an increasing p_{nf}, we see a deviation increase except for very high controller loads. However, the deviation remains small.

In Figure 7, the coefficients of variation for the sojourn times are shown also dependent on the controller load. We observe an increasing coefficient of variation with a decreasing p_{nf}. This is caused by the fact that with a smaller p_{nf} less packets are subject to the delay imposed by the controller and therefore, the deviation from the mean value for these packets is much higher. Again, we can also see a small discrepancy between simulation and model for medium controller loads. This discrepancy increases with smaller values for p_{nf} complementary to what we see in Figure 6. However, simulation and model seem to be a good fit.

Figure 8 depicts the simulated as well as modeled mean sojourn times dependent on the controller load for an expected

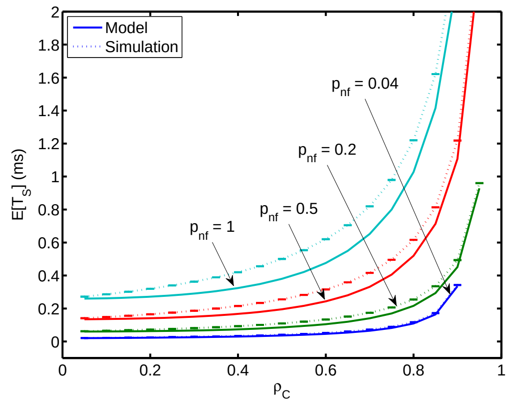

Fig. 8. Impact of the forwarding probability on the mean packet sojourn time for $E[B_C] = 240\mu s$.

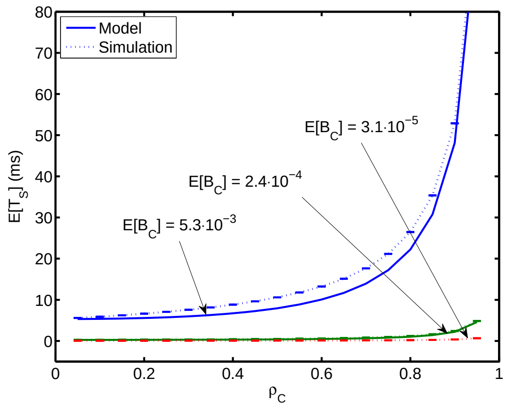

Fig. 10. Impact of the controller service time on the mean packet sojourn time.

controller service time value of $E[B_C] = 240\mu s$, corresponding to Figure 6. For $p_{nf} = 0.04$ and a controller load of 95 percent no value is given as this would violate our assumption of an offer $a \leq 1$. Overall, we observe the same effects as discussed for Figure 6 albeit for mean sojourn times two orders of magnitude smaller. The point where the gradient starts to increase exponentially is shifted to lower controller loads, e.g., for $p_{nf} = 0.04$ we see an increase already at 45 percent load as opposed to 75 percent in Figure 6.

Corresponding to Figure 7, Figure 9 shows the coefficients of variation for the sojourn times for $E[B_C] = 240\mu s$. While the progressions of the coefficients are very similar here to those in Figure 7 for $p_{nf} = 0.5$ and $p_{nf} = 1.0$, they differ for $p_{nf} = 0.2$ and $p_{nf} = 0.04$. Contrary to Figure 7, the model curve for these two values of p_{nf} now underestimates the simulation for small controller loads. We also note an increasing gradient for $p_{nf} = 0.2$. For $p_{nf} = 0.04$ we see an exponential decrease in the coefficient of variation at controller load above 70 percent. Furthermore, the model curve now underestimates the simulation for all observed controller loads. In this scenario, not only the delay imposed by the controller is relevant, but we also observe a non-zero waiting time at the switch. This is caused by the now much smaller difference between the service time of the switch and that of the controller. A high utilization of the controller also leads to a non-negligible utilization of the switch, resulting in a non-empty queue.

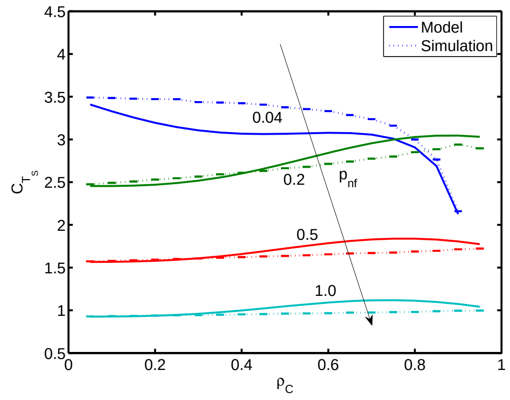

Fig. 9. Impact of the forwarding probability on the coefficient of variation for $E[B_C] = 240\mu s$.

Finally, Figure 10 displays the simulated as well as modeled mean sojourn times dependent on the controller service time for $p_{nf} = 1.0$. We can observe the influence of the controller performance relative to the switch performance on the total system. As we have seen in Figure 6 and Figure 8, the mean sojourn time increases exponentially with increasing load. However, for an $E[B_C] = 5.3ms$ the gradient already shows a much higher increase at smaller controller loads than that for $E[B_C] = 240\mu s$. With an $E[B_C] = 31\mu s$, the curve is governed by the switch delay and therefore, the mean sojourn time is barely distinguishable from the service time of the switch and does not show an increase at all.

In all cases, the observed blocking probabilities (p_B) for packets at the controller queue were zero for the simulation and infinitesimal small in the model. This indicates that the OpenFlow architecture is stable for our input values.

VI. Conclusion

In this paper, we proposed a basic model to analyze the forwarding speed and blocking probabilities of an OpenFlow architecture. Blocking can thereby only occur in the forwarding queue to the controller. The results show that the sojourn time mainly depends on the processing speed of the OpenFlow controller. Our measurements have shown that the processing time of the controller lies between 220 μs and 245 μs. The impact of the controller processing time can be best seen in the variation of the sojourn time. The higher the probability of new flows arriving at the switch, the lower is the coefficient of variation, but the longer is the sojourn time.

The presented model helps to see the importance of the controller performance for installing new flows. When using

OpenFlow in high speed networks with 10 Gbps links, today's controller implementations are not able to handle the huge number of new flows. Possible directions for future research might consider the performance of an OpenFlow system, where more than one switch is connected to the OpenFlow controller. Here, it is of special interest how the sojourn time of a single switch is influenced when a burst of new flows arrive at different switches. Another possibility would be to integrate the queues of the line cards into the model and to evaluate the impact of different processing strategies in form of a polling system.

Acknowledgments

The authors would gratefully thank Michael Düser and Fritz-Joachim Westphal from Deutsche Telekom Laboratories Berlin for the fruitful discussions and support during the course of this work.

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," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, p. 69, 2008.

[2] "OpenFlow Switch Specification, Version 1.0.0," December 2009.

[3] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "Elastic Tree: Saving Energy in Data Center Networks," in *7th USENIX Symposium on Networked System Design and Implementation (NSDI)*, San Jose, CA, USA, April 2010, pp. 249–264.

[4] M. Casado, D. Erickson, I. A. Ganichev, R. Griffith, B. Heller, N. McKeown, D. Moon, T. Koponen, S. Shenker, and K. Zarifis, "Ripcord: A Modular Platform for Data Center Networking," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-93, June 2010.

[5] S. Das, G. Parulkar, P. Singh, D. Getachew, L. Ong, and N. McKeown, "Packet and Circuit Network Convergence with OpenFlow," in *Optical Fiber Conference (OFC/NFOEC'10)*, San Diego, CA, USA, March 2010.

[6] R. Braga, E. S. Mota, and A. Passito, "Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow," in *35th Annual IEEE Conference on Local Computer Networks*, Denver, CO, USA, October 2010, pp. 416–423.

[7] V. Tanyingyong, M. Hidell, and P. Sjödin, "Improving PC-Based OpenFlow Switching Performance," in *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, New York, NY, USA, 2010, pp. 13:1–13:2. [Online]. Available: http://doi.acm.org/10.1145/1872007.1872023

[8] Y. Luo, P. Cascon, E. Murray, and J. Ortega, "Accelerating OpenFlow Switching With Network Processors," in *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, New York, NY, USA, 2009, pp. 70–71. [Online]. Available: http://doi.acm.org/10.1145/1882486.1882504

[9] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "OpenFlow Switching: Data Plane Performance," in *IEEE ICC*, Cape Town, South Africa, May 2010.

[10] Endace, "DAG 7.5G2 Datasheet."

[11] R. Sherwood and K.-K. Yap, "Cbench Controller Benchmarker," http://www.openflowswitch.org/wk/index.php/Oflops, 2010.

[12] F. Wamser, R. Pries, D. Staehle, K. Heck, and P. Tran-Gia, "Traffic Characterization of a Residential Wireless Internet Access," *Special Issue of the Telecommunication Systems (TS) Journal*, vol. 48: 1-2, May 2010.