Collaborative Traffic Measurement in Virtualized Data Center Networks

Houssam Elbouanani, Chadi Barakat, Guillaume Urvoy-Keller, Dino Lopez-Pacheco

Collaborative Traffic Measurement in Virtualized Data Center Networks

Houssam ElBouanani, Chadi Barakat
Université Côte d'Azur, Inria, France

Guillaume Urvoy-Keller, Dino Lopez-Pacheco
Université Côte d'Azur, CNRS/I3S, France

***Abstract*—Data center network monitoring can be carried out at hardware networking equipment (e.g. physical routers) and/or software networking equipment (e.g. virtual switches). While software switches offer high flexibility to deploy various monitoring tools, they have to utilize server resources, esp. CPU and memory, that can no longer be reserved fully to service users' traffic.**

In this paper we closely examine the costs of (i) sampling packets; (ii) sending them to a user-space program for measurement; and (iii) forwarding them to a remote server where they will be processed in case of lack of resources locally.

Starting from empirical observations, we derive an analytical model to accurately predict ($R^2 = 99.5\%$) the three aforementioned costs, as a function of the sampling rate. We next introduce a collaborative approach for traffic monitoring and sampling that maximizes the amount of collected traffic without impacting the data center's operation. We analyze, through numerical simulations, the performance of our collaborative solution. The results show that it is able to take advantage of the uneven loads on the servers to maximize the amount of traffic that can be sampled at the scale of a data center. The resulting gain reaches 200% compared to a non collaborative approach.

***Index Terms*—Traffic Monitoring, Packet Sampling, Collaborative Sampling.**

I. Introduction

Over the last decade, companies have put a lot of efforts into migrating their IT infrastructures to public or private clouds deployed on data centers facilities. These data centers use virtualization technologies to provide flexibility and easy IT operations (e.g. dynamic service deployment, network monitoring and measurement), hence moving from the traditional hardware-centric mindset to a software-centric one. The flexibility delivered by such approach comes however at the cost of a higher hardware/software stack complexity.

In practice, data centers involve a large set of physical servers connected by a physical network. Each server hosts a number of virtual machines (VMs) that typically connect to each other and to the rest of the data center using a virtual switch. Identifying and monitoring VMs' traffic at the virtual switch is simpler than doing it at hardware networking equipment. Indeed, when the traffic leaves the physical server, it is likely to be encapsulated (e.g., as in Openstack [1] or VMware NSX [2]), and split by multi-path mechanisms (e.g. ECMP), hence increasing the complexity of its identification and capture.

The question we address in this work is how to maximize the traffic that can be passively monitored on a set of physical servers hosting VMs while minimizing the impact that such monitoring tools might have on the data center's operation. We consider the case of Open vSwitch (OvS) [3], which is currently the most used virtual switching solution. OvS offers high performance packet forwarding, implements the OpenFlow protocol [4] for SDN integration, and is compatible with multiple state-of-the-art monitoring and traffic collection services. One such service is sFlow [5], which samples part of the traffic seen by the switch and collects metadata on the sampled traffic to be further analysed at a, possibly remote, collector.

sFlow is a good option when the CPU consumption of traffic capturing must be minimized, as it limits by design the set of instructions to be performed on-board by the virtual switch. However, sFlow must be carefully configured in order to avoid a negative impact on the performance of VMs hosted by the physical server, because hardware resources held by sFlow might be needed to execute VMs' operations and/or convey their traffic.

Our contributions in this work are the following:

- We trace the operations (capture, transfer, processing) performed by the system for packet sampling for the case of OpenvSwitch and sFlow;
- We analytically model the cost of these individual operations;
- We formulate an optimization problem that computes the maximum tolerated sampling rates in a multi-server infrastructure without disrupting its operation and devise a collaborative traffic monitoring approach.

The main idea behind our optimization problem is to make virtual switches within physical servers collaborate and perform traffic sampling in a distributed fashion when required. Indeed, if locally performing the entire sFlow processing is too costly for a virtual switch and the users' traffic might be affected, we demonstrate that it is still possible to sample the traffic in one virtual switch and forward it to a virtual switch in a different physical server with enough resources, which in turn will execute the sFlow traffic analysis.

The remainder of this paper is organized as follows. In section II we present recent results upon which our work is built. Section III presents our measurement-based analysis to identify the root causes behind the switching performance drop and evaluate its intensity. We then build on these experimental results to propose a model that optimizes the deployment of sFlow in Section IV, and discuss current and future works in Section V.

II. Related Work

Network traffic measurements in cloud environments combines two commonly studied problems:

a) Network measurement and debugging: The complex blend of virtual and physical network resources and their abstraction in virtualized environments hinder the tasks of measuring the traffic and debugging the virtual infrastructure for both tenants and cloud providers. Many studies have highlighted the problem in different scenarios (public and private IaaS clouds [6], SDN [7] and NFV [8]), and designed new frameworks to facilitate measurement [9], or to efficiently use legacy solutions [10]. Other works [11], [12] have exposed the performance penalties of network traffic measurement in virtual settings. They have shown that implementing a sampling-based measurement function (sFlow) into a virtual switch (OvS) can impact its forwarding performance with nonnegligible degrees of intensity. While they effectively give accurate enough estimations of the drops in throughput caused by measurement, they do not formulate a model to predict them. These works have set the ground for our study.

b) Optimizing network measurement tools: Performing measurement operations in large networks is a complex task. In [13], [14], the authors use mathematical programming to optimize the placement [14] and packet sampling rates [13] of NetFlow, a widely used measurement tool, under the constraints of resources availability in hardware routers. Another approach based on matrix completion was proposed in [15] to efficiently conduct monitoring and measurement in virtual wireless networks. While the methodologies and objectives in [13], [14], [15] are similar to ours, their solutions are not suitable for virtualized data center networks as this type of environments introduces new constraints (e.g. both the virtual networking equipment and the VMs share the same hardware resources) and the authors do not use a distributed approach to traffic measurement.

III. Sampling Cost Evaluation

The system under study consists of a set of OvS switches and an sFlow process on each of them. Traditionally, and for maximum performance, packet switching in OvS is done at the kernel space to avoid the costly context switching overhead[1]. To achieve this, OvS implements the datapath, a kernel module responsible for the switching operations. Its design is simple enough to receive packets, to match them against rules injected by the main OvS program, and to perform the corresponding actions in an OpenFlow-like fashion (output to a certain port, forward to a user space program, update certain header fields, etc.).

When packet sampling is turned on in OvS, an additional action is added to all OvS table entries to forward packets to the user space sFlow agent with a certain probability (i.e. the sampling rate). As the datapath resides at the kernel space and the sFlow agent at the user space, the communication between them must achieved through one of the IPC (Inter-Process Communication) protocols made available by Linux. In particular, OvS uses the Netlink interface to allow asynchronous communications: the datapath module samples each packet and wraps it in a Netlink message that is sent to sFlow. The operations involved in this exchange of Netlink messages are the most costly and therefore the ones that impact the performance of the physical server and the virtual machines it hosts.

Hence, in this work, we empirically evaluate both the cost of sampling packets in OvS and the cost of transmission to the sFlow agent. Following a set of experiments, we establish a model for such costs and explain the rationale behind it. This model will serve as foundation for our network-wide optimization of sFlow sampling rates whose objective is to maximise the total volume of traffic to capture and export by sFlow. Once the sFlow agent receives a sampled packet, the packet is trimmed, compacted, and concatenated to other sampled packets and statistics related to the ports of the virtual switch to create one sFlow packet that will be sent to a central collector. Since only the headers of the packets are sent by the sFlow agents to the sFlow collector, this traffic overhead is negligible compared to the user traffic.

A. Testing Environment

To empirically evaluate the different costs pertaining to the sFlow operations, we consider the testbed depicted in Figure 1. The server is a physical machine with 16GB of memory and 8 CPU cores at 2.13 GHz running a Fedora 29 distribution on a Linux kernel 5.0.6. We use KVM/Qemu as hypervisor to create two virtual machines to which we assign 1 CPU core, 2 GB of memory, and 1 virtual network interface card (vNIC) each. We use iPerf3 [2] to generate traffic between the two VMs.

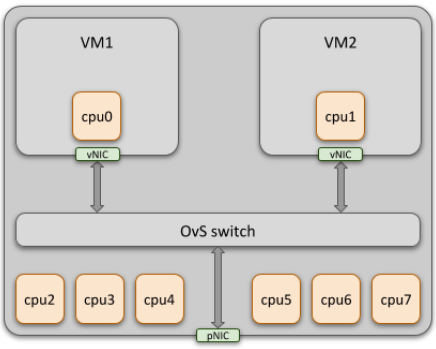


Fig. 1: Experimental testbed

In addition to the standard implementation of sFlow, we devised a program that emulates its relevant mechanisms (i.e. capturing sampled packets and reading their contents). Our program has two operational modes: either it simply receives

[1]We do not focus here on kernel bypassing techniques such as DPDK, where the switching logic is implemented in user space and no price is paid in terms of context switching.

[2]iPerf3 (v3.1.3) - The TCP, UDP and SCTP network bandwidth measurement tool, https://iperf.fr/en/

the sampled packets through the Netlink socket (setting 1); or it further reads the packet content (setting 2).

Our experimental methodology is as follows. Using iPerf, we generate TCP traffic from one VM to the other at maximum achievable speed for 1000 seconds and record the measured throughput each second. Meanwhile, the virtual switch is instructed to sample a certain fraction of the packets and send them to the user space, either to sFlow or to our custom program. The objective is to compare the different scenarios and quantify the drop in performance induced by user space to kernel space communications at different sampling rates. And since we are interested in cases where a virtual switch might outsource its traffic measurement task to another sFlow program, we will also analyse a setting where the virtual switch copies the packets and sends them through a physical port to the exterior as a way to emulate port mirroring to a remote server.

B. Experimental Results

Figure 2a shows the evolution of the measured iPerf throughput over time between the VMs for the different settings at 100% packet sampling rate. Our results show that port mirroring, where do not leave the kernel space, has the lowest cost, decreasing the performance of the TCP transfer by 30%. We notice also that the communications between user space and kernel space heavily penalize the transfer throughput. Indeed, in setting 1, where packets are only forwarded to the user space, decreases the TCP throughput by 40%, while additionally reading the packets' contents, by up to 60% (setting 2). Moreover, all further packet processing executed by sFlow (i.e. trimming, aggregating and exporting to the sFlow collector) is relatively inexpensive as it only decreases the throughput by 3%.

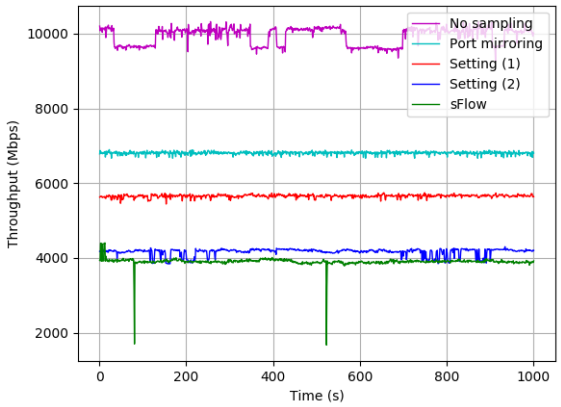
Figure 2b shows the average measured iPerf throughput at different sampling rates. The drop in performance depends on the sampling rate and closely matches the following law:

$$T_m(s) = \frac{T_m(0)}{1 + \alpha s},$$

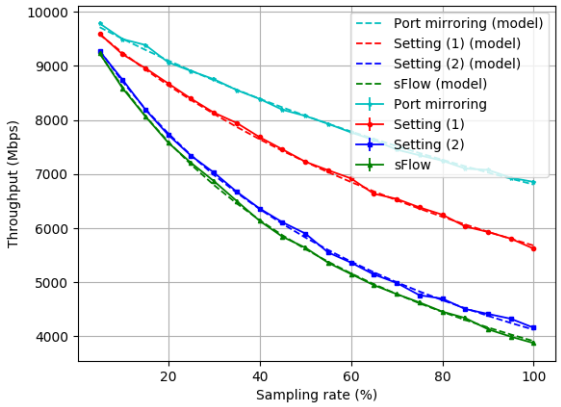
with high confidence indicators ($R^2 = 99.5\%$), where $T_m(s)$ is the maximum achievable throughput at a sampling rate $s \in [0, 1]$, and α a coefficient corresponding to the cost of the sampling operation (from our experiment dataset, this is equal to 0.75 for setting 1; 1.41 for setting 2; 1.54 for full sFlow; and 0.46 for port mirroring). Therefore, sending traffic at a certain rate T will result in a throughput

$$T^* = \min\left(T,\ \frac{T_m(0)}{1 + \alpha s}\right).$$

This result motivates the analytical model described in the following section, which tries to capture the region where the performance of the server is not impacted by sampling, for both cases (i) sFlow in the physical server, and (ii) sFlow outside the physical server and reached by port mirroring.



(a) Throughput variation in time at 100% sampling rate for port mirroring and different user space programs.



(b) Maximum achievable iPerf throughput vs. sampling rate for port mirroring and different user space programs.

Fig. 2: Experiment results.

C. Analytical model for sampling costs

The main contribution of our experiments is to quantify how sampling and processing packets at a virtual switch impacts its switching performance. So far we have demonstrated that while only sampling packets introduces some cost, the main cost is due to the communication between the two logical regions of the operating system (i.e. user and kernel spaces). This is fundamentally caused by the additional CPU operations needed to wrap packets in Netlink messages and send them to a process in a different context. Since the computing resources are only available in finite quantities, the sampling operations will ultimately compete with the forwarding operations of a switch and result in performance drops.

To model this behaviour, we assume the existence of a quantity C that corresponds to the server's available resources for virtual switching, that we call the server's capacity. If forwarding each packet consumes 1 unit of capacity, then sending one packet per second to sFlow consumes α units of capacity, and mirroring a packet per second to a remote

server consumes β units of capacity. Therefore the average total cost of forwarding a volume V (in packets per second), sampling a fraction $s^{(l)} \in [0,1]$ to a local sFlow program, and mirroring a fraction $s^{(r)} \in [0,1]$ to a remote physical server is

$$(1 + \alpha s^{(l)} + \beta s^{(r)}) \cdot V$$

It follows that the data path of the virtual switch will not be impacted if this total cost does not exceed the server's capacity C. This imposes an upper bound V_m on the traffic volume seen by the virtual switch to stay within the server capacity in the presence of sampling, which is given by

$$V_m(s^{(l)}, s^{(r)}) = \frac{C}{1 + \alpha s^{(l)} + \beta s^{(r)}}$$

.

As we allow servers to either process locally the sampled packets or send them to another server (which can give better results as $\beta < \alpha$) and maybe both at the same time (depending on the availability of resources at remote servers), the traffic to be processed at any virtual switch is the sum of both components. For instance, in a data center with two servers 1 and 2 each seeing traffic with volumes V_1 and V_2 respectively, the total cost at server 1 is given by

$$(1 + \alpha s_{1,1} + \beta s_{1,2})V_1 + (\alpha s_{2,1} + \beta s_{2,1})V_2$$

where $s_{i,j}$ is the rate at which server i sends packets to server j's sFlow program. Note that in the second term (V_2), β is the cost of receiving the packet from the remote server (the reception cost β equals the sending cost) and α is the cost to send this packet to the local sFlow instance. This linear model for the cost highly simplifies the formulation of the optimization problem.

IV. Sampling Optimization

A. A linear programming formulation

Our ultimate objective is to get the most accurate and complete information on the network traffic using the sFlow instances running in the different servers. This can be achieved by finding the optimal way servers sample their packets and send them to either a local or a remote sFlow process. However, this can cause considerable drops in traffic throughput if the sampling rates are not carefully tuned. The optimization problem can be formulated differently depending on the objective function to reach (minimize resource utilization, minimize impact on traffic, maximize inference accuracy, etc.). For our formulation, we set as objective to maximize the total average volume of collected traffic (in packets per second) by the sFlow monitoring plane. To achieve this, we impose the following assumptions and hypotheses that hold in a practical scenario:

- $A1$: the sampling overhead must not impact user traffic;
- $A2$: switches can process packets locally or send them to remote servers (or maybe both and possibly to multiple servers at once);
- $A3$: sampled packets must not be processed more than once;
- $A4$: each server hosts one and only one virtual switch;
- $A5$: the physical network is a full mesh (each server can directly communicate with any other server in the data center without passing by intermediate virtual switches[3]);
- $A6$: user packets are seen by at most two switches (when they are sent from a source VM in a server to a destination VM in a different server); and
- $A7$: for packets seen by two servers, sampling is performed only at server hosting the source VM.

$A1$ expresses our objective of introducing sufficient sampling across the data center while not disrupting the normal operation of the virtual network. $A2$ is motivated by the fact that the cost for a server of sending a packets to a remote sFlow ($\beta = 0.46$) is less than the cost of processing it locally ($\alpha = 1.54$). Thus a server might be in a situation where it does not have enough capacity to process an acceptable fraction of its traffic but can offload it to servers with sufficient amount of resources. Note that with the possibility for a server to send its packets to multiple destinations for processing, packets might be redundantly treated multiple times and thus result in suboptimal sampling allocations. Thus we add $A3$ that, albeit easy to enforce in OvS switches, effectively forbids this behaviour. $A4$-7 are simplifying assumptions that enable an easy formulation and resolution of the problem but that still correspond to most if not all real scenarios. These assumptions enable us to reach our objective of demonstrating how our analytical model can be used to optimize sampling rates of sFlow across a data center by leveraging our models for the cost of sampling and the collaboration between switches via port mirroring.

Consider a data center with a set $\mathcal{S} = \{1 \dots n\}$ of n physical servers. For any two servers i and j (not necessarily distinct as traffic might remain in a server), let $V_{i,j}$ be the total volume of traffic in packets per second sent by VMs in server i to VMs in server j, and let $U_i \triangleq \sum_j V_{i,j}$ be the total volume of traffic that server i has to sample. Also define $s_{i,j} \in [0,1]$ as the rate at which server i samples and sends its traffic for processing in server j. Then, following from $A2$ and $A3$, a packet seen by i is sampled with probability $\sum_k s_{i,k}$, and costs $\alpha s_{i,i} + \beta \sum_{k \neq i} s_{i,k}$ at server i and $(\alpha + \beta) s_{i,k}$ at any server $k \neq i$ to which it has mirrored its traffic. Therefore the optimization problem can be formulated as:

$$\text{maximize} \qquad \sum_i \sum_k s_{i,k} U_i$$

subject to the constraints for all $i \in \mathcal{S}$

$$U_i + \sum_{j \neq i} V_{j,i} + \big(\alpha s_{i,i} + \beta \sum_{k \neq i} s_{i,k}\big) U_i + \sum_{k \neq i} (\alpha + \beta) s_{k,i} U_k \leq C_i,$$

$$\sum_k s_{i,k} \leq 1,$$

[3]Openstack or the NSX solution of VMware [2] rely on a full mesh topology.

Symbol	Meaning
$s_{i,k}$	sampling rate in k of a packet from i
$V_{i,j}$	volume of traffic sent from a VM in i to a VM in i
U_i	total volume of traffic that i is responsible for measuring
C_i	total capacity of i
α	cost of sending a packet to local sFlow
β	cost of mirroring or receiving a mirrored packet

TABLE I: Variables and constants of the optimization problem.

$$s_{i,k} \geq 0.$$

The first constraint essentially expresses that no server may exceed its capacity for forwarding and sampling, and directly follows from $A1$. In its left-hand side, the first two terms correspond to the cost of forwarding the server's own traffic; the next term models the cost of sampling it, which can be either sent to its own local sFlow process or offloaded to another server's one; and the last term models the cost of receiving and processing other servers' sampled traffic.

Note that with the assumption that packets can be sent to multiple external sFlow programs but they must not be processed more than once, the sum of all sampling rates for a certain server i is at most 100%, hence the second constraint.

The optimization problem can thus be formulated as a linear program, and can therefore be solved in polynomial time (and fairly fast in practice) by optimized known LP solvers (e.g., [16], [17]). However, the number of variables in n^2 can slow down its resolution for large data centers. This motivates us to add a pre-processing step to reduce the problem's size before using a solver by removing those servers for which the solution is intuitive and optimizing over the rest.

B. A simple reduction rule

Making the servers collaborate to increase the total collected traffic results in a better utilization of the data center's resources. However, for a server to take the charge of sampling the traffic of another switch, it must have already sampled the entirety of its own. This is always true because sampling a local packet will cost α units of capacity and will add one packet to the total collected traffic, while sampling a remote server's packet will cost $\alpha + \beta$ units for the same gain. Put formally,

$$\forall i, k = 1 \dots n, (s_{k,k} < 1 \implies s_{i,k} = 0)$$

The idea then is to partition the set of servers into two subsets: a subset P of servers that are capable of sampling all of their traffic, i.e., those k that satisfy

$$U_k + \sum_{j \neq k} V_{j,k} + \alpha U_k \leq C_k,$$

and a subset Q of servers that do not. Next, if $i \in Q$ and $k \in P$ then $s_{i,k} = 0$. With this, and with $p = \frac{|P|}{n}$ and $q = \frac{|Q|}{n} = 1 - p$, the number of variables left in the problem is down to $p \cdot q \cdot n^2 \leq \frac{n^2}{4}$. This simple reduction rule therefore gives a decrease in size by at least 75%.

C. Numerical evaluation

The collaborative approach is guaranteed to yield better results than the fully local one, but the gain gets significantly more interesting in data center situations where the traffic load distribution among servers is skewed, i.e. some servers have less traffic to process than others. To demonstrate this assertion, we conduct the following series of numerical simulations to evaluate the performance of our approach.

Consider a data center with a set $\mathcal{S} = \{1 \dots n\}$ of n physical servers. Suppose that all servers share the same capacity C but have different loads of traffic to forward and sample. In particular, suppose that the total volume of traffic $\mathcal{V}$ in the data center is randomly distributed among all servers following a geometric-like law with parameter p (i.e., server i gets a share proportional to p^{i-1} of $\mathcal{V}$ as long as it does not exceed its own capacity).

The gain is finally defined as the relative increase in total collected traffic from collaborative sampling compared to non-collaborative sampling where each server samples in the limit of its own capacity without mirroring traffic to other servers.

Figure 3 shows the gain for a fixed C and different values of p and $\mathcal{V}$. Note that for certain values of p and $\mathcal{V}$, the traffic load assigned to certain servers exceeds their capacities, thus making the problem unfeasible. This unfeasible set is the blank region in the heatmap. This is more likely to be the case when p (x-axis) is small (total load of the data center concentrated at few servers). When p increases, the load becomes more evenly distributed and it is thus possible to increase $\mathcal{V}$. This explains the parabolic shape of the frontier with the white region. As for the gain compared to a non collaborative approach (the heatmap colors), it approaches 0% when the traffic is low (small $\mathcal{V}$ values). Indeed, in such case, all the traffic can be sampled locally, and there is no need to collaborate. When the traffic increases and the load is not evenly distributed on the servers (intermediate values of p), it pays off to offload the traffic to be sampled on less loaded servers. The gain in this case can reach up to 200%. For these numerical simulations, the average gain of the collaborative approach compared to fully local sampling and measurement is around 13%.

V. Discussion and Future Works

A. Estimating the capacity of a server

The capacity C of a server is a virtual quantity that translates its spare computing resources into packet forwarding potential. In practice however, determining such quantity can be difficult, as the amount of unused CPU cycles can vary greatly over time. We have identified three approaches to periodically estimate its value up to a reasonable degree of accuracy:

- Predicting the capacity's value from current CPU usage statistics using controlled experimentation backed with statistical inference and machine learning algorithms;
- Dynamically estimating the capacity using active measurement tools (e.g., iPerf or ping);
- Passively inferring when a server's capacity is reached from drastic increases in latency and/or jitter.

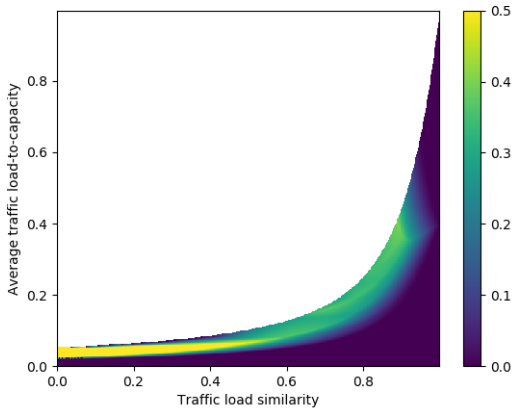


Fig. 3: Collaborative sampling gain relative to fully local sampling. The average traffic load to capacity is $\frac{\mathcal{V}}{nC}$, and the traffic dispersion is p.

Each of these approaches will be investigated in future works. In the present paper, the virtual switch in the testbed has a dedicated number of CPU cores and therefore the value of the capacity can be assumed to be static. This assumption can be used in production environments where physical servers may have dedicated CPU cores for virtualization tasks –and consequently for virtual switching– to guarantee client SLAs. In this case, determining the capacity once is sufficient and can be achieved by running active measurement tests during low-utilization periods.

B. Implementation and validation

Although the root cause analysis of the cost of sFlow is highly experiment-based, the solution that we have proposed to limit it by introducing collaborative traffic measurement has been only theoretically evaluated. We have effectively investigated the implementability of the different assumptions (e.g., packet sampling at multiple switches without redundancy) in virtual networks, and the next step in our research is to develop a framework for implementing the solution in production scenarios and use it to validate our approach in large-scale settings.

VI. Conclusion

In this paper we have investigated the impact of sFlow's sampling mechanisms on a virtual switch's forwarding performance. In particular, we have successfully identified the root causes behind this impact. At the system level, the cost comes from the forwarding of sampled packets to user space programs; at the hardware level, it comes from additional CPU cycles needed for copying packets' contents and sending them to different processes' memory contexts.

We have also proposed a model to predict this impact with high degrees of accuracy. Although it was motivated by a dataset from a single-server testbed, our model can be safely generalized to multi-server situations. This model was used to formulate and solve a mathematical problem to optimize the utilization of data center resources for traffic measurement tasks by making the servers collaborate with each other by mirroring their sampled traffic. A numerical simulation-based evaluation of performance was conducted to highlight situations where our approach significantly increases the volume of measured traffic compared to purely local sampling.

References

[1] https://www.openstack.org/

[2] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, P. Ingram, E. J. Jackson, A. Lambeth, R. Lenglet, S.-H. Li, A. Padmanabhan, J. Pettit, B. Pfaff, R. Ramanathan, S. Shenker, A. Shieh, J. Stribling, P. Thakkar, D. Wendlandt, A. Yip, R. Zhang: "Network Virtualization in Multi-tenant Datacenters." NSDI 2014, pp. 203-216.

[3] B. Pfaff, J. Pettit, T. Koponen, E. J. 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," NSDI 2015, Berkeley, pp. 117-130.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks." *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69-74, Mar. 2008.

[5] "InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks," RFC 3176, 2001.

[6] A. Ciuffoletti, "Monitoring a Virtual Network Infrastructure: An IaaS Perspective," SIGCOMM Computer Communication Review, vol. 40, no. 5, pp. 47-52, 2010.

[7] T. Zhang, M. Chen, X. Wei, B. Chen and C. Hu, "SDNMS: A software defined network measurement system for NFV networks," China Communications, vol. 16, no. 4, pp. 59-74, April 2019.

[8] Y. Ran, X. Wu, P. Li and Y. Luo, "Dynamic Virtual Measurement Function scheduling in software-oriented measurement environment," IEEE International Conference on Communications (ICC), Paris, 2017, pp. 1-6.

[9] S. Clayman, A. Galis and L. Mamatas, "Monitoring virtual networks with Lattice," 2010 IEEE/IFIP Network Operations and Management Symposium Workshops, Osaka, 2010, pp. 239-246.

[10] A. Roy, D. Bansal, D. Brumley, H. K. Chandrappa, K. Harish, P. Sharma, R. Tewari, B. Arzani and A. C. Snoeren, "Cloud Datacenter SDN Monitoring: Experiences and Challenges," Internet Measurement Conference, Boston, MA, 2018, pp. 464-470.

[11] K. Gogunska, C. Barakat, G. Urvoy-Keller and D. Lopez-Pacheco, "On the Cost of Measuring Traffic in a Virtualized Environment," IEEE 7th International Conference on Cloud Networking (CloudNet), Tokyo, 2018, pp. 1-6.

[12] V. Mann, A. Vishnoi and S. Bidkar, "Living on the edge: Monitoring network flows at the edge in cloud data centers," Fifth International Conference on Communication Systems and Networks (COMSNETS), Bangalore, 2013, pp. 1-9.

[13] G. R. Cantieni, G. Iannaccone, C. Barakat, C. Diot and P. Thiran, "Reformulating the Monitor Placement Problem: Optimal Network-Wide Sampling," 40th Annual Conference on Information Sciences and Systems, Princeton, NJ, 2006, pp. 1725-1731.

[14] M. Bouhtou, S. Gaubert and G. Sagnol, "Optimization of Network Traffic Measurement: A Semidefinite Programming Approach," International Conference on Engineering Optimization, Rio de Janeiro, 2008.

[15] X. Wang, C. Xu, G. Zhao, K. Xie and S. Yu, "Efficient Performance Monitoring for Ubiquitous Virtual Networks Based on Matrix Completion," IEEE Access, vol. 6, pp. 14524-14536, 2018.

[16] A. Makhorin, GLPK (GNU Linear Programming Kit) v4.47.1, Moscow Department for Applied Informatics, Moscow Aviation Institute, 2011.

[17] International Business Machines Corporation, IBM ILOG CPLEX Optimization Studio v12.4, Armonk, 2011.