Online Resource Mapping for SDN Network Hypervisors using Machine Learning

Christian Sieber, Arsany Basta, Andreas Blenk, Wolfgang Kellerer
Chair of Communication Networks
Department of Electrical and Computer Engineering
Technical University of Munich, Germany
Email: {c.sieber,arsany.basta,andreas.blenk,wolfgang.kellerer}@tum.de

***Abstract*—The virtualization of Software-Defined Networks (SDN) allows multiple tenants to share the same physical infrastructure and to use their own SDN controllers. SDN virtualization is achieved through an SDN network hypervisor that operates between the tenants' controllers and the SDN infrastructure. In order to provide performance guarantees, resource mapping is required for both data plane as well as control plane for each virtual SDN network. In the context of SDN virtualization, the control plane resources include the network hypervisor, which needs to be assigned to guarantee the performance for each tenant. In previous work, the hypervisor resource mapping is based on offline benchmarks that measure the hypervisor resource consumption against the control plane work load, e.g., control plane message rate. These offline benchmarks vary across different hypervisor implementations, e.g., single or multi-threaded, and depend on the capabilities of the deployed hardware platform, e.g., the used CPU. We propose an online approach based on machine learning techniques to determine the mapping of hypervisor resources to the control workload at runtime. This concept is already successfully applied in the context of self-configuring networks. We propose three models to estimate hypervisor resources and compare them for two SDN hypervisor implementations, namely FlowVisor and OpenVirteX. We show through measurements on a real virtualized SDN infrastructure that resource mappings can be learned on runtime with insignificant error margins.**

I. Introduction

A. Motivation

Software-Defined Networking (SDN) decouples the control plane from the data plane leading to better programmability. Virtualizing SDN networks does not only reveal the power of resource sharing among different tenants, it also enables the tenants to deploy their own network operating systems, i.e., SDN controller. Thus, tenants can specifically program the networking behavior of their virtual networks through their controller according to their own demands.

The most used concept to virtualize SDN networks that are based on OpenFlow (OF) is to apply SDN network hypervisors [1]. For SDN networks, they run as intermediate software layers between the tenant controllers and the virtual resources of the data plane. They are responsible for connecting the SDN controllers of the tenants to their virtual networks. Furthermore, SDN network hypervisors control and manage the allocation of physical data plane resources, e.g. network bandwidth, among the different tenants.

978-1-4673-9486-4/16/$31.00 © 2016 IEEE

In non-virtualized SDN networks, the performance of the control plane can have severe impact on the performance of the data plane [2], [3]. This also holds true when virtualizing SDN networks. For instance, in case a network hypervisor instance is overloaded, the overload impacts the processing of the control plane messages of the tenants. This can result in a negative impact on the data plane performance of the tenants, e.g., an increase in web page loading time [4], [5]. Therefore, when requesting virtual SDN networks, tenants should not only demand for data plane resources, they should be able to request control plane resources such as a specific control message rates. The network hypervisor, accordingly, is not only responsible to assure the performance for each tenant on the data plane, but also on the shared control plane. As the hypervisor processes the control traffic of all tenants, a sophisticated resource mapping of hypervisor processing resources to virtual network demands is needed.

B. Related Work and Contribution

In order to allocate hypervisor processing resources to virtual network demands, a mapping between the hypervisor processing resources and the control plane workload, e.g., control message rate is required. Such mapping guarantees that, e.g., enough CPU resources are assigned for the control plane of each virtual SDN network in order to provide the requested performance. The importance of such a mapping has already been shown in [4]. However, the former investigation is based only on an offline performance benchmark of one particular hypervisor. Other existing work on SDN network hypervisors has also only provided offline benchmarks proofing their implementation [6], [7], [1]. Due to the heterogeneity of existing hardware and software solutions, an offline benchmark is generally not feasible in practice.

In this paper, we introduce the concept of determining resource allocation models at runtime. The concept applies online machine learning to estimate the parameter setting for different resource models. Our overall goal is to enable a network hypervisor to work in a self-configuring manner. Self-configuration is, for instance, used for the configuration of virtual machines in cloud systems [8]. In [9], the authors describe parametrized abstract models for the performance of SDN controllers. The authors show in particular that the performance of SDN controllers does not scale linearly with the available resources. SDN hypervisors are not considered in the work. We demonstrate the feasibility of the concept via measurements for network hypervisors in a real testbed. Further, as existing benchmark tools do not allow the generation of specific control traffic mixes, we developed our own benchmarking framework *hvbench* and make it available to the public. Compared to existing solutions such as cbench [10], ofcbenchmark [11], SDLoad [12], PktBlaster [13] and the framework presented in [14], hvbench allows for flexible traffic mixes and message inter-arrival times. Further data and control plane benchmarks for SDN can be found in [15]. For measurement purpose, we emulate virtual network tenants with specific control traffic mixes. The measurement results demonstrate the feasibility of online learning of resource allocation mappings with insignificant error margin.

The remainder of this paper is structured as follows. In Section II, we describe the set-up, a novel benchmarking tool, the proposed models, and the learning approach. Section III shows the evaluation results. Section IV outlines conclusions.

II. Methodology

In the following, we first introduce the experimental set-up, including a novel hypervisor benchmarking tool *hvbench*. hvbench can generate control traffic mixes based on probability distributions. Afterwards, we discuss the control plane arrival process of the tenants and the considered OF control messages in detail. Finally, three learning models for the hypervisor processing resources are proposed and evaluated. The evaluation framework *hvbench* and the run traces are available as open-source for download [16].

A. Setup

Figure 1 depicts the experimental setup. The setup consists of an emulated SDN data plane based on mininet [17], a network hypervisor instance, a resource monitor and the benchmarking tool hvbench. One virtual network is configured, where hvbench serves as the controller of this virtual network. hvbench is based on libfluid [18], which is a library that provides the basic features of an OpenFlow controller. The experimental procedure is as follows. First, the switches establish a connection to the hypervisor, which consequently connects to hvbench. Next, the control plane arrival process starts while the resource monitor records the resource usage of the hypervisor instance. Note that hvbench uses only one control plane connection and emulates the messages of different tenants via this one connection.

The hypervisor resource monitor records key performance indicators (PKI) of the hypervisor process. The PKIs are recorded with a frequency of $1/s$. In this work, we focus on the CPU utilization ρ induced by the hypervisor process.

We synchronize the measurements of the control message rate (measured by hvbench) and the resource monitor based on the time of the underlying operating system.

B. Control Plane Arrival Process

Table I summarizes the nomenclature, constants and distributions used in this work. The overall control plane process

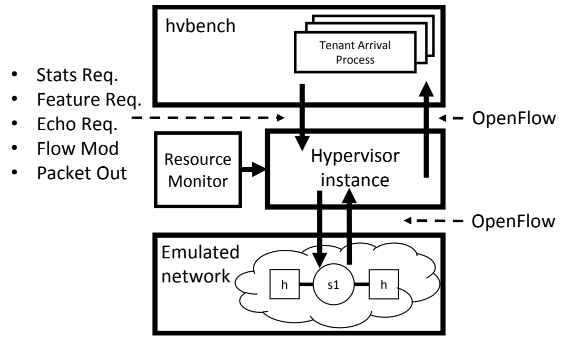

Figure 1. Experimental setup including hvbench, hypervisor instance, resource monitor and an emulated SDN network environment.

can be structured in two independent processes. First, there is the tenant arrival process which describes when a new tenant is admitted and with which average request rate. Second, each tenant has a specific control message pattern which describes the type of requests that are generated. Each request type follows a random distribution, which describes the amount of messages generated during the tenants lifetime and also defines the arrival time of each request message.

Table I
Nomenclature & Variables

Nomenclature	
$l \in N$	Duration of the experiment in seconds.
$t, t \in \{0, 1, .., l\}, l > 0$	Experiment time.
$O := \{fr, er, sp, sf, po, fm\}$	Types of OpenFlow requesst.
r^t_o	Requests per second of type o at t.
r^t_{all}	$\sum_{o \in O} r^t_o$.
r^t	Synonym for r^t_o and r^t_{all}
$\rho^t_{meas} \in [0, 1]$	Measured CPU utilization at t.
$\rho^t_{lin}(r^t_{all})$	Single dimen. linear model for ρ^t_{meas}.
$\rho^t_{lin2}(\{r^t_o, \forall o \in O\})$	Multi dimen. linear model for ρ^t_{meas}.
$\rho_{exp}(r^t_{all})$	Exponential model for ρ^t_{meas}.
$M := \{lin, lin2, exp\})$	Investigated models.
$T := \{1, 2, 3, 4, 5, 6\}$	Type of tenants (see Table II).
Constants & Distributions	
$c_{iat} := 5$: Constant tenant inter-arrival time (s).	
$R_{avg} \sim \mathcal{N}(200, 50^2)$: Average tenant message rate (req/s).	
$\tau_x \sim \exp(R_{avg})$: Tenant x message inter-arrival time.	

We model a gradual admission of tenants with a constant inter-arrival time of $c_{iat} := 5s$. The first tenant arrives at $t := 0$ and marks the start of an experiment run. R_{avg} describes the average control message rate of all message types per tenant. It follows a normal distribution with $\mu := 200$ and $\sigma := 50$. For each tenant, the average request rate is chosen upon tenant's arrival. R_{avg} is restricted to the interval $[10, 2{\cdot}\mu]$. The inter-arrival time of the requests generated by each tenant is exponentially distributed with an average rate of R_{avg}. Each tenant is assigned a set of weights, which describes the discrete distribution of the request type. Table II shows the weights of each tenant type as well as the weights for each request type per tenant type. We illustrate the whole process by example. When at $c_{iat} \cdot 1 = 5s$ the second tenant is admitted, first, the process chooses a normal distributed request rate R_{avg} for the

Table II
TENANT CONTROL PLANE ARRIVAL WEIGHTS

T	Weight	$\{r_{fr}, r_{er}, r_{sp}, r_{sf}, r_{po}, t_{fm}\}$
T_1	0.1	$\{0.5, 0.1, 0.2, 0.1, 0.1, 0.1\}$
T_2	0.1	$\{0.1, 0.3, 0.4, 0.1, 0.1, 0.1\}$
T_3	0.2	$\{0.3, 0.3, 0.2, 0.1, 0.1, 0.1\}$
T_4	0.2	$\{0.05, 0.04, 0.45, 0.45, 0.1, 0.1\}$
T_5	0.2	$\{0, 0, 0, 0, 1, 0\}$
T_6	0.2	$\{0.1, 0.1, 0.1, 0.1, 0.1, 0.5\}$

Table III
HARDWARE CONFIGURATIONS

C1	Virtual, 2 x i7-4770 CPU 3.40GHz
C2	Virtual, 1 x i7-4770 CPU 3.40GHz
C3	Physical, 2 x i7-4790 CPU 3.60GHz

tenant, e.g., 180 requests per second. Afterwards, the tenant is assigned a tenant type T based on a discrete distribution with the weights as given in Table II, e.g., T_5. Here, the tenant generates only packet out messages. The tenant now generates exponentially distributed packet out messages with an average rate of 180 messages per second until the experiment finishes.

C. Hypervisor Processing Learning Models

This work defines three learning models for the hypervisor CPU utilization ρ. Equation 1 defines a linear model where ρ depends on the total average requests per second r^{all} and on the model coefficients β_1 and β_2. β_1 represents the processing cost per one request and assumes that the processing cost for all request types is equal. Equation 2 describes a model where ρ depends on the request rate r_o and cost β_o for each of the request types ($\forall o \in O$). The third cost function in Equation 3 assumes an exponential relationship between the request rate r_{all} and ρ. $\beta_2, \beta_I, \beta_b$ are constant offsets.

$$\rho_{lin}(r_{all}) := \beta_1 \cdot r_{all} + \beta_2 \tag{1}$$

$$\rho_{lin2}(r) := (\sum_{o \in O} \beta_o \cdot r_o) + \beta_I \tag{2}$$

$$\rho_{exp}(r_{all}) := \beta_c \cdot (1 - e^{-\beta_a \cdot r_{all}}) + \beta_b \tag{3}$$

Next, we describe the iterative fitting process to estimate the model coefficients β_0, β_1, β_o, β_a, β_b and β_c during the gradual tenant admission process. Subsequently, we denote the CPU utilization estimated by the models at point in time t as ρ^t and ρ^t_{meas} as the measured CPU utilization at time t. β^t describes the value of β at time t in the learning process. Based on an initial training, we define the following initial state for the model coefficients β. For ρ_{lin}, $\beta_1 = 0.000029$ and $\beta_2 = 0.1$. For ρ_{lin2}, $\beta_o := 0.00001, \forall o \in O$ and $\beta_I := 0.1$. For ρ_{exp}, $\beta_a = 0.000026$, $\beta_b = 0.201$, and $\beta_c = 1.004103$.

D. Online Learning Method

The online learning is an iterative process structured in three phases. In the first phase, new samples are collected. The resource monitor continuously gathers a new sample each second ($l = l + 1$). Second, fitting the model function(s) to the collected samples. Third, the models are updated based on the fitting results. The process is continuously repeated when a new measured sample is available.

We restrict the adaptation of β per iteration to 10 % ($\beta^{t+1} \in [\beta^t \cdot 0.9, \beta^t \cdot 1.1]$) to make the process more robust against short term fluctuations or fitting errors. For fitting the models, we use weighted orthogonal distance regression (ODR) as introduced by Boogs et al. in [19]. ODR considers measurement errors in the input, e.g. clock drift or inaccuracy in the sending process of hvbench or in the resource monitor, and output dimension, e.g. model discrepancy.

III. RESULTS

We evaluate the processing learning models for two popular OpenFlow-based SDN hypervisors, FlowVisor (FV) [6] and OpenVirteX (OVX) [20]. We compare the results to a scenario without a hypervisor. For this we establish a direct connection between the switch in the emulated network and hvbench. Three configurations of hardware platforms as shown in Table III are evaluated. C1 is a virtual environment with two virtual CPU cores, C2 is a virtual environment with one core and C3 is a physical platform with two physical CPU cores.

First, we evaluate the performance of the proposed processing learning models given a specific hardware configuration, namely C1. The learning models performance is computed in terms of the mean squared estimation error, which is the difference between estimated CPU utilization ρ^t over experiment time t compared to the real measured values ρ^t_{meas}.

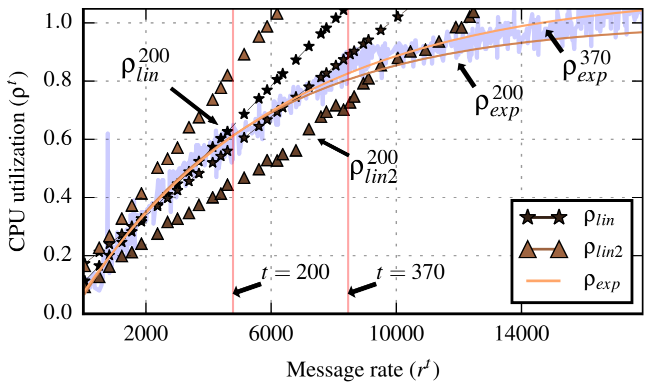

Figure 2. Snapshot of the state of the three models at two points in time ($t = 200, 370$) during experiment run. ρ_{exp} at $t = 370$ can predict the CPU utilization accurately. Horizontal red lines estimate t based on r_t and c_{iat}.

A configuration with OpenVirteX running on a virtual machine, i.e., OVX_C1, is taken as a show case. The experiment duration is 800 s. Figure 2 illustrates the iterative fitting process of estimating the CPU utilization for the three proposed learning models ρ^t_{lin}, ρ^t_{lin2} and ρ^t_{exp}. The figure shows a snapshot of the estimation at two points in time, namely at $t = 200$ s and $t = 370$ s.

The vertical lines indicate the approximate translation of r^t to t, extrapolated from the constant tenant arrival rate. The semi-transparent area in the background visualizes the

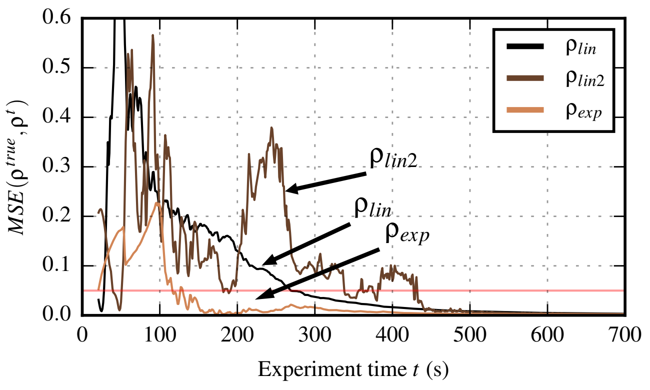

Figure 3. Mean squared error (MSE) between the model ρ^t at time t and the true CPU usage ρ^{meas} for all $t \in T$. Horizontal red line marks an error of 0.05. ρ_{exp} stays underneath threshold starting from $t = 120$.

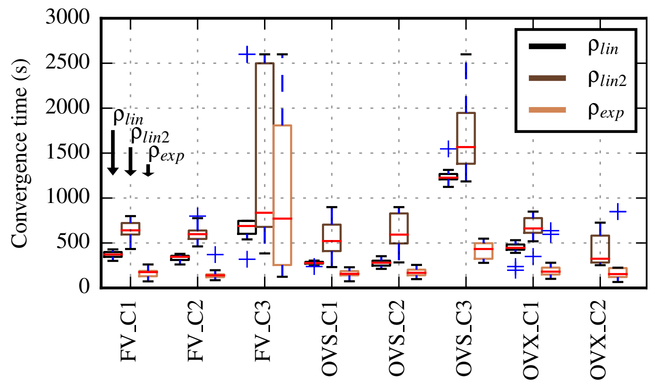

Figure 4. Experiment time t where the MSE between the model and the measured CPU time is and stays lower than threshold 0.05. Lower values are better. ρ_{exp} outperforms the ρ_{lin} and ρ_{lin2} models for most configurations.

measured CPU utilization ρ^t_{meas}. The lines and curves depict ρ^t_{lin}, ρ^t_{lin2} and ρ^t_{exp} for $t \in \{200, 370\}$. We see that ρ^{200}_{exp} slightly underestimates the real utilization by about 10 %, but at $t = 370$, the model stays close to the real CPU utilization. The linear models ρ_{lin} and ρ_{lin2} are not able to predict the future CPU state accurately, as they overfit to the slope between $r := [0, 6000]$ and therefore overestimate the future utilization.

Figure 3 depicts the mean squared errors $MSE(\rho^t_m, \rho^t_{meas}), \forall t \in \{0, 1, .., l\}, m \in M$ between the model ρ^t_m at time t and the real CPU utilization ρ^t_{meas} for all $t \in T$ for configuration OVS_C2, excluding the first and last 10 % of collected samples. Hence, the figure shows the fitting quality in terms of MSE of all iterative steps during the experiment run. The horizontal line marks an error threshold of 0.05. The results show that the three models require different run time in order to obtain an estimation with a MSE lower than the threshold of 0.05. For instance, the exponential model can reach the MSE threshold for $t > 180s$, while the linear model can only reach it at $t > 270s$. This shows the difference in convergence time to reach an estimation quality between the three evaluated processing models.

Next, we summarize the results of all experiment runs among the three configurations and the two considered hypervisors in terms of convergence time over 20 experiment runs for each hypervisor and configuration. Convergence time is defined here as the experiment time at which the fitting quality in terms of MSE reaches the threshold of 0.05 and stays under the threshold for the remaining run time, i.e, $(\exists! t : \forall x \in \{t, t+1, .., l\}, MSE(\rho^x, \rho^{meas}) \leq 0.05)$.

In Figure 4, the lower axis depicts all combinations of configurations and hypervisors. The left axis gives the convergence time in seconds. The whiskers indicate the range of the values, the box indicates the 25 % and 75 % quartiles, respectively, and the line the median of the values. From the figure we conclude that ρ^{exp} can predict the CPU usage of the hypervisors at about 200 s into the experiment for most combinations. Furthermore, the linear model, which does not distinguish the request types, outperforms the second linear model, which considers the type of the requests. The results also show that there is a difference between the different configurations. While the linear models are good in predicting the usage for OVS in the virtual environments, for the hardware environment C3, the prediction quality is significant lower. For FlowVisor on the physical hardware, the exponential model shows a highly varying convergence time compared to the linear model.

Next, we take a closer look at whether the three models under- or overestimate the CPU utilization and we quantify the under- or overestimation for each model for one configuration. Overestimating the future CPU utilization can result in degrading the overall performance, e.g., slow tenant admission or tenant request rejection, and it could also result in an inefficient resource utilization due to over dimensioning. On the other hand, underestimation can result in an overload of the hypervisor instance by accepting tenant requests exceeding the CPU capacity. The under- or overestimation are calculated as follows:

$$X^t_m := \{\forall \alpha \in \{t, t+1, .., l\} | \rho^\alpha_{meas} - \rho^t_m(r^\alpha) > 0\} \quad (4)$$

$$\phi^t_m := \frac{\sum_{x \in X^t_m} (\rho^\alpha_{meas} - \rho^t_m(r^\alpha))}{|X^t_m|} \quad (5)$$

$$Y^t_m := \{\forall \alpha \in \{t, t+1, .., l\} | \rho^\alpha_{meas} - \rho^t_m(r^\alpha) < 0\} \quad (6)$$

$$\psi^t_m := \frac{\sum_{x \in X^t_m} (\rho^\alpha_{meas} - \rho^t_m(r^\alpha))}{|X^t_m|} \quad (7)$$

X^t_m in Equation 4 defines the points in time where the model $\rho^t_m, m \in M$ at experiment time t underestimates the future CPU utilization $\rho^x_{meas}, \forall x : x > t$. In Equation 5, ϕ^t_m defines the mean absolute underestimation of model m starting from time t till the end of experiment time l. Y^t_m and ψ^t_m in Equations 6 and 7 define the mean absolute overestimation, analogous to X and ϕ for underestimation.

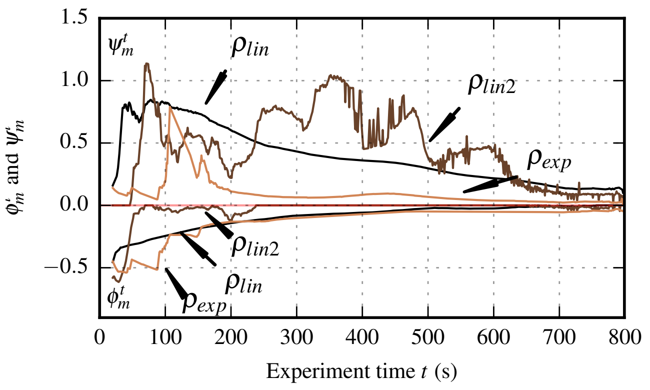

Figure 5. Mean absolute over- and underestimation of the models over time for an OpenVirteX instance running on hardware configuration C1. The top part of the figure depicts the overestimation, the bottom part the underestimation. Overestimation can lead to inefficient resource usage, while underestimation can overload the hypervisor, thus effecting all tenants.

Figure 5 depicts ϕ and ψ for the SDN hypervisor OpenVirteX on hardware configuration C1 over experiment time t. Overestimation ψ is shown on the top of the figure, i.e., positive values > 0. Underestimation ϕ is shown on the lower part of the figure, i.e., negative values < 0. From the figure we conclude that ρ_{lin} and ρ_{exp} both underestimate the utilization in this configuration. ρ_{lin2} exhibits mostly overestimation. In terms of convergence speed, both, ρ_{lin} and ρ_{exp}, decrease the underestimation at a similar rate starting from experiment time 100. For the overestimation case, the figure shows that ρ_{lin} and ρ_{lin2} exhibit high overestimation compared to ρ_{exp}. In addition to high overestimation, ρ_{lin2} shows an oscillating behavior. ρ_{exp} can decrease the amount of overestimation soon into the experiment and can keep it stable at a low level starting from $t > 130$. Although the results of the figure can not be generalized to all hardware configurations, the results give important indications on the under- and overestimation characteristics of the three models.

IV. Conclusion

Network hypervisors will become an essential part of software-defined future communication networks. As they are implemented in software and operated in virtual environments with varying processing resources, it is important to understand the relationship between control message rate and CPU consumption in general and adapt that understanding online to the current virtual environment. We use an iterative online-learning approach to compare the prediction quality of three different models in different environments with two popular OpenFlow hypervisors and an OpenFlow switch implementation. The results show that linear models are inadequate to predict the CPU consumption for increasing message rates. An exponential model can predict future CPU consumption earlier and with high accuracy. In particular, the exponential model converges fast to a state with low over- and underestimation of the resource usage. The results are a first step towards dynamic and autonomous load-balancing of network hypervisors in virtual environments with varying resources. In the future we plan to extend the current approach to scenarios where the available computing resources are not constant. Furthermore, we will apply the methodology to a larger set of configurations and hypervisors.

Acknowledgment

This project has received funding, in part, from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No 647158 FlexNets). This work reflects only the authors' view and the funding agency is not responsible for any use that may be made of the information it contains.

References

[1] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on Network Virtualization Hypervisors for Software Defined Networking," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 655–685, 2016.

[2] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proc. INM/WREN'10*, Berkeley, CA, USA, 2010.

[3] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proc. USENIX Hot-ICE'12*, Berkeley, CA, USA, 2012, pp. 10–10.

[4] A. Blenk, A. Basta, and W. Kellerer, "HyperFlex: An SDN virtualization architecture with flexible hypervisor function allocation," in *Proc. IFIP/IEEE Int. Symp. on Integrated Network Management (IM)*, 2015.

[5] A. Basta, A. Blenk, Y.-T. Lai, and W. Kellerer, "HyperFlex: Demonstrating control-plane isolation for virtual software-defined networks," in *Proc. IFIP/IEEE Int. Symp. on Integrated Network Management (IM)*, 2015.

[6] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. Mckeown, and G. Parulkar, "FlowVisor: A network virtualization layer," OpenFlow Consortium, Tech. Rep., 2009.

[7] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, W. Snow, and G. Parulkar, "OpenVirteX: a network hypervisor," in *Proc. Open Networking Summit (ONS)*, Santa Clara, CA, Mar. 2014.

[8] X. Bu, J. Rao, and C.-Z. Xu, "Coordinated self-configuration of virtual machines and appliances using a model-free learning approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 4, pp. 681–690, 2013.

[9] T. Sato, S. Ata, I. Oka, and Y. Sato, "Abstract model of sdn architectures enabling comprehensive performance comparisons," in *Network and Service Management (CNSM), 2015 11th International Conference on*, Nov 2015, pp. 99–107.

[10] "cbench," https://github.com/andi-bigswitch/oflops/tree/master/cbench.

[11] M. Jarschel, F. Lehrieder, Z. Magyari, and R. Pries, "A flexible openflow-controller benchmark," in *2012 European Workshop on Software Defined Networking (EWSDN)*, Oct 2012, pp. 48–53.

[12] N. Laurent, S. Vissicchio, and M. Canini, "Sdload: An extensible framework for sdn workload generation," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014.

[13] "PktBlaster," http://www.veryxtech.com/products/product-families/pktblaster-sdn-software-defined-network-test.

[14] J. Teixeira, G. Antichi, D. Adami, A. Del Chiaro, S. Giordano, and A. Santos, "Datacenter in a box: Test your sdn cloud-datacenter controller at home," in *2013 Second European Workshop on Software Defined Networks (EWSDN)*, Oct 2013, pp. 99–104.

[15] D. Kreutz, F. M. Ramos *et al.*, "Software-defined networking: A comprehensive survey," *proceedings of the IEEE*, vol. 103, no. 1, 2015.

[16] "Source code and experiment traces used in the paper," http://git.io/v4ffJ.

[17] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, p. 19.

[18] C. R. A. Vidal, E. Fernandes and M. Salvador, "libfluid," http://opennetworkingfoundation.github.io/libfluid/, 2015.

[19] P. T. Boggs, R. H. Byrd, and R. B. Schnabel, "A stable and efficient algorithm for nonlinear orthogonal distance regression," *SIAM Journal on Scientific and Statistical Computing*, vol. 8, no. 6, 1987.

[20] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, "Extending networking into the virtualization layer." in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2009.