On The Impact of the Network Hypervisor on Virtual Network Performance

Andreas Blenk[1,2] Arsany Basta[1] Wolfgang Kellerer[1] Stefan Schmid[2]
[1] Technical University of Munich, Germany
[2] Faculty of Computer Science, University of Vienna, Austria

***Abstract*—Virtualization and multi-tenancy are attractive paradigms to improve the utilization of computing infrastructures and hence to reduce costs. In order to provide a high degree of resource sharing without sacrificing predictable cloud application performance, strict performance isolation needs to be ensured. This is non-trivial and requires models which account for *all* components where applications may interfere: similarly to security, the predictability of cloud application performance can only be as good as the least predictable component in the model.**

This paper identifies a new source of potential performance interference that has been overlooked so far: the *network hypervisor* — a critical component in any multi-tenant network. We present a first measurement study of the performance implications of the network hypervisor in Software-Defined Networks (SDNs). For the purpose of our study, we developed a new open-source benchmarking tool for OpenFlow control and data planes.

We show that cloud application performance may appear unpredictable if the network hypervisor is not accounted for: the performance does not only depend on the specific hypervisor implementation and workload (e.g., OpenFlow message types), but also on the number of tenants and the size of the network. Hence, our results suggest that hypervisors should be included in our performance models, and their performance benchmarked and compared similarly to other crucial software components such as the SDN controller.

***Index Terms*—Network Virtualization; Software-Defined Networking; Network Measurements**

I. Introduction

Virtualization enables a high degree of resource sharing and hence reduces costs. However, it also introduces a challenge of providing a predictable application performance. Only recently, measurement studies have demonstrated that in order to provide a predictable cloud application performance, it is also important to account for the potential interference *on the network* [1]: applications such as batch processing, streaming, and scale-out databases, generate a significant amount of network traffic and a considerable fraction of their run time is due to network activity.

Network virtualization promises a solution by offering a unified abstraction and resource isolation across servers and their interconnecting communication networks. Over the last years, several virtual network abstractions such as *virtual clusters* [2], have been proposed and used in many systems, e.g., [2], [3], [4].

At the heart of any network virtualized architecture, lies a *network hypervisor* which multiplexes different tenants across the shared substrate network. The network hypervisor is responsible for network abstraction and control plane translation. Software-Defined Networks (SDNs) provide a particularly interesting framework for network virtualization [5], [6]: a virtual SDN (*vSDN*) network offers great management flexibilities to its tenant. Network hypervisors expose a northbound OpenFlow API which allows tenants to "bring" their own controller. As such, tenants can target the controller platform that better matches their application, using OpenFlow API that is "in most cases" a unified interface for SDN control.

Interestingly, not much is known today about the *performance impact* of the network hypervisor itself. This is problematic as, in order to ensure a predictable performance, models are required which account for *all* involved resources where applications may interfere. Put differently, the performance predictability of a given cloud application can only be as good as its least predictable component. Cloud applications based on models which ignore certain components (such as the network hypervisor) entirely, may perform in unexpected ways. This paper makes a case for including the network hypervisor in our cloud application performance models. We first present a simple web browing case study which demonstrates the potential performance impact of the SDN hypervisor. It motivates us to subsequently conduct an explorative measurement study.

We find that the network hypervisor can influence performance in several ways: for example, we show that both control latency overhead introduced by the network hypervisor as well as its variability depend on the specific hypervisor technology. We also show that the performance depends on the workload, and more specifically, the OpenFlow message types. Another interesting finding regards the effects tenants have on each other. In particular, we identify a novel and subtle source of interdependancy among tenants, which can harm predictability: we expose additional delays induced by multi-tenancy, which depend on the number of tenants. We also investigate the impact of the network size, i.e., number of switches, on the hypervisor performance.

For the purpose of this study and in order to investigate the above questions, we developed a novel benchmarking tool, called *perfbench*. The tool is built specifically to support experiments in a multi-tenant virtual environment. It is built on top of the libfluid C++ library [7]. In contrast to our initial publication of *perfbench* [8], this paper presents a detailed description of the tool.

ISBN 978-3-903176-16-4 © 2019 IFIP

The remainder of this paper is organized as follows. In order to verify our hypothesis that the network hypervisor influences performance, we discuss a simple web surfing use case in Section II. Section III introduces our methodology and Section IV discusses our measurement results. After reviewing related work in Section V, we summarize our contributions and outline future work in Section VI.

II. A Simple Case Study

To test our hypothesis about the performance impact of the network hypervisor, we consider a case study: *web surfing*. While the case study is admittedly simplistic, it will motivate the subsequent explorative measurement study.

Conventionally in OpenFlow, a flow setup invokes an `OFPT_PACKET_IN`, containing the new packet, from the switch to controller. The controller responds with an `OFPT_FLOW_MOD`, i.e., the flow rule, and an `OFPT_PACKET_OUT`, containing the new packet, back to the switch. In *virtual SDN networks (vSDNs)*, see Fig. 1, a hypervisor is used to translate the OpenFlow control messages from/to the controller used by the tenants to manage their *slices*. This translation can include several fields, e.g., switch Datapath ID (DPID) and port numbers.

For this experiment, we use the FlowVisor [9] network hypervisor and Open vSwitch (OvS) running OpenFlow version 1.0. The hypervisor connects to an SDN controller based on Ryu: it serves as the controller of the virtual SDN network. On the data plane, the OvS switch connects on one port to an Apache HTTP web server, hosting a web page which consists of 100 image file queries, and on another port, to a web client. The client requests the web page periodically, each time from a different source port, triggering a new flow setup upon every web page load event.

We evaluate two scenarios. In the first scenario, the web page load latency is measured when the hypervisor computing resources (in terms of CPU) are less loaded, i.e., less than 50%. In the second scenario, the measurement is performed when the hypervisor CPU is highly loaded, i.e., operating at 90% - 100% utilization, by running a background control application.

Fig. 2 shows the impact of the hypervisor performance on the virtual network performance for a new flow setup. In case the CPU resources of the hypervisor are highly loaded, the web page load latencies for users of the virtual SDN network increase from an average of 5 ms up to 70 ms. This implies that the hypervisor may become a bottleneck for the control plane, and consequently also for the data plane: the performance of the control plane can have a significant impact on the network performance [10].

III. The Perfbench Benchmark and Methodology

Our use case motivates us to conduct a more systematic study of the network hypervisor performance, and develop benchmarks accordingly. In general, there exists a wide spectrum of workloads to be supported on the hypervisor. For example, in a data center, typical traffic loads [11], [12] can vary significantly in the number of network flows, from a few thousand to tens of thousands of flows: e.g., 2k to 10k flows were reported at EC2 and Azure [13]. They scale with the number of tenants running on top of the hypervisor.

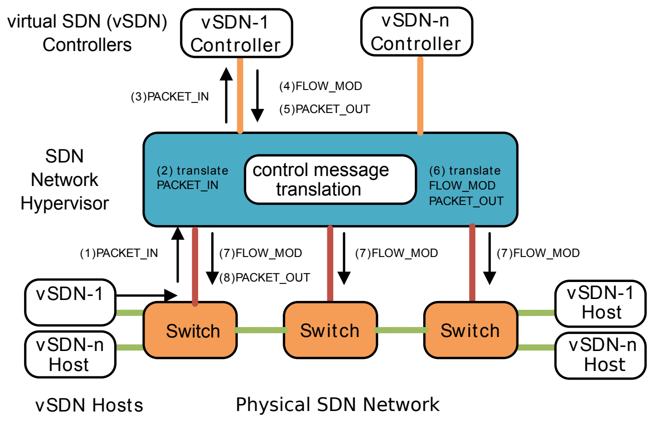


Fig. 1: SDN network hypervisor concept.

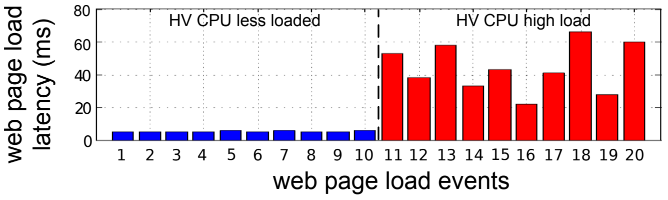


Fig. 2: Impact of the network hypervisor performance on the performance of virtual networks.

For the purpose of our study, we developed a novel benchmarking tool called *perfbench* for OpenFlow-based SDN networks: *perfbench* supports high and stable message rates, allowing us to study the performance under different rates. This section gives an overview of the *perfbench* tool, and puts it into perspective with respect to existing tools, focusing on our particular use case: multi-tenant virtual SDN networks. The *perfbench* framework is available open source [1].

A. High Throughput Measurement Tool

The *perfbench* tool is tailored toward high throughput performance benchmarks for OpenFlow-based SDN networks: it emulates high OpenFlow (OF) message rates, and can be used to conduct measurements for both multi-tenant as well as non-virtual SDN networks. It builds on top of the libfluid C++ library [7], which provides the basic implementation and interfaces for OF messages. As libfluid supports OF versions 1.0 and 1.3, *perfbench* can benchmark OF networks with these versions. Fig. 3 gives a conceptual view of *perfbench*'s design and how it operates in multi-tenant SDN networks.

The *perfbench* tool consists of two parts, as illustrated in Fig. 3: a control plane part (*perfbenchCP*) and a data plane part (*perfbenchDP*). In this paper, we will employ *perfbench* in both modes:

[1]"perfbench". online at: https://github.com/tum-lkn/perfbench

perfbenchCP: The control plane part *perfbenchCP* runs the processes that emulate the tenant SDN controllers. To guarantee isolation between the controllers, we run each controller in its own thread and connect it (via a unique TCP socket) to the measured entity, i.e., either a hypervisor or an SDN switch.

For each controller process, a message rate of OF messages can be generated by specifying two parameters: sending interval (inter-arrival time) and burst size. The sending interval in ms can be specified beforehand and its resolution ranges from $1\,\mathrm{ms}$ to $1\,\mathrm{s}$. For instance, an interval of $1\,\mathrm{s}$ emulates bursts on a per-second basis, whose effect, however, is not investigated in this work. Throughout this paper, we will employ the minimum possible sending interval, i.e., $1\,\mathrm{ms}$. The burst size determines the number of OF messages sent at each send interval. Hence, the combination of send interval and burst size determines the generated OF message rate (per second).

perfbenchDP: The data plane part *perfbenchDP* emulates the data plane. *perfbenchDP* has two modes of operation. It can be connected to existing OF switches directly, i.e., providing a data plane port or interface to the OF switch. This serves the purpose of measuring OF control messages that contain a payload, e.g., to receive UDP packets from the OF switch that were sent via `OFPT_PACKET_OUT` messages. Alternatively, *perfbenchDP* can also emulate OF switches: in this mode, it connects directly to the control plane of either an SDN controller or a hypervisor, e.g.,to generate `OFPT_PACKET_IN` messages. In case of OF switch emulation, several *perfbenchDP* switches can be instantiated, each in its own thread to guarantee isolation in the data plane as well. The OF message generation in the emulated switch follows the same procedure as in the controller process.

A scheduler runs on top of the controller process(es) and emulated switch(es), which determines the OF message rate distribution over the duration (specified in seconds) of a measurement run. *perfbench* implements three inter-arrival time message distributions that can be used by the scheduler: uniform, exponential, or weibull. Alternatively, a user can give a pre-defined scheduling of messages via an input file. Hence, a user can generate any kind of control plane traffic in terms of the inter-arrival times and the burst size of sent messages, e.g., based on real network traces.

B. Comparison to Existing Tools

Tab. I provides a comparison of *perfbench* to existing tools with focus on the supported features.

Existing benchmark tools for OF networks can be classified into two types, namely switch and controller benchmarks. Switch benchmark tools emulate controllers and test the performance of switches, while controller benchmark tools emulate switches to measure the performance of controllers.

Only *perfbench* and *hvbench* [17] are designed to measure the performance in virtual SDN networks, i.e., supporting the emulation of multiple tenant controllers and the interaction with an SDN network hypervisor. *hvbench* is based on the libfluid C++ library, and provides several OF message types for both OF versions 1.0 and 1.3. In contrast to *perfbench*,

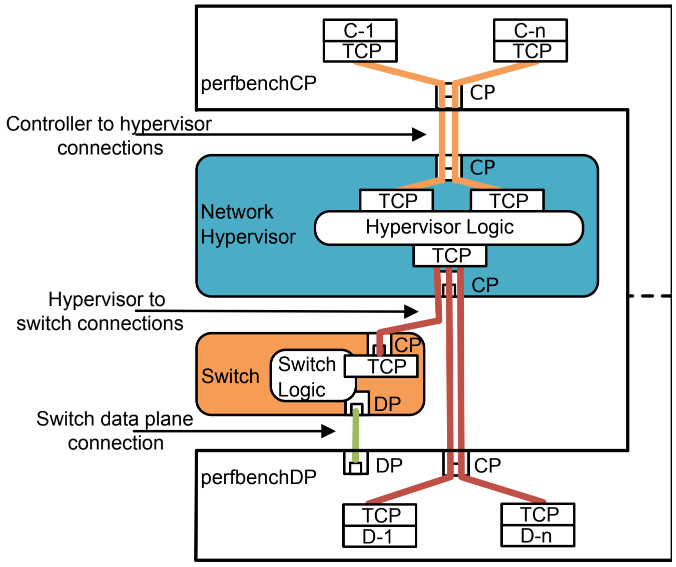


Fig. 3: Perfbench conceptual view and architecture.

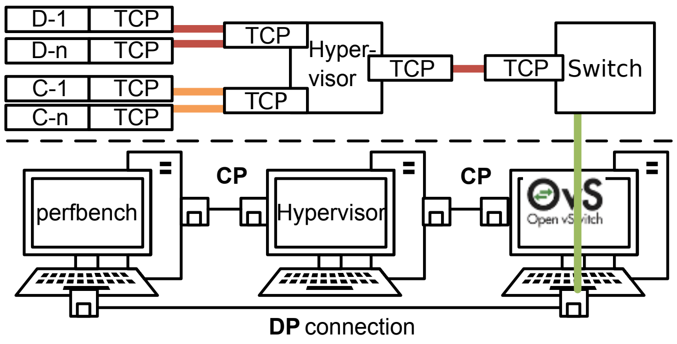


Fig. 4: Network hypervisor benchmarking and measurement setup. Data plane processes: D-1 to D-n. Control plane processes: C-1 to C-n.

hvbench has not been designed to be used in a non-virtual SDN environment. Additionally, *hvbench* can only generate OF control traffic with an exponential distribution.

OFTest [14] is a switch benchmark tool that supports OF version 1.0 only and provides performance test for OF switches, i.e., the control as well as data planes of switches. Besides OFTest, *perfbench* is the only tool that can benchmark both control plane and data plane. Compared to *perfbench*, OFTest provides a much lower throughput of OF messages (only rates up to 1000 messages per second), due to its implementation in Python. It was one of our main motivations to implement a high performance benchmarking tool that is able to generate stress loads.

Another switch benchmark tool is OFLOPS [16], which focuses on evaluating switch capabilities from different vendors. Example performance metrics considered by OFLOPS are flow table update rate and flow insertion latency. Although these are important metrics for non-virtual OF networks, they cannot directly evaluate the performance of network hypervisors. Hence, there is a need for a tool that targets performance

TABLE I: Feature comparison with existing OF benchmarking tools. ✓ indicates whether a tool supports a given feature. *Multi-tenant:* can interact with hypervisors. *CB:* controller benchmark. *SB:* switch benchmark. *OF traffic generation rate:* OF message sending behavior (best-effort, distribution).

Tool	Multi-tenant	CB	SB	OF traffic generation rate
OFTest [14]			✓	best-effort
OFCProbe [15]		✓		best-effort, distribution: pre-defined (Normal, ChiSquared, Exp., Poisson)
CBench [10]		✓		best-effort
OFLOPS [16]			✓	best-effort: traces
hvbench [17]	✓			distribution: pre-defined (Exp.)
OFBench [18]			✓	constant rate, e.g., 64 Kilo bit to 1 Giga bit per second
perfbench	✓	✓	✓	pre-defined distribution: pre-defined (Uniform, Exp., Weibull) and custom

metrics of hypervisors, i.e., control plane processing latency, CPU utilization and multi-tenant context switching.

The other type of OF benchmark tools are controller benchmarks which include CBench [10], OFCProbe [15], and OFBench [18]. CBench is written in C++ and supports OF version 1.0, while OFCProbe is written in Java and supports OF versions 1.0 as well as 1.3. Both tools emulate multiple OF switches and evaluate the performance of controllers in terms of control plane throughput and latency. OFCProbe can generate different OF control message types with several distributions. CBench, however, can only generate control messages in a sequential manner, i.e., after sending a message request, it waits for a message reply in order to send the next request. This send behavior does not entirely map the traffic behavior in multi-tenant SDN networks. OFBench [18] is designed to reveal performance details of switches, like buffer size, or the time a packet takes through the pipeline.

With respect to traffic generation, *perfbench* generates OF message rates on a per-second basis, with flexibly definable sending intervals and burst sizes, in contrast to most available tools that generate OF messages in a best-effort manner or pre-defined distributions only.

C. Perfbench Tool Features

The *perfbench* tool supports synchronous OF messages, i.e., requests expecting a reply (e.g., `OFPC_PORT_STATS`, `OFPT_ECHO_REQUEST`, `OFPT_FEATURES_REQUEST`), and asynchronous OF messages (e.g., `OFPT_PACKET_IN`, `OFPT_PACKET_OUT`, `OFPT_FLOW_MOD`). For synchronous messages, the control latency is measured as the time it takes from issuing the request until receiving the reply.

For asynchronous messages, the latency calculation is different: For `PACKET_IN`, there are two modes. In case of using an existing OF switch, *perfbenchDP* sends UDP data packets for each tenant via its data plane connection. The latency is then calculated as the time it takes from issuing the UDP data packet until the `OFPT_PACKET_IN` is received at *perfbenchCP*. In case of emulating OF switches, *perfbenchDP* generates the `OFPT_PACKET_IN` messages and the latency is calculated as the time difference between when an `OFPT_PACKET_IN` is sent by *perfbenchDP* until it is received by *perfbenchCP*.

`PACKET_OUT` has two modes as well. First, *perfbenchCP* sends an `OFPT_PACKET_OUT` containing a UDP packet. When using an existing OF switch, after having received the `OFPT_PACKET_OUT`, the switch forwards the encapsulated packet on the DP to *perfbenchDP*. Here, we measure the time it takes from sending the `OFPT_PACKET_OUT` until receiving the UDP data packet at *perfbenchDP*. Second, without an existing OF switch, *perfbenchDP* receives the `OFPT_PACKET_OUT`, which marks the measurement time.

For `OFPT_FLOW_MOD`, the latency is calculated for each tenant as the time difference between when a `OFPT_FLOW_MOD` is sent by *perfbenchCP* until it is received by *perfbenchDP*.

In addition, *perfbench* can set the `TCP_NODELAY` flag for a specific TCP connection. In particular, setting `TCP_NODELAY` disables NAGLE's algorithm: NAGLE aggregates more data to reduce the packet overhead per TCP packet. However, while NAGLE's algorithm improves network performance, as we will see, the aggregation of packet content can lead to higher latency per packet: we argue that this is particularly undesired in vSDNs in latency critical use cases. To the best of our knowledge, this feature has not been investigated so far by any existing SDN performance measurement study.

D. Measurement Setup and Test Cases

We will use *perfbench* to explore the performance implications of the network hypervisor. Fig. 4 shows the measurement setup we will use throughout this paper. Three PCs are used to investigate the hypervisor performance benchmarks in this paper. The left PC runs *perfbenchCP* and *perfbenchDP*, the middle PC runs the SDN hypervisor, and the right PC runs an Open vSwitch (OvS) [19] instance, in cases where *perfbenchDP* does not emulate the data plane part. *perfbenchCP* is connected to the hypervisor PC, and the hypervisor PC is connected to the OvS PC. The *perfbenchDP* is connected via a dedicated line to the data plane part of the OvS PC.

The 3 PCs feature 16 GB RAM and 4 physical CPU cores (8 with Hyperthreading): Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz. All PCs run Ubuntu 14.04.5 LTS with the kernel 3.19.0-26-generic x86_64.

We examine two state-of-the-art SDN hypervisor implementations, namely *FlowVisor (FV)* [9] and *OpenVirteX (OVX)* [20]. We use the code from their latest GIT branch (FV:1.4-MAINT and OVX:master). We configure *perfbench* according to the hypervisors' specifics. When used with OVX, *perfbenchDP* uses artificial unique MAC addresses per tenant: a pre-requisite for the operation of OVX. For FV, such a setting is not necessary.

TABLE II: Measurement configurations.

Hypervisor	OF Message Type	Tenants	Switch emulation	Message Rate	TCP_NODELAY
FV/OVX	OFPT_PACKET_IN	1	No	10k,20k,30k,40k, 50k	0
FV/OVX	OFPT_FLOW_MOD	1	Yes	1k-30k	0/1
FV	OFPT_FLOW_MOD	5:20	Yes	100 per tenant	1
OVX	OFPT_FLOW_MOD	25:100	Yes	100 per tenant	1
FV	OFPT_PACKET_IN	1	Yes (5:20)	100 per switch	1
OVX	OFPT_PACKET_IN	1	Yes (25:100)	100 per switch	1

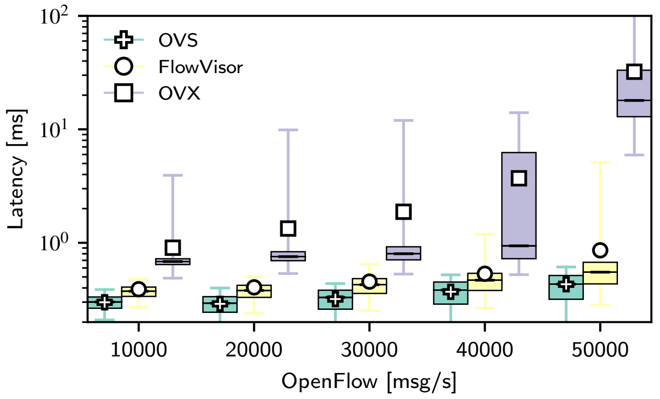


Fig. 5: Hypervisor overhead, OFPT_PACKET_IN, 10K - 50K.

Table II provides an overview of all conducted measurements. Single-tenant as well as multi-tenant measurements are conducted for a range of rates and TCP_NODELAY settings. Every setup is repeated at least 10 times for a duration of at least 30 seconds. Being interested in the steady-state performance, we omit the first and last 10 seconds from the data analysis.

The SDN hypervisor performance is evaluated in terms of control plane latency and CPU utilization, and compared against different message rates as well as number of tenants and switches. Further resources such as memory, which can also impact the overall performance, are out-of-scope of this study, as we always ensured that enough memory is available. Note, however, that a deeper analysis of memory usage is an interesting measurement aspect for future work. We measure four fundamental message types: We consider (1) asynchronous OFPT_PACKET_IN , (2) asynchronous OFPT_PACKET_OUT and (3) asynchronous OFPT_FLOW_MOD messages, whose performance is most relevant in OF-based SDN networks, e.g., to ensure short flow setup times. In terms of synchronous messages, we consider (4) OFPC_PORT_STATS: these are used by SDN applications to collect port statistics, e.g., for load balancing or congestion-aware routing.

IV. Performance Impact

In order to better understand the performance impact of the network hypervisor, we conducted a number of experiments; we report on the most important empirical insights.

How much overhead do network hypervisors add to the performance? We first investigate the performance overhead induced by the indirection via the hypervisor, see Fig. 5. The evaluation considers a showcase setting where OFPT_PACKET_IN messages arrive at rates from 10k to 50k messages per second. approaching the maximum rate at which the hypervisors can process this OpenFlow (OF) message type on the used computing platform. The performance is considered in terms of the control plane latency.

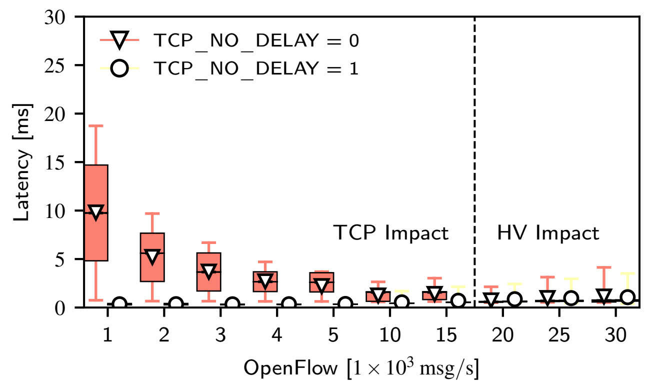
Due to the addition of extra intermediate network processing, a significant control latency overhead can be observed for both FlowVisor (FV) and OpenVirteX (OVX) for 10k messages: FV results in an average latency of $1\,\mathrm{ms}$, compared to $0.3\,\mathrm{ms}$ in a switch-only scenario; with $5\,\mathrm{ms}$ OVX adds even more latency. Note that the switch-only scenario results in sub-millisecond control latency for all following experiments.

While the latency is relatively increased by an order of magnitude, in absolute terms, the latency may still seem small. However, especially in latency-critical environments, such latency values may already be unacceptable, or at least introduce unacceptable uncertainties. Low latency communication is a primary metric for building data center and rack-scale networks [21], [22]. Human-computer interaction and haptic applications similarly show that people react to small differences in the delay of operations where milliseconds are critical. Indeed, according to [23], even slightly higher web page load times can significantly reduce visits from users and directly impact revenue.

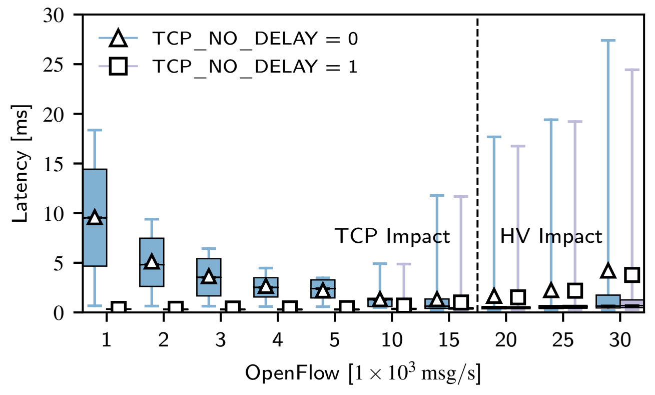
How does the tenant's controller impact the hypervisor performance? We next investigate the impact of the tenant's controller behavior by enabling the TCP_NODELAY setting on the controller's operating system. This means that the tenant's controller sends the control messages to the hypervisor as soon as they are ready, thus avoiding possible aggregation of OF messages in TCP packets. The performance of both hypervisors is evaluated with OFPT_FLOW_MOD messages, using TCP_NODELAY = 0 and 1 at the tenant's controller, as shown in Fig. 6. The measurement is carried out at message rates between 1k and 30k per second. Note that, for OVX we had to disable a flow table lookup inside OVX's code[2]. This part of the code holds a copy of the virtual switch flow table; however, it was diminishing the performance of OFPT_FLOW_MOD messages.

For both hypervisors, Fig. 6 shows that for small rates, from 1k up to 15k, the control latency is affected by the TCP aggregation algorithm that the tenant's controller uses. TCP_NODELAY = 0 shows higher control latency compared to TCP_NODELAY = 1, which is induced by the waiting time at the controller, to aggregate OFPT_FLOW_MOD messages into

[2]See openvirtex/elements/datapath/OVXFlowTable.java.



(a) FV, single tenant, `TCP_NODELAY`, `OFPT_FLOW_MOD`.



(b) OVX, single tenant, `TCP_NODELAY`, `OFPT_FLOW_MOD`.

Fig. 6: Impact of the `TCP_NODELAY` settings on the control plane latency of `OFPT_FLOW_MOD` messages for 1-30k message rates for FV and OVX.

TABLE III: Hypervisor control plane message throughputs (maximum OpenFlow message rate per second) at a single tenant with a single switch.

OF Message Type	FV	OVX
`OFPT_PACKET_IN`	58,170 ± 123	51,941 ± 579
`OFPT_PACKET_OUT`	57,980 ± 247	51,899 ± 301
`OFPT_FLOW_MOD`	39,975 ± 138	31,936 ± 402
`OFPC_PORT_STATS`	7,993 ± 22	199,937 ± 34

TCP packets. This can be easily mistaken as a performance problem of the hypervisor in SLAs. At higher rates, from 20k up to 30k, the control latency is influenced by the hypervisor processing overhead regardless of the TCP configuration of the tenant's controller. We could observe that the tenant's behavior can also impact the control performance, depending on the generated control rates.

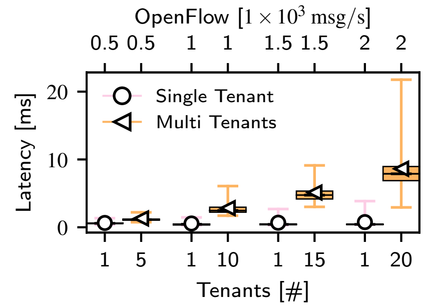
How does the control plane throughput depend on the specific network hypervisor? In this experiment, we investigate the control plane throughput (maximum rate per second) of two network hypervisors; FV and OVX. For this purpose, a single tenant (*perfbenchCP*) and a single switch (OvS) are used in order to avoid any cross effects to the experiment. *perfbench* is scheduled to increase the generated OF message rate: until losses start, hypervisor CPU overloads or control latency drastically bloats. Reaching any of these conditions, determines the maximum OF message throughput of the network hypervisor.

Table III shows that FV can provide a higher throughput for asynchronous `OFPT_PACKET_IN`, `OFPT_PACKET_OUT` and `OFPT_FLOW_MOD` messages, e.g., FV can support ~7k `OFPT_PACKET_IN` messages per second more than OVX. This can be explained by the data message translation process: OVX includes data plane packet header re-writing from a given virtual IP address, specified for each tenant, to a physical IP address used in the network. This is done in addition to control message translation. Note that for both hypervisors, the supported rate of `OFPT_FLOW_MOD` messages is lower than the received `OFPT_PACKET_IN` rate, which sets the upper bound for the flow setup (new connections) rate. For example, even though FV can support a rate of up to ~58k *packetin* messages, it can only respond with a maximum of ~40k *flowmod* messages per second, which means it can only setup new connections at a rate of 40k per second.

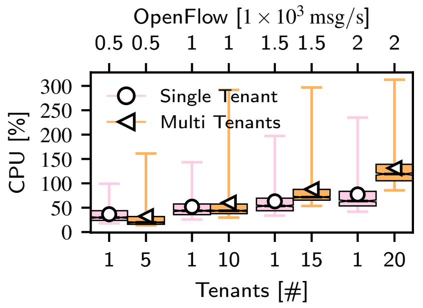
However, for synchronous `OFPC_PORT_STATS` messages, OVX shows much higher throughput (~200k messages per second) compared to FV (only ~8k messages per second). Since FV transparently forwards all messages to the switch, the switch becomes the bottleneck for port stats throughput. OVX uses a different implementation for synchronous messages: it does not forward the port stats to the switches, but rather *pulls* it from the switch, given a pre-configured number of times per second. The default pulling rate of OVX is 1 `OFPC_PORT_STATS` message per second. OVX replies on behalf of the switch to all other requests (using the same port statistics) , and hence, increases throughput in receiving `OFPC_PORT_STATS` messages. However, all tenants are limited by the `OFPC_PORT_STATS` pulling rate set by OVX. In fact, the "factual" `OFPC_PORT_STATS` throughput of OVX is equal to its statistics pulling rate.

How does the control latency and the hypervisor's CPU utilization depend on the number of tenants? Having identified several important performance factors of the network hypervisor in a single tenant scenario with different OpenFlow message types, we now move on to study how the vSDN performance depends on the number of deployed tenants: ideally, the performance provided for a single virtual network should scale (transparently) to multiple tenants.

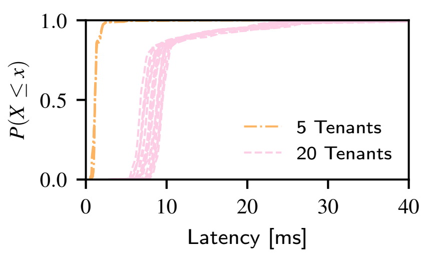
In order to study this question for the multi-tenant evaluation, we use `OFPT_FLOW_MOD` messages to check the case of multiple tenants' controllers (*perfbenchCP*) setting rules on the same network switch (*perfbenchDP*). We increase the number of tenants deployed on the hypervisor. Each tenant generates 100 `OFPT_FLOW_MOD` messages per second. In order to quantify the overhead of having multiple tenants, we compare the performance of the multi-tenant scenario to a single-tenant one, generating the cumulative rate of all deployed tenants:



(a) Control latency

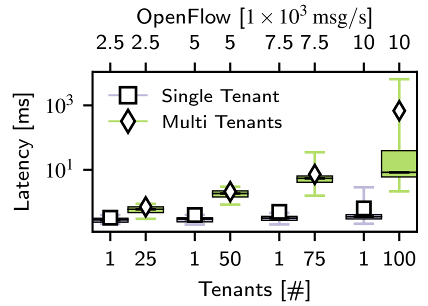


(b) CPU utilization

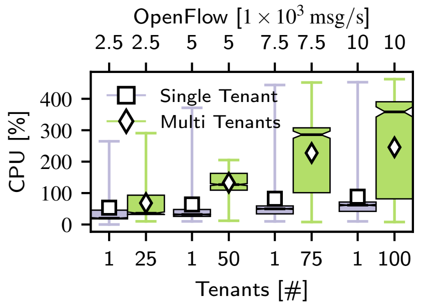


(c) Latency distribution

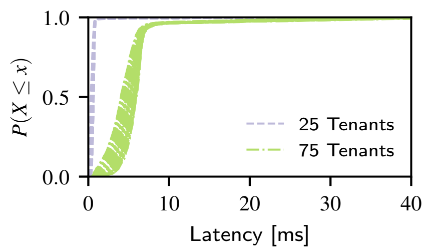
Fig. 7: FV performance, `OFPT_FLOW_MOD`, multi-tenants (5:20) vs. single tenant (baseline).



(a) Control latency (*Note the logarithmic scale of the y-axis.*)



(b) CPU utilization



(c) Latency distribution

Fig. 8: OVX performance, `OFPT_FLOW_MOD` , multi-tenants (25:100) vs. single tenant (baseline).

that is, we compare 5 tenants (each generating 100 messages per second) to a single tenant (generating 500 messages per second). This means also that with the number of tenants, also the workload on the hypervisor increases in terms of OpenFlow messages. We also provide an evaluation for the CPU consumption of the hypervisor in the presence of multiple tenants compared to a single tenant.

The impact of increasing the number of tenants (and the workload) on FV is shown in Fig. 7. We see that for FV, increasing the number of tenants results in higher control latency compared to the latency of a single tenant generating the same rate. For example, with 10 tenants and a total rate of 1k, the control latency increases to 3 ms compared to 1 ms for a single tenant generating 1k, as shown in Fig. 7a. Fig. 7b shows the CPU consumption of FV, which differs from the single tenant case starting from 15 tenants. This is due to the extra work (in terms of context switching, IO event handling, slice retrieval and verification) incurred by FV to handle the `OFPT_FLOW_MOD` messages from the different tenants. Note that due to load constraints, we cannot deploy more than 20 tenants with FV: to provide isolation between the tenants, FV implements a protection method that replicates the `OFPT_FLOW_MOD` message of one tenant to all other tenants, using a "forward to controller" action; however, this mechanism puts high loads on the control channel between FV and the switch, as well as on the CPU.

OVX shows similar effects when increasing the number of tenants, as seen in Fig. 8. We can deploy up to 100 tenants on OVX. We observe, in Fig. 8a, a significant increase in the control latency when using multiple tenants: e.g., with 100 tenants and 10k `OFPT_FLOW_MOD` rate, the median control latency increases to 10 ms, much more than in a single tenant scenario (generating a 10k rate). The CPU consumption of OVX also shows a drastic increase, e.g., an average of 230 % with 100 tenants compared to 100 % with a single tenant which is over a five-fold increase, as shown in Fig. 8b. Note that OVX is multi-threaded, hence can utilize more than 1 CPU core, compared to FV which is only single threaded.

The latency distribution of each tenant is shown in Fig. 7c for 5 and 20 tenants deployed on FV, and Fig. 8c for 25 and 75 tenants on OVX, using one run (of 30 seconds). For the maximum number of tenants for both hypervisors, there is a gap of around 3 to 5 ms in the mean latency between the worst and best case tenant. This means that the maximum observed latency can define the control latency guarantees that are provided by the hypervisor.

How does the control latency and the hypervisor's CPU utilization depend on the network size? Having observed the impact of increasing the number of tenants on the control performance of the hypervisors, next we investigate the impact of scaling the underlying network topology, i.e., number of switches, on the performance: intuitively, the hypervisor performance should scale with the number of switches (switch connections) in the same manner as it scales with the number of tenants (controller connections). For this purpose, we use OFPT_PACKET_IN messages originating from the switches (*perfbenchDP*) towards a tenant's controller (*perfbenchCP*.) that represent new flows arriving on different switches of the network. We increase the number of switches the same way as the number of tenants as in the previous experiment: 5:20 for FV and 25:100 switches for OVX. Each switch generates 100 OFPT_PACKET_IN messages per second (thus increasing the workload with the number of switches). We also compare the performance of the multi-switches scenario to a single switch scenario generating the cumulative rate of all switches.

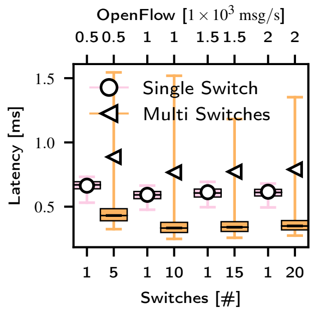
The performance of FV and OVX with respect to varying the number of switches and the workload are shown in Fig. 9 and Fig. 10, respectively. Surprisingly for FV, the mean control latency and mean CPU consumption are not impacted as much by increasing the number of switches as one might expect, e.g., with 20 switches, the mean control latency increases with $800\,\mu sec$ while the CPU utilization shows no clear overhead due to processing multiple switches compared to a single switch generating the same total OFPT_PACKET_IN rate. OVX leads to similar observations for the control latency performance for 25 and 50 switches. Note, however, the increasing latency variety in case of multiple switches. The reason here is that *perfbench* sends all messages with a minimum inter-arrival time equally distributed for one switch, whereas it accumulates all messages to be sent in case of multiple switches. Hence, more messages arrive at the hypervisor at the same time, leading to less CPU sleeps but more messages that need to be processed at the same time. For 75 and 100 switches, we can observe an additional latency of up to $5\,\%$. However, with multi-threaded CPU, OVX shows significant increase in the CPU utilization that scales with the number of switches, e.g., at 100 switches, a mean CPU of 395% compared to 44% CPU utilization with a single switch generating the same rate. This means that OVX exploits the multi-threading feature in order to support more switch connections and to guarantee the same latency performance.

In general, we can observe that both hypervisors' performance are not impacted by increasing the number of switches in the same manner as increasing the number of tenants. This suggests that handling the abstraction of more slices (slice verification and context switching) has a significantly higher impact than handling the switch abstractions.

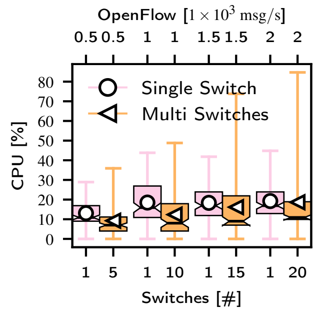
V. Related Work

The overheads and sources of unpredictable performance in cloud applications have been studied intensively over the last years. Many network virtualization architectures and prototypes have been proposed, leveraging admission control and bandwidth reservations and enabling tenants to specify absolute guarantees [2], [24], [25].

Also performance and measurement aspects of OF have been studied before in the literature. For example, Kuźniar et al. [26] report on the performance characteristics of flow table updates in different hardware OF switches, and highlight differences between the OF specification and its implementations, which may threaten correctness or even network security. Researchers have also considered the suitability of OF as a traffic measurement tool [27] (see [28] for a survey on the topic), and found that the quality of actual measured data can be questionable. Also other authors observed inconsistencies between bandwidth measurements results and a packet-based ground truth [29]. OF monitoring systems are implemented similarly to NetFlow, and accordingly, problems regarding insufficient timestamp resolution [30], [31], and device artifacts [32] also apply. Whereas an initial study investigated the impact of topology abstractions [33], the network hypervisor and especially its performance and possible overheads have received little attention in general so far.

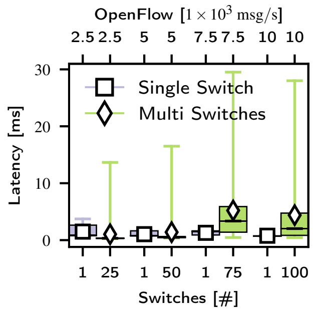


(a) Control latency

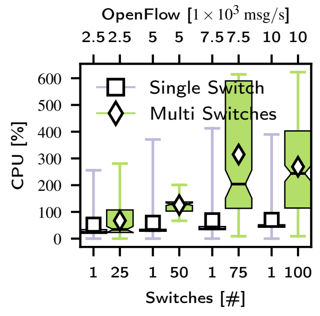


(b) CPU utilization

Fig. 9: FV performance, OFPT_PACKET_IN, multi-switches (5:20) vs. single switch (baseline).



(a) Control latency



(b) CPU utilization

Fig. 10: OVX performance, OFPT_PACKET_IN, multi-switches (25:100) vs. single switch (baseline).

VI. Conclusions and Discussion

Our empirical results suggest that in order to understand cloud application performance and make executions more predictable, we need to account for the network hypervisor in our models. In particular, our initial experiments show that the supported number of tenants per network hypervisor instance varies drastically depending on the hypervisor artefact. Different hypervisors show different performance: not only because of different implementations, but also because of different design choices (e.g., forward everything to the network, such as in FV, *vs.* take over some replies, such as in OVX). We could also observe that in multi-tenant vSDNs, the performance impact differs depending on the number of controllers resp. tenants and the network size resp. number of switches per tenant.

We understand that our work is a first step, and there are several dependencies and parameters that require further investigation. For instance, measuring the performance of distributed hypervisor platforms is an important next step due to scalability reasons. Other next steps would include measuring the impact of unbalanced and dynamic workload distributions, as well as the study of mixtures of OpenFlow message types. In general, we hope that our results as well as the identified performance criteria can help researchers develop more refined performance models for multi-tenant SDNs, eventually leading to the design of more performant and predictable virtual networks. These findings may also help improving the choices for network hypervisors which better suit the tenant applications.

Acknowledgment

This work has been performed in part in the framework of the CELTIC EUREKA project SENDATE-PLANETS (Project ID C2015/3-1) funded by the German BMBF (Project ID 16KIS0473), and in part in the framework of the EU project FlexNets funded by the European Research Council under the European Union's Horizon 2020 research and innovation program (grant agreement No 647158 - FlexNets), and in part in the DFG ModaNet Project (grant No KE 1863/8-1). 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] J. C. Mogul and L. Popa, "What we talk about when we talk about cloud network performance," *ACM SIGCOMM CCR*, vol. 42, no. 5, pp. 44–48, 2012.

[2] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *ACM SIGCOMM CCR*, vol. 41, no. 4. ACM, 2011, pp. 242–253.

[3] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The only constant is change: incorporating time-varying network reservations in data centers," vol. 42, no. 4. ACM, 2012, pp. 199–210.

[4] C. Fuerst, S. Schmid, L. Suresh, and P. Costa, "Kraken: Online and elastic resource reservations for multi-tenant datacenters," in *Proc. of IEEE INFOCOM*. IEEE, 2016, pp. 1–9.

[5] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," *IEEE Internet Computing*, vol. 17, no. 2, pp. 20–27, 2013.

[6] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn," *ACM Queue*, vol. 11, no. 12, p. 20, 2013.

[7] libfluid. [Online]. Available: http://opennetworkingfoundation.github.io/libfluid/

[8] A. Blenk, A. Basta, L. Henkel, J. Zerwas, W. Kellerer, and S. Schmid, "perfbench: A tool for predictability analysis in multi-tenant software-defined networks," in *Proc. ACM SIGCOMM 2018 Conference on Posters and Demos*, New York, NY, USA, 2018, pp. 66–68.

[9] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, pp. 1–13, 2009.

[10] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *2nd USENIX Workshop Hot-ICE*. USENIX, 2012, pp. 1–6.

[11] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. of ACM IMC*. ACM, 2010, pp. 267–280.

[12] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *ACM SIGCOMM CCR*, vol. 39, no. 4. ACM, 2009, pp. 51–62.

[13] K. He, A. Fisher, L. Wang, A. Gember, A. Akella, and T. Ristenpart, "Next stop, the cloud: Understanding modern web service deployment in ec2 and azure," in *Proc. of ACM IMC*. ACM, 2013, pp. 177–190.

[14] "OFTest—Validating OpenFlow Switches." [Online]. Available: http://www.projectfloodlight.org/oftest/

[15] "Measuring EC2 system performance," http://goo.gl/V5zhEd.

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

[17] C. Sieber, A. Blenk, A. Basta, and W. Kellerer, "hvbench: An open and scalable sdn network hypervisor benchmark," in *Proc. of IEEE NetSoft*. IEEE, 2016, pp. 403–406.

[18] Y. Lin, Y. Lai, C. Wang, and Y. Lai, "Ofbench: Performance test suite on openflow switches," *IEEE Systems Journal*, vol. 12, no. 3, pp. 2949–2959, Sep. 2018.

[19] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, "Extending networking into the virtualization layer," in *ACM Workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.

[20] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow, "Openvirtex: Make your virtual sdns programmable," in *Proc. of the workshop on hot topics in software defined networking (HotSDN)*. ACM, 2014, pp. 25–30.

[21] P. L. Suresh, M. Canini, S. Schmid, and A. Feldmann, "C3: Cutting tail latency in cloud data stores via adaptive replica selection." in *Proc. of USENIX NSDI*, 2015, pp. 513–527.

[22] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: trading a little bandwidth for ultra-low latency in the data center," in *Proc. of USENIX NSDI*, 2012, pp. 19–19.

[23] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low latency via redundancy," in *Proc. of ACM CoNEXT*. ACM, 2013, pp. 283–294.

[24] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "SecondNet: A data center network virtualization architecture with bandwidth guarantees," in *Proc. ACM CoNEXT*, 2010.

[25] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. Guedes, "Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks," in *Proc. of WIOV*, 2011.

[26] M. Kuźniar, P. Perešíni, and D. Kostić, "What you need to know about sdn flow tables," in *Proc. of PAM*. Springer, 2015, pp. 347–359.

[27] L. Hendriks, R. d. O. Schmidt, R. Sadre, J. A. Bezerra, and A. Pras, "Assessing the quality of flow measurements from openflow devices," 2016.

[28] A. Yassine, H. Rahimi, and S. Shirmohammadi, "Software defined network traffic measurement: Current trends and challenges," *IEEE Instr. & Meas. Mag.*, vol. 18, no. 2, pp. 42–50, 2015.

[29] N. L. Van Adrichem, C. Doerr, and F. A. Kuipers, "Opennetmon: Network monitoring in openflow software-defined networks," in *Proc. of IEEE NOMS*. IEEE, 2014, pp. 1–8.

[30] J. Kögel, "One-way delay measurement based on flow data: Quantification and compensation of errors by exporter profiling," in *Proc. of ICOIN*. IEEE, 2011, pp. 25–30.

[31] B. Trammell, B. Tellenbach, D. Schatzmann, and M. Burkhart, "Peeling away timing error in netflow data," in *Proc. of PAM*. Springer, 2011, pp. 194–203.

[32] I. Cunha, F. Silveira, R. Oliveira, R. Teixeira, and C. Diot, "Uncovering artifacts of flow measurement tools," in *Proc. of PAM*, 2009, pp. 187–196.

[33] N. Deric, A. Varasteh, A. Basta, A. Blenk, and W. Kellerer, "SDN hypervisors: How much does topology abstraction matter?" in *Proc. of CNSM*, 2018, pp. 328–332.