

# An Empirical Model of Packet Processing Delay of the Open vSwitch

†Danish Sattar and ‡Ashraf Matrawy

†Department of Systems and Computer Engineering, ‡School of Information Technology

Carleton University, Ottawa, ON Canada

Email: {danish.sattar, ashraf.matrawy}@carleton.ca

**Abstract**—Network virtualization offers flexibility by decoupling virtual network from the underlying physical network. Software-Defined Network (SDN) could utilize the virtual network. For example, in Software-Defined Networks, the entire network can be run on commodity hardware and operating systems that use virtual elements. However, this could present new challenges of data plane performance. In this paper, we present an empirical model of the packet processing delay of a widely used OpenFlow virtual switch, the Open vSwitch. In the empirical model, we analyze the effect of varying Random Access Memory (RAM) and network parameters on the performance of the Open vSwitch. Our empirical model captures the non-network processing delays, which could be used in enhancing the network modeling and simulation.

## I. INTRODUCTION

SDN is a new concept of networking with emphasis on better and easier network management. In traditional networks, each device is fully or partially autonomous in decision making and forwarding of information (packets) [1]. On the contrary, in SDN the control plane is solely responsible for making the forwarding decisions and the data plane only forwards the packets according to the rules set by the control plane. The control and data plane communicate using OpenFlow protocol.

SDN utilizes several virtual network components (e.g. switches, firewalls, intrusion detection/prevention systems, etc.) and one of the widely used virtual network switch used by the SDN is the Open vSwitch (OVS). In SDN, the network manager defines the traffic forwarding rules at the controller. Every new packet follows the following procedure to reach the destination node: Once a packet reaches the `in-port` of OVS, it checks whether it has a flow rule or not. If it has a flow rule, then forward the packet accordingly otherwise pass it to the controller. Next, the controller will check if there is a pre-defined rule for the incoming flow, if there is a rule then forward it using OpenFlow protocol to the OVS otherwise use broadcast to find a route in the network.

There are three types of delays experienced by a packet in the process above; i) controller delay: the time it takes to find an appropriate rule/route, ii) two-way propagation delay between the controller and the requesting OVS and iii) processing delay of the OVS.

We are motivated by the lack of studies, models, and simulators that consider the processing delays that happen inside network elements. In this paper, we are investigating different processing delays experienced by the packet *inside*

the OVS to develop an empirical model for processing delays inside the OVS. The controller and two-way propagation delays are outside the scope of this paper. We have developed an empirical model to characterize these delays. We use commodity hardware to get the measurement in our empirical model. We performed experiments on two platforms; i) virtual and ii) baremetal. We tested using a variable amount of RAM and different network parameters on various hardware configurations. We believe that our empirical model can be beneficial for data plane simulators, and it could provide insight into choosing the appropriate platform (virtual or baremetal) for an application.

We note that while we use different systems in our experimental setup, we are not trying to benchmark or test the performance of these systems or create comparisons with other systems. Rather, we are trying to capture the processing delay behavior of OVS in our experimental setup, and we understand that the behavior might be different in a different setup.

The rest of the paper is organized as follows. In section II existing works are presented. Section III describes the Open vSwitch components and functionality. The empirical model is presented in section IV. Results are discussed in section V, VI and finally section VII concludes this paper.

## II. RELATED WORK

S. Azodolmolky *et al.* [2] developed a network calculus based an analytical model to describe the functionality of Software-Defined Networks. They modeled the behavior of an SDN switch in terms of delay, queue length boundaries, buffer length and controller buffer length. K. Mahmood *et al.* [3] provided SDN modeling based on queuing theory, where a Jackson network was used to model the data plane and  $M/M/1$  queue was used to model the controller. They determined the average time a packet spends in the SDN and the maximum data that can be injected in the network given some delay requirements. Authors built a custom simulator to validate their analytical model. Another queuing theory based model was developed by Xionget *al.* [4] to evaluate the OpenFlow-based software-defined network. To obtain the average time a packet spends in the system, an  $M^X/M/1$  queue was used to model the switch. They evaluated the switch queuing model with different performance parameters using numerical analysis. The  $M/G/1$  queue was used to model the controller

packet-in behavior and it was evaluated using widely used benchmark Cbench under various network scenarios.

U. Javed *et al.* developed a stochastic model for transit latency in SDN [5]. They performed experiments on three different platforms (i.e. Mininet, MikroTik RouterBoard 750GL and GENI) and used the Round Trip Time (RTT) between end hosts as a measurement metric to formulate their model. They also proposed and demonstrated that the log-normal mixture distribution is more suited for transit latency in SDN as compared to  $M/M/1$  models suggested in earlier studies. A hybrid approach was proposed by M. Jarschel *et al.* [6], where they first used hardware switches to measure the average packet forwarding time then selected one of the hardware switch performance values to develop a queuing model to analyze the network. They simplified the OpenFlow architecture queuing system to  $M/M/1$  (forwarding model),  $M/M/1-S$  (controller model) from  $M/GI/1$  and  $M/GI/1-S$ , respectively. To validate the results from analytical model, they used OMNeT++ to implement packet based simulation.

### III. OPEN vSWITCH

Open vSwitch (OVS) is a widely used OpenFlow virtual switch that is flexible and programmable. The OVS can work with or without a controller. In the first case, when there is a controller it uses the control logic to forward the packets. In the latter, if there is no controller present, it forwards the packet according to the layer two logic [7].

There are two major components of the OVS. The first is a userspace module called `ovs-vswitchd`. It is a userspace daemon that is implemented independently of the operating system. The second major component is *kernel datapath module*, which varies operating system to operating system [8].

Figure 1 shows an overview of the working of the OVS. The *datapath module* in the kernel always receives the packets from the Network Interface Card (NIC). Either *kernel datapath module* or `ovs-vswitchd` has the instructions on how to handle this type of packet. In the first case, instructions called *actions* are already cached at the *kernel datapath module*, and it will forward the packets accordingly. In the latter, when the *kernel datapath module* does not have the actions cached, it will pass the packet to the `ovs-vswitchd`. Once a packet reaches the `ovs-vswitchd`, if there is a controller present and `ovs-vswitchd` does not have the instructions on how to handle these type of packets, it will forward it to the controller and wait for a reply. In the second case, when there is no controller present, it will use its internal logic to define actions for the incoming packets. Once the `ovs-vswitchd` has defined the appropriate actions, it will pass them to the *kernel datapath module* and also instruct it to cache them for future use [8].

As mentioned before, the most common use of the OVS is as an SDN switch to control packet forwarding in OpenFlow. The OpenFlow protocol allows a controller to dynamically add, update, remove, obtain statistics on the flow tables and monitor the flows as well as inject, redirect or drop the packets. The `ovs-vswitchd` receives the flow tables from

the SDN controller, matches any incoming packets against these OpenFlow tables, gathers the actions applied, and caches the results in the *kernel datapath module*. It allows the OVS to work independently of any SDN controller as it only needs to understand the OpenFlow protocol. The separation of userspace module and kernel module is transparent to the OpenFlow protocol, which makes it simpler from the network programmer and SDN controller's point-of-view. In SDN controller's point-of-view, every packet goes through a series of OpenFlow tables and the OVS finds the highest priority matching flow whose conditions are satisfied by the packet and executes its OpenFlow actions [8].

### IV. EMPIRICAL MODEL OF OVS PROCESSING DELAY

In the literature, there are several analytical models based on network calculus and queuing theory [1]–[4], [6], [9] to evaluate the performance of SDN. Most analytical models simplify the network elements to provide a mathematical formulation to evaluate the network performance but in reality, today's network is affected by several external factors as well. In recent years, a significant amount of network functionality has been virtualized (NFV), e.g., virtual routers, switches, firewalls, etc. These virtualized network functions run on top of an operating system (OS) or a hypervisor, which also affects the network performance.

In this paper, we provide an empirical model of the Open vSwitch to characterize different processing delays (non-network related delays) which affect the performance of the OVS. We used different sizes of RAM as well as variable packet sizes and data rates to cover a broad spectrum of configurations.

In the packet processing function of *kernel datapath module* (Figure 1), each packet passes through four steps, which are:

- get statistics of the current datapath from the CPU
- flow rule lookup in the flow tables
- if flow rule not found in the existing flow tables send it to the `ovs-vswitchd` (upcall)
- update the statistics and apply actions

First, the datapath module gets some statistics (counters) about this datapath from the CPU then checks whether it has a cached flow rule or not. If it is present, it will use that rule otherwise send it to the userspace module (upcall) and lastly, it updates the statistics about the current flow (usetime, packet, bytes, tcp\_flag, etc.) and executes the corresponding flow actions. We present delays in microseconds at each of these steps as an input for our empirical model.

Our methodology to calculate the empirical model is as follows [10]; initially, we collect processing time data from each experiment then calculate their frequencies. In the second step, we convert those frequencies into relative frequencies according to equation 1 and calculate commutative relative frequencies to produce the empirical Empirical Cumulative Distribution Function (ECDF).

$$f = \frac{PT}{N} \quad (1)$$

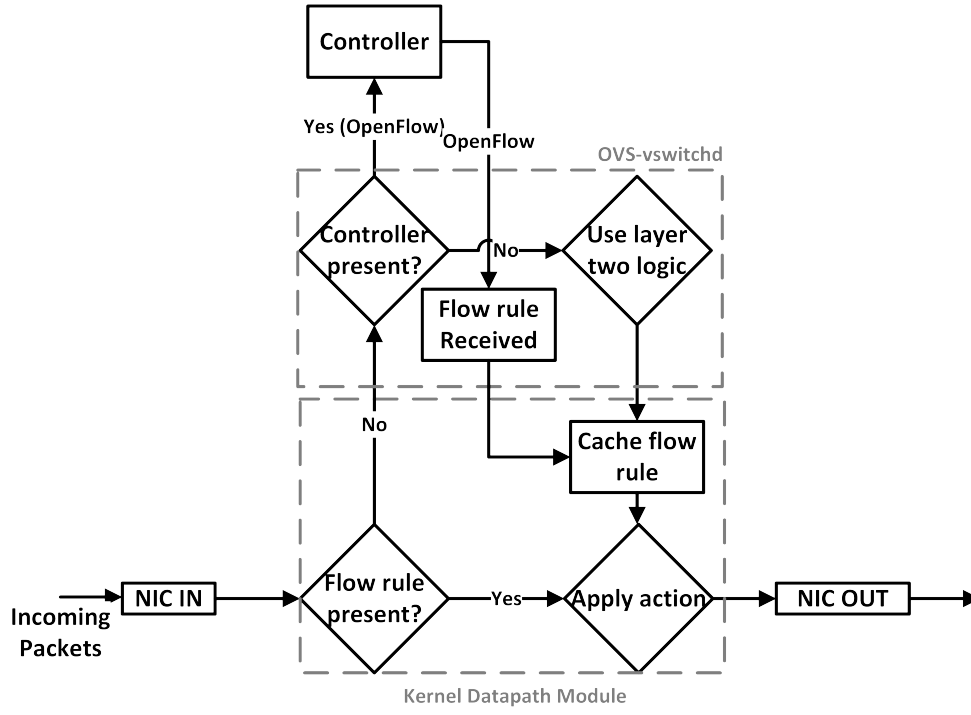


Fig. 1. Overview of the working of the Open vSwitch

PT is the processing time of a packet, N is the total number of packet samples for an experiment and  $\bar{x}$  is the relative frequency.

To cover a broad spectrum of configurations, we used two environments for the experimentation. In the first, we installed OVS in a virtual environment. For the second, OVS was installed directly on the baremetal. The OVS installation for all configurations is single core. We created multiple scenarios. In each scenario, we fixed all the parameters except one and analyzed the effect of that parameter on the packet processing time of OVS, and each experiment is repeated several times. In the following two sections, we discuss the results of virtual and baremetal OVS installation.

## V. VIRTUAL OVS INSTALLATION (VOI)

In the virtual OVS installation, we used Xen server 7.1 [11] to create a virtual machine (VM) and installed Ubuntu server 14.0.4 LTS with OVS 2.5.1 [12]. The hardware specifications of the host system are 8.0GB RAM and Intel E5420@2.50GHz (4 cores).

In this configuration, we analyzed the effect of RAM, packet size and data rate on the processing time of the OVS.

### A. First scenario (variable RAM)

We analyzed the effect of RAM on the packet processing time by setting the number of CPU cores to 1, data rate to 10-15Kb/s and the packet size to 56B. Figure 2 shows the ECDF of processing delay for different sizes of RAM (0.5GB to 2.0GB). Increasing the RAM does reduce the processing delay but it is not significant because the majority of packets

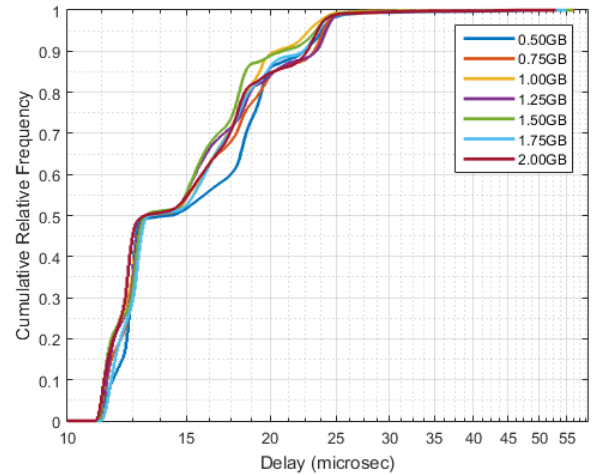


Fig. 2. VOI: Variable RAM (ECDF). RAM: 0.5GB-2.0GB, CPU: 1 core, Packet size: 56B and Data rate: 10-15Kb/s

experience the average total delay of approximately 25  $\mu$ s or less for all the RAM configurations.

### B. Second scenario (variable data rate)

After varying the RAM in the first scenario, the RAM is set to constant values at 1.0GB for the second and third scenarios.

We increase the packet size to 576B in the second scenario to allow for higher data rates. We present the effect of different data rates in Figure 3. At lower data rate (250 Kb/s), the OVS

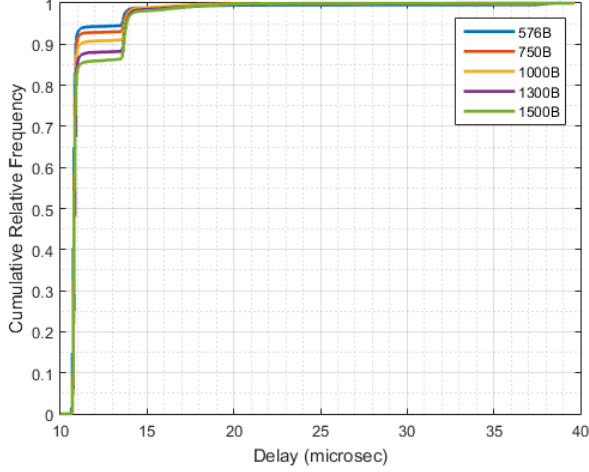


Fig. 3. VOI: Variable data rate and constant packet size (ECDF). RAM: 1.0GB, CPU: 1 core, Packet size: 576B and Data rate: variable

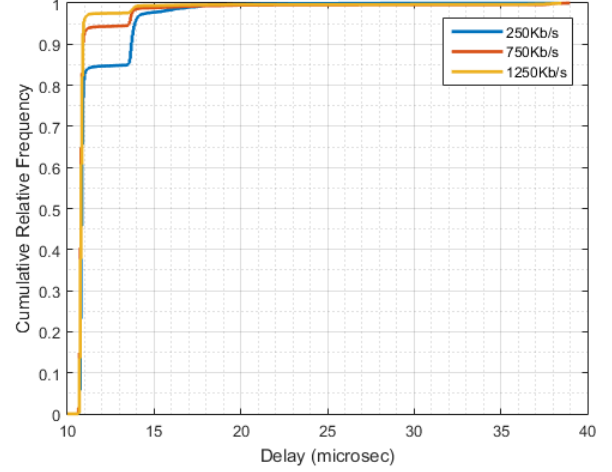


Fig. 4. VOI: Variable packet size and constant data rate (ECDF). RAM: 1.0GB, CPU: 1 core, Packet size: variable and Data rate: 750Kb/s

takes more time to process each packet. It is also evident in comparison with the previous scenario (Figure 2), where data rates (10-15Kb/s) were much lower that Open vSwitch took more time to process a packet as compared to the higher data rates (compare the x-axis values of the Figures 2 and 3). U. Javed *et al.* [5] reported similar behavior while they considered the Round Trip Time (RTT) as a metric. They also reported how the higher data rates lead to lower processing delays. We believe this could be an explanation for the results we get for the processing delay with varying data rates, i.e., they are impacted by context switching and cache hits.

#### C. Third scenario (variable packet size)

We fixed the data rate to 750Kb/s and used variable packet size. The smaller packets take less time to process but the processing time for larger packets is also not significantly different from smaller packet sizes as shown in Figure 4. The processing delay in this scenario is very similar to the second scenario (Figure 3), a maximum delay experienced by any packet is less than or equal to  $40\mu s$ .

#### D. Comparing different types of delay

One interesting result we obtained from our experiments is the comparison of various components of packet processing delay inside the *kernel datapath module*. In particular, we refer to the four delays(i.e. upcall, lookup, statistics update and CPU counters). Figure 5 captures the delays of each component. The parameters used for this experiment are same as the third scenario. The OVS spends a considerable amount of time (in comparison with other types of delay) obtaining information (or waiting for the CPU) about the current datapath from the CPU. The second largest component that contributes to the overall processing delay of the OVS is flow lookup delay.

We also noticed that the time required to obtain CPU counters is higher then upcall to the userspace daemon, flow lookup

and updating the datapath statistics (and preform necessary flow actions).

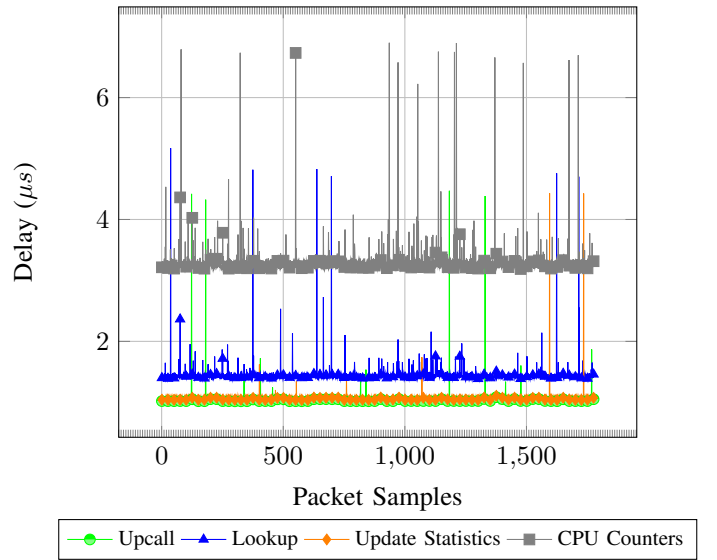


Fig. 5. VOI: Processing delay comparison of different components of *kernel datapath module*

We also performed experiments where we added 2000 arbitrary flow rules to OVS's flow tables to see the effect on the lookup delays (not shown here). The flow lookup delays we obtained in these experiments were not as significant as we expected.

## VI. BAREMETAL OVS INSTALLATION (BOI)

In the second configuration, we installed Ubuntu server 14.0.4 LTS with OVS 2.5.1 directly on the baremetal. The second configuration is used to provide results for the different network parameters. The hardware specifications of the baremetal system are 8.0GB RAM and Intel Core 2 Quad Q6600@2.4GHz (4 cores). In this configuration, we varied

the packet size and data rate. The network parameters for both scenarios are similar to VOI experiments (section V). We also performed these experiments on multiple baremetal systems that produced results (not shown here) similar to BOI results.

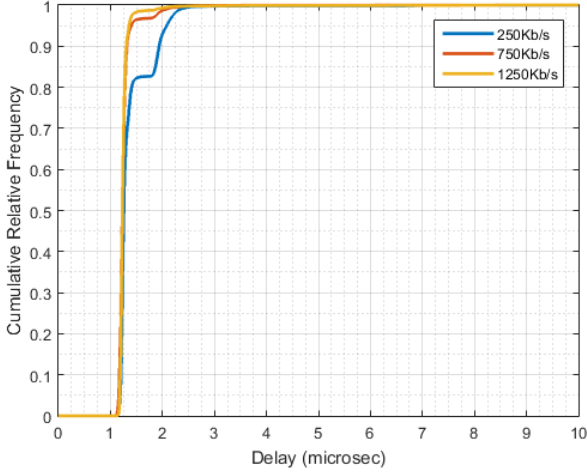


Fig. 6. BOI: Variable data rate and constant packet size (ECDF). RAM: 8.0GB, CPU: 4 cores, Packet size: 576B and Data rate: variable

#### A. First scenario (variable data rate)

We repeated the variable data rate experiment on baremetal as shown in Figure 6. On baremetal, OVS produced similar behavior to the VOI, but it did improve the processing time of the OVS. The processing time is reduced to  $10\mu s$  from  $40\mu s$  (Figure 3).

#### B. Second scenario (variable packet size)

In the second scenario, we repeated the variable packet size experiment on baremetal. This experiment also produced similar results as shown in Figure 7. The OVS took more time to process larger packets but an interesting point to note here is that in VOI the minimum processing delay experienced by any packet is approximately  $10\mu s$ . On the other hand,  $10\mu s$  is the maximum delay encountered by any packet in the BOI, which is a significant reduction as we expected.

#### C. Comparing different types of delay

We used same network parameters as scenario VI-B. This experiment also produced similar behavior. The CPU counters contributed the most in the OVS processing time followed by the lookup delay as shown in Figure 8. One difference we can see between Figure 5 and 8 is that in BOI all delays are more consistent and stable as compared to VOI. The reason could be that in BOI, the CPU scheduling is done by the OS. While in VOI there are two levels of scheduling; one by the OS and other by the Xen Server.

## VII. CONCLUSION

An empirical model of Open vSwitch has been presented in this paper. We conducted experiments using a different

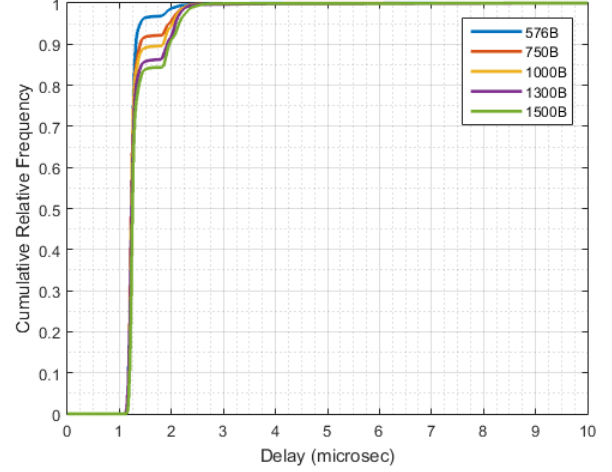


Fig. 7. BOI: Variable packet size and constant data rate (ECDF). RAM: 8.0GB, CPU: 4 cores, Packet size: variable and Data rate: 750Kb/s

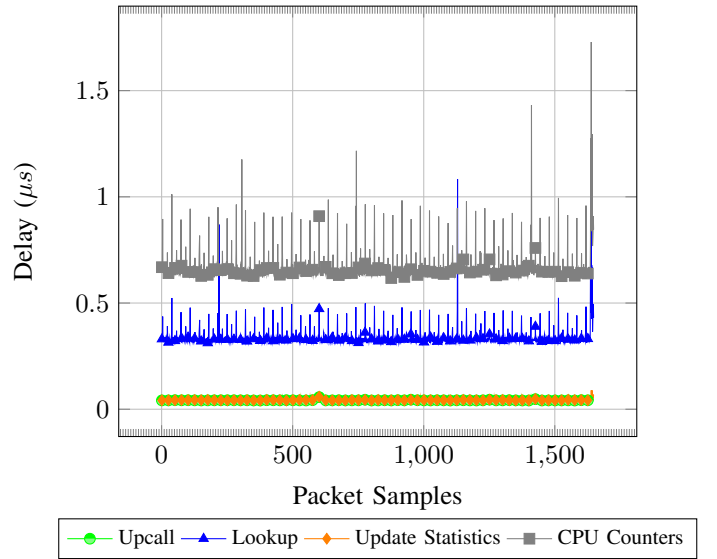


Fig. 8. BOI: Processing delay comparison of different components of *kernel datapath module*

hardware and network configurations. Our empirical model captures the non-network processing delays that a packet experiences due to the nature of different OVS installations as well as other operating system factors. These experiments revealed some interesting results. Increasing the RAM size did not have a significant effect on the delay a packet experiences inside the OVS in the experiments we ran. Smaller packet size and higher data rates reduce the overall processing time of the OVS.

## ACKNOWLEDGEMENT

The second author acknowledges funding from Canada's NSERC through the Discovery Grant Program.

## REFERENCES

- [1] B. Xiong, K. Yang, J. Zhao, W. Li, and K. Li, "Performance evaluation of OpenFlow-based software-defined networks based on queueing model," *Computer Networks*, vol. 102, pp. 172 – 185, 2016.
- [2] 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 *2013 IEEE Global Communications Conference (GLOBECOM)*, Dec 2013, pp. 1397–1402.
- [3] K. Mahmood, A. Chilwan, O. Østerbø, and M. Jarschel, "Modelling of OpenFlow-based software-defined networks: the multiple node case," *IET Networks*, vol. 4, no. 5, pp. 278–284, 2015.
- [4] B. Xiong, K. Yang, J. Zhao, W. Li, and K. Li, "Performance Evaluation of OpenFlow-based Software-defined Networks Based on Queueing Model," *Comput. Netw.*, vol. 102, no. C, pp. 172–185, Jun. 2016.
- [5] U. Javed, A. Iqbal, S. Saleh, S. A. Haider, and M. U. Ilyas, "A stochastic model for transit latency in openflow {SDNs}," *Computer Networks*, vol. 113, pp. 218 – 229, 2017.
- [6] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and performance evaluation of an OpenFlow architecture," in *Teletraffic Congress (ITC), 2011 23rd International*, Sept 2011, pp. 1–7.
- [7] OVS, "Open vswitch manual," accessed 16 May 2017. [Online]. Available: <http://openvswitch.org/support/dist-docs/ovs-appctl.8.txt>
- [8] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The Design and Implementation of Open vSwitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, 2015, pp. 117–130.
- [9] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "OpenFlow Switching: Data Plane Performance," in *Communications (ICC), 2010 IEEE International Conference on*, May 2010, pp. 1–5.
- [10] J. Banks, I. John S. Carson, B. L. Nelson, and D. M. Nicol, *Discrete-Event System Simulation*. Prentice Hall, 2001.
- [11] XEN, "Xen server 7.1," accessed 16 May 2017. [Online]. Available: <https://xenserver.org/overview-xenserver-open-source-virtualization/download.html>
- [12] OVS, "Open vswitch 2.5.1," accessed 16 May 2017. [Online]. Available: <http://openvswitch.org>