On the Cost of Measuring Traffic in a Virtualized Environment

Karyna Gogunska, Chadi Barakat, Guillaume Urvoy-Keller, Dino Lopez-Pacheco

On the Cost of Measuring Traffic in a Virtualized Environment

Karyna Gogunska, Chadi Barakat
Université Côte d'Azur, Inria, France

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

***Abstract*—The current trend in application development and deployment is to package applications and services within containers or virtual machines. This results in a blend of virtual and physical resources with complex network interconnection schemas mixing virtual and physical switches along with specific protocols to build virtual networks spanning over several servers. While the complexity of this set-up is hidden by private/public cloud management solutions, e.g. OpenStack, this new environment constitutes a challenge when it comes to monitor and debug performance related issues. In this paper, we introduce the problem of measuring traffic in a virtualized environment and focus on one typical scenario, namely virtual machines interconnected with a virtual switch. For this scenario, we assess the cost of continuously measuring the network traffic activity of the machines. Specifically, we seek to estimate the competition that exists to access the physical resources (e.g., CPU) of the physical server between the measurement task and the legacy application activity.**

***Keywords*—sFlow, IPFIX, measurement, datacenter, open vswitch, virtualization.**

I. Introduction

Modern IT infrastructures heavily rely on virtualization with the so-called public or private clouds and cloud management tools such as OpenStack[1]. The typical path taken by a packet sent from a virtual machine (VM) in a data center illustrates the complexity of such a set-up. The packet crosses the virtual network interface card (NIC) of the VM to reach a virtual switch where it is encapsulated, e.g., in a VXLAN tunnel, either to reach the remote tunnel endpoint (switch) before being delivered to the destination VM, or to a virtual router before leaving the virtual LAN. This complexity and blend of software and hardware equipments raise the difficulty to monitor and debug performance issues in such a virtualized environment. Monitoring and capturing traffic at the departure or arrival of its journey, i.e., at the first/last virtual switch, reduces the complexity of the task for the cloud provider or manager. It also allows to limit the impact on the physical switches that interconnect the racks. Still, it should be done carefully as the networking device (virtual switch) and the VMs share the resources (CPU, memory) of the same physical server. This key question has been overlooked in previous studies, so that in this paper we shed light on the interplay between measuring and delivering traffic in a physical server with VMs and a virtual switch.

We follow an experimental approach for the purpose of our study. We set up a testbed around an Open vSwitch (OvS)[2] switch, which is arguably the most popular virtual switch nowadays, natively integrated in OpenStack and VMware. We consider the two typical levels of granularity of traffic collection tools, namely the packet level monitoring offered by sFlow [1] and the flow level monitoring offered by IPFIX [2]. We aim at answering the following questions:

- What is the resource consumption (in terms of CPU) of those measurement tools as a function of the configuration parameters, e.g., sampling rate, granularity of measured data, and report generation time?
- What is the trade-off between the measurement accuracy and the system performance, i.e. the impact of measurement on the flows being measured (e.g., completion time, throughput)?

Our contribution can be summarized as follows: (i) We explore the system resources consumption of typical monitoring processes run in the virtual switches; (ii) We demonstrate the existence of a negative correlation between the measurement tasks and the operational traffic in the network under flow and packet level monitoring; (iii) We show that such an effect is not caused by a lack of system resources.

In the next Section we overview the state of the art in the field. In Section III we describe our testbed. We present the results for the two use cases of sFlow and IPFIX in Sections IV and V, respectively. Section VI concludes the paper with a discussion and ideas for future research.

II. Related Work

SDN (Software Defined Networking) is gaining momentum in modern data centers [3]. As such, a significant amount of works have focused on measuring SDN networks. Monocle [4] injects probe packets in the network to check the consistency between the controller view and the actual SDN rules. VeriDP [5] and Everflow [6] use a passive approach to solve the same problem, either by capturing the control messages between the controller and the switches for the former, or by capturing specific packets at different routers for the latter. DREAM [7] distributes the measurement tasks over several SDN switches depending on the utilization of the flow rule table and the targeted accuracy. UMON [8] modifies the way OvS handles SDN flow tables to decouple monitoring

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

[2]http://openvswitch.org/

from traffic forwarding, hence building specific monitoring tables. Hansel and Gretel [9], [10] focus on troubleshooting the control plane of OpenStack, i.e. the REST calls made between the OpenStack modules. A data driven approach is followed by these tools to pinpoint the origin of system errors. Virtual Network Diagnosis as a Service [11] offers to monitor tenants' network applications experiencing problems, by capturing traffic at the virtual switches (relying on port mirroring) and accessing the measurements through an SQL-like interface.

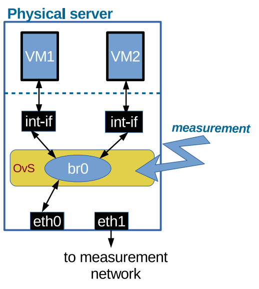
None of these works evaluate the cost of measurement in its broad sense neither the influence of the measurement plane on the data plane, which is our primary purpose in this work.

III. Test environment

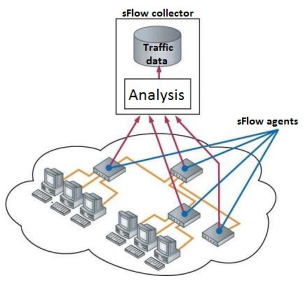
A. Testbed

We consider a typical scenario with VMs interconnected by an OvS switch located in the same physical server, see Fig. 1a. The physical server runs a Ubuntu 16.04 Operating System, and has 8 cores with no Hyper-Threading support at 2.13 GHz, 12 GB of RAM and 4 GigaEthernet ports. We use KVM to deploy the VMs (centOS) configured with 1 Virtual CPU (VCPU) and 1 GB of RAM each. We conduct experimentation with $\{2, 4, 6, 8\}$ VMs to investigate how measurement tasks behave under different conditions of server CPU occupancy: from low utilization of physical resources (2 VMs with 2 cores dedicated to VMs) to a case where all eight cores are occupied (8 VMs). One pair of VMs is acting as source and destination and the amount of traffic generated is directly related to number of VMs (more senders – more traffic).

For traffic monitoring, we consider legacy tools natively supported by OvS, namely sFlow [1] and IPFIX [2]. The measurement collector is placed outside the server, in a remote node across the network.



(a) Our experimental testbed



(b) sFlow architecture[3]

Fig. 1: Testbed and measurement process

sFlow is considered to be a scalable, light-weight solution for network monitoring [1]. It is a packet-level technology for monitoring traffic in data networks. It performs monitoring using a sampling mechanism (1 out of n), which implies exporting a packet header of every n-th packet over the network to a remote collector, which further builds statistics on the monitored network traffic. With sFlow, no computation is made at the switch, which should limit its CPU consumption. Fig. 1b portrays how sFlow works.

In contrast to sFlow, IPFIX is a flow-based measurement tools, as it performs aggregation of sampled packets in flows on board of the switch, and reports statistics on flows rather than simply copying and exporting the headers of the sampled packets as in sFlow. Having said that, IPFIX can also operate at the packet level by disabling flow aggregation (by setting either flow caching or active timer to zero - see Section IV for details), thus reporting per-packet statistics. To some extent, one can see sFlow as an extreme version of IPFIX where aggregation on board is disabled. Based on that, we start by evaluating the overhead of sFlow in Section IV, then we move in Section V to a comparison between the two measurement approaches in terms of their load on the physical server and the impact they incur for the application data plane.

B. Traffic workload

In our testbed, traffic between the VMs is produced with two tools: *flowgrind* [12] and *iperf3* [13]. Each experiment is run 10 times to smooth the results. In general, the different runs of each experiment are producing close results, as we operate in a closed and controlled environment.

1) Flowgrind: This is an advanced TCP traffic generator for testing and benchmarking Linux, FreeBSD, and Mac OS X TCP/IP stacks. We use it for its ability to generate TCP traffic following sophisticated request/response patterns. Indeed, in flowgrind, one single long-lived TCP connection is supposed to mimic transactional Internet services as the Web. Hence, a TCP connection in flowgrind consists of a sequence of requests/responses separated by a time gap between the requests. The request/response traffic model in flowgrind is controlled with the following four parameters: (i) Inter-request packet gap; (ii) Request/response size, (iii) Request/response size distribution, and (iv) Number of parallel TCP connections.

We use a fixed inter-request gap equal to 10^{-4}s in our experiments. Requests and responses are sent as blocks, with request and response messages fitting in one or several IP packets. While keeping the request size constant at 512B, we vary the response size (using a constant distribution, meaning that all responses have the same size) to achieve different rates, both in packets/s and in bits/s, as presented in Table I.

TABLE I: Flowgrind parameters used for traffic generation

Response size, bytes	Throughput, Mb/s	Throughput, packets/s
1024	80	16000
2048	160	26000
3072	240	40000
4096	320	45000

2) iPerf3 : We use iPerf3 to generate TCP traffic at maximum achievable rate (10 Gb/s) to investigate whether monitoring with sFLow may cause any kind of impact on the workload produced between VMs. For IPFIX and sFLow comparison we

[3]Source of the figure: https://www.juniper.net/documentation/en_US/junos/topics/example/sflow-configuring-ex-series.html

also generate UDP traffic at constant and controlled rate of 100 Mb/s to minimize data plane interference. The exploited traffic pattern is depicted in Table II.

TABLE II: iPerf3 parameters used for traffic generation

Protocol	Throughput, Mb/s	Throughput, packets/s
TCP	10000	25000
UDP	100	8600

IV. Measuring with sFlow

We focus in this section on sFlow. Its behavior is mainly driven by the following parameters:

- *Sampling rate*: Ratio of packets whose headers are captured and reported by sFlow.
- *Header bytes*: The number of bytes reported by sFlow for each sampled packet. The default value in our experiments, unless otherwise stated, is equal to 128 bytes. More than one header can be aggregated in one UDP datagram, depending on the configured header size.
- *Polling interval*: sFlow also reports aggregate port statistics to the collector. This parameter models the frequency at which sFlow reports these statistics.

A. Initial Experiment: measurement plane vs. data plane

We conducted a first experiment with a long lived TCP flow generated with iPerf3. The experiment lasts 600 seconds and the sampling rate is changed (or disabled) every 100 seconds. We plot the throughput achieved by iPerf3 in Fig. 2. Note that with zero packet sampled (i.e., no sFlow measurement), we can reach a throughput up to 10 Gb/s as TCP generates jumbo packets of 65 KB. It is then clear that monitoring with sFlow can influence the application traffic and that this influence depends on the level of sampling. Finding the right sampling rate to use is a trade-off between desired monitoring accuracy and expected application performance. In the next sections, we explore this trade-off in more details and try to understand whether this result is due to a lack of resources or to the implementation of sFlow in OvS.

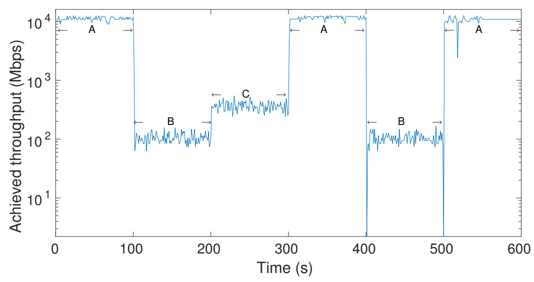


Fig. 2: iPerf3 throughput at different sampling rates of sFlow: no sampling (A), 100% sampling (B), 50% sampling (C).

B. Resources consumption and competition

In this section, we vary the sampling rate and observe the CPU consumption of sFlow summed over all CPUs (hence potentially from 0 to 800%) as OvS uses multi-threading and can run over different cores. Fig. 3 reports the results obtained with the flowgrind workload, as described in Table I. Note that the sampling rate is expressed in sFlow as the number of packets (the unsampled ones plus the sampled one) between each two consecutively sampled packets. This means that when $sampling = 1$, we sample every packet (100% rate), $sampling = 2$ we sample one packet out of two (50% rate), and so on.

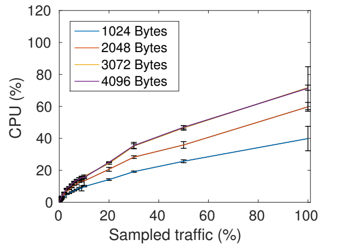
We can make two observations in Fig. 3. First, the CPU consumption increases with both the sampling rate and the traffic rate, in line with intuition. Still, at 100% sampling, we observe a decrease of CPU consumption in some cases[4]. We defer the study of this phenomenon to the next section.

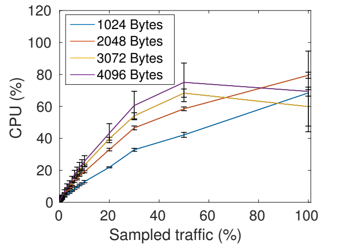
Second, while OvS is multi-threaded, a single thread is used to handle sFlow measurement task (as revealed by *pidstat*) and the utilization of the core where sFlow operates is high in our experiments. Considering high value and sublinear increase of CPU consumed by the tool at high sampling rates, one can wonder whether interference with monitored traffic exists. This interference is indeed visible in Fig. 4 when plotting the number of packets generated by flowgrind for the different sampling rates and for the different traffic profiles. We report the results for the case of 2 VMs and 4 VMs. The results for 6 and 8 VMs are similar to the 4 VMs case. Normally, in the absence of interference, this number should stay constant whatever the sampling rate is, which is not the case in the figure, especially when the sampling rate gets close to 100%. This decrease in the number of generated packets points to a possible interference caused by the CPU consumption of sFlow. Having less data packets at high sampling rates can also explain the sublinear trend of CPU consumption with sampling rate observed in Fig. 3.

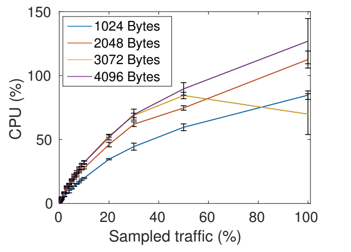
C. High sampling rate anomaly

To better understand the performance anomaly aforementioned, we looked at how the OvS measurement process was utilizing the available cores in the case of 4 VMs, where there should be enough resources for both measurement process and VMs generating traffic. We used *pidstat* to track the core utilized and its usage by sFlow. We observed that while there should be almost no competition between the iPerf3 (embedded inside single-core VMs) and OvS processes, as there are 8 cores in the server, the OvS process was regularly scheduled to a different core by the Linux scheduler. To check if this variable allocation results in suboptimal performance, we pinned each VM and the OvS process to a specific core using the *taskset* utility. Fig. 5a and Fig. 5b portray a single experiment for the case of 4 VMs (for traffic of 320 Mb/s and

[4]Note that due to the implementation of the sampling parameter in sFlow, there is no value between 50% (one packet out of two) and 100% (every packet) in our graphs.


(a) CPU consumption of sFlow, 2 VMs


(b) CPU consumption of sFlow, 4 VMs


(c) CPU consumption of sFlow, 6 VMs

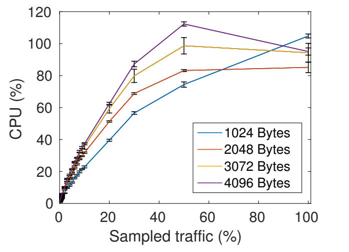
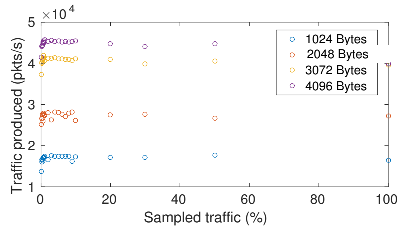

(d) CPU consumption of sFlow, 8 VMs

Fig. 3: CPU consumed by sFlow vs. sampling rate (flowgrind)


(a) Traffic generated for 2 VMs

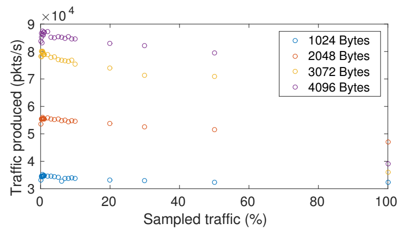

(b) Traffic generated for 4 VMs

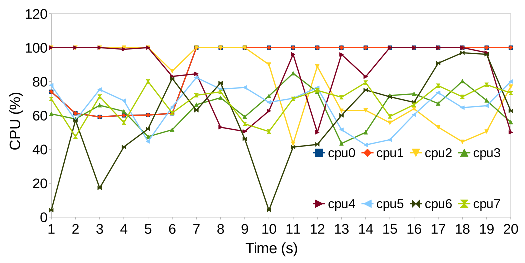
Fig. 4: Nb. of packets generated vs. sampling rate (flowgrind)

sampling rate of 100%), and shows a clear improvement in terms of core utilization variability. Still, the impact on user's traffic was observed to be similar. We can thus conclude that the OS scheduler is not the cause behind the performance problem we observed.

A better understanding of the phenomenon we observed would require a precise profiling of the OvS code, which is out of the scope of our work. Indeed, we suspect the bottleneck to manifest at the boundary between the kernel space where forwarding is done by OvS and the user space where sFlow operates, hence slowing down the rate of traffic going through both of them.

D. Varying sFlow parameters

The effect of the sampling rate on the CPU consumption has been investigated in the previous section. In this section, we


(a) OvS is **not** assigned to a specific core

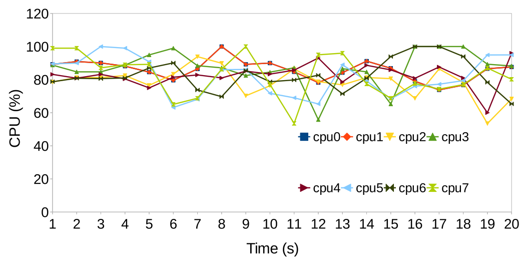

(b) OvS is assigned to a specific core

Fig. 5: CPU load with/without pinning OvS to specific core

extend the study to the other parameters of sFlow to show their impact as well. We cover in particular the impact of the header length and the frequency of interface statistics reporting, which are appended to packet samples by sFlow at the desired time interval. To evaluate influence of these two parameters, we show results for the set-up with four virtual machines, as in this case the physical server is under medium utilization: four cores are assigned to four virtual machines and the remaining four cores of the physical server are left for the proper operation of the hypervisor, OvS and the other system services.

1) Header length: sFlow reports contain the first N bytes of sampled packets. The sFlow implementation in Open vSwitch samples by default the first 128 Bytes of each packet.

We vary the header length and assess its impact on the system performance with the help of a flowgrind workload of response size equal to 1024 Bytes in our set-up of 4 VMs.

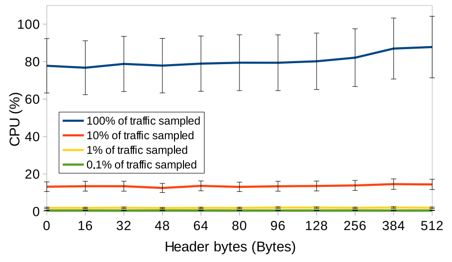


Fig. 6: CPU consumed by sFlow for different header lengths and different sampling rates

Several reports (packet samples or statistical reports) compose an sFlow datagram. Depending on its size, more or less reports may fit into one datagram. Maintaining fewer or more reports does not matter for the software switch, as finally they are to be encapsulated into one datagram. Moreover, the process of sending datagrams to the collector does not incur a computational burden in terms of CPU. It follows, and according to what we see in Fig. 6, that changing the header length does not impact the CPU utilization. In this figure, we explore values of header length up to 512 Bytes, which is the recommended maximum header length in most sFlow implementations. Next, and if not explicitly mentioned, we restrict ourselves to the default header length of 128 Bytes in our experiments.

2) Polling period: The polling parameter refers to the time interval (default 30s) at which sFlow appends to its reports aggregate traffic statistics (total-packets and total-samples counters) associated with the different interfaces of the virtual device, which can be either ingress or egress interfaces. These reports are usually piggybacked with sampled packet headers, but as sampling is random, and to avoid periods of no reporting, the sFlow agent can be configured to schedule polling periodically in order to maximize internal efficiency. According to sFlow developers, the polling interval should not have a big influence on CPU consumption. To confirm this statement, we performed experiments with varying polling intervals under different flowgrind workloads and sampling rates. Results were similar for the considered workloads. We report in Fig. 7 the case of a flowgrind workload of about 80 Mb/s. We can observe in the figure that varying the polling interval from 1 second to 30 seconds does not induce any additional CPU overhead. This is because the sFlow agent opportunistically inserts the counters into sFlow datagrams together with samples. If many packets are to be sampled, counters may not be even included as frequently as configured.

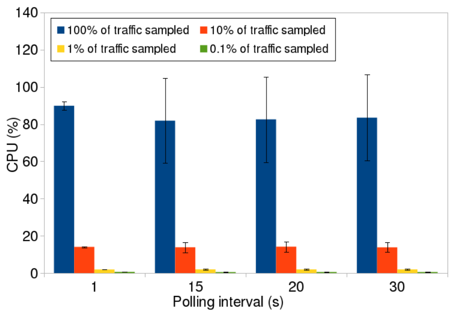


Fig. 7: CPU consumed by sFlow for different polling intervals

V. Measuring with IPFIX

As explained earlier, IPFIX and sFlow are two representatives of the two main classes of traffic monitoring approaches: the flow level approach and the packet level approach, respectively. sFlow was designed to limit the processing at the switch/router side by simply forwarding packet headers to the collector while IPFIX maintains a flow table to aggregate packets in flows, then reports on flows rather than on packets. This normally should entail a higher processing load but a smaller network footprint. Next, we compare the two tools. We configure sFlow with its default header length of 128 Bytes. The size of each IPFIX flow report is equal to 115 Bytes.

A. Running IPFIX without flow aggregation

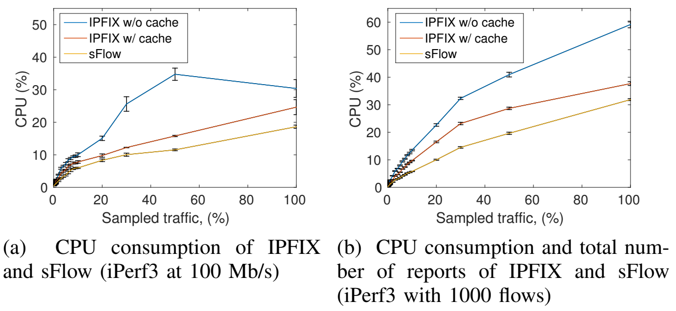
We first configured OvS to send one IPFIX flow record per packet sample to the collector. The task of IPFIX is thus similar to the one of sFlow in this case.

There exist two caching options for IPFIX in OvS:

- *cache active timeout* – maximum period for which IPFIX flow record is cached and aggregated before being sent;
- *cache maximum flows* – maximum amount of IPFIX flow records that can be cached at any time.

To ensure per-packet flow exporting, the caching feature of IPFIX is disabled and the active timers are set to zero.

As there is no flow aggregation in this specific experiment, we consider a scenario with one long run TCP flow produced by iPerf3 at a rate of 100 Mb/s. Results are reported in Fig. 8a where we compare sFlow to IPFIX for different sampling rates, both in terms of CPU consumption and total number of reports. We focus in this section only on the IPFIX w/o cache case. Clearly, and because of packet processing on-board (in the switch), IPFIX consumes more CPU than sFlow. The latter simply reports headers without on-board processing.



(a) CPU consumption of IPFIX and sFlow (iPerf3 at 100 Mb/s)

(b) CPU consumption and total number of reports of IPFIX and sFlow (iPerf3 with 1000 flows)

Fig. 8: IPFIX vs. sFlow resource consumption

Similarly to the case of sFlow, IPFIX also impacts the traffic forwarding function of the virtual switch, as presented in Fig. 9. For this, we use the same set-up as for the sFlow experiment in Fig. 2: 600 seconds of TCP iPerf3 traffic where sampling rate is changed (or disabled) every 100 seconds. For the same sampling rate and by comparison with Fig. 2, IPFIX has a more pronounced impact on the achieved rate than sFlow in this configuration, in line with its higher CPU consumption.

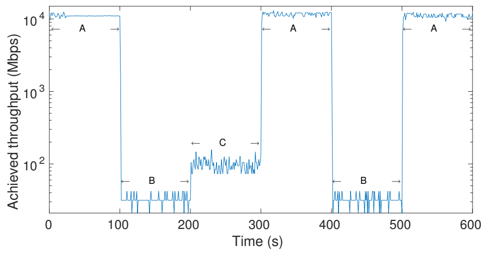


Fig. 9: iPerf3 throughput at different sampling rates of IPFIX: no sampling (A), 100% sampling (B), 50% sampling (C).

B. Introducing cache and flow aggregation

The usual way of configuring IPFIX is with a non-zero value for caching and aggregation. This allows to aggregate several packets from the same flow (TCP or UDP conversation) into a single flow record. When a flow is aggregated on board of the switch, only one report about this flow is sent containing aggregated statistics on it.

Considering the results from the previous section (without aggregation), the next natural question is to evaluate how flow aggregation impacts CPU consumption, as flow aggregation induces more computation on the virtual switch while reducing the network footprint of the measurements.

In a first experiment, we consider the same workload as in the previous section (one long run UDP flow generated by iPerf3) and tune IPFIX caching capacity to cover the entire experiment, hence reducing drastically the number of reports: only one report is sent at the end of the experiment. By doing so, we reduce to almost zero the networking cost of reporting and we only leave the cost of flow aggregation. Results for this experiment are reported in Fig. 8a along with those of sFlow and IPFIX without caching. The cost of on-board processing is, as expected, smaller with caching than without caching for IPFIX. What is striking here, however, is that IPFIX still consumes more CPU than sFlow even when it does not send reports. Note that IPFIX, similarly to sFlow, is implemented in the user space. It is thus normal that IPFIX suffers from the same performance problem at high sampling rate as sFlow (we did not report the impact on traffic of the IPFIX measurement process, but it is similar to the one of sFlow). We can thus conclude on the benefit of aggregation in IPFIX, but also on the importance of the CPU cost of IPFIX, which for its two variants (with and without caching), remains more greedy than sFlow in terms of CPU consumption.

We performed a second set of experiments with a richer workload in terms of number of flows (1000), which is more challenging for IPFIX as it induces more computation on the OvS side. Comparing the results of Fig. 8a and 8b, we can observe that the CPU consumption of IPFIX without caching has slightly increased as compared to the single flow experiment. A further increase in the number of flows could have led to observe a more pronounced difference between IPFIX and sFlow. However, at the scale of physical servers hosting a few tens of VMs in a data center, 1000 flows is already a reasonably large value.

VI. Discussion

We have investigated the influence of virtual network monitoring on the physical server performance and the throughput of monitored applications. Two legacy monitoring tools were considered, sFlow and IPFIX. We performed a sensitivity analysis of CPU consumption and network footprint of the two tools regarding different traffic profiles and monitoring configuration parameters.

Among the set of influencing parameters, sampling rate and traffic throughput in packets per second are the two dominant factors. Indeed, both cause an increase in the number of samples to be generated, thus leading to an increase in the physical resources consumed at the virtual switch. As for sFlow, the polling interval (of counters) induces no CPU overhead for the virtual switch, as counters sent within this interval are small and this interval may be adjusted by the sFlow agent for efficiency reasons. IPFIX appears to be more expensive than sFlow (in terms of CPU consumption) because of its on board flow aggregation feature. Finally, we observed interference between monitoring tasks and monitored traffic, as the cost of monitoring transforms into reduced throughput for the monitored applications.

To reduce the load on the CPU and the impact on regular traffic, the best option is to tune the sampling rate in a way to balance between monitoring accuracy and application performance. Finding this optimal sampling rate is an interesting future direction for our work. We also intend to study an alternative approach whereby the OvS switch would simply mirror traffic to an external measurement process embedded in a dedicated virtual machine. This approach could consume more networking resources but has the potential to alleviate the impact on the regular traffic that we intend to measure.

Acknowledgment

This work was partly funded by the French Government (National Research Agency, ANR) through the Investments for the Future Program reference #ANR-11-LABX-0031-01.

References

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

[2] P. Aitken, B. Claise, and B. Trammell, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information," RFC 7011, 2013.

[3] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic sdn controller assignment in data center networks: Stable matching with transfers," in *IEEE INFOCOM*, 2016.

[4] P. Perešíni, M. Kuźniar, and D. Kostić, "Monocle: Dynamic, fine-grained data plane monitoring," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. ACM, 2015.

[5] P. Zhang, H. Li, C. Hu, L. Hu, L. Xiong, R. Wang, and Y. Zhang, "Mind the gap: Monitoring the control-data plane consistency in software defined networks," in *ACM CoNext*, 2016.

[6] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu, R. Mahajan, D. Maltz, L. Yuan, M. Zhang, B. Y. Zhao *et al.*, "Packet-level telemetry in large datacenter networks," in *ACM CCR*.

[7] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "Dream: dynamic resource allocation for software-defined measurement," *ACM SIGCOMM CCR*, vol. 44, no. 4, pp. 419–430, 2015.

[8] A. Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen, "Umon: Flexible and fine grained traffic monitoring in open vswitch," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. ACM, 2015, p. 15.

[9] D. Sharma, R. Poddar, K. Mahajan, M. Dhawan, and V. Mann, "H ansel: diagnosing faults in openstack," in *ACM CoNext*, 2015.

[10] A. Goel, S. Kalra, and M. Dhawan, "Gretel: Lightweight fault localization for openstack," in *ACM CoNext*, 2016.

[11] W. Wu, G. Wang, A. Akella, and A. Shaikh, "Virtual network diagnosis as a service," in *ACM SoCC*, 2013, p. 9.

[12] A. Zimmermann, A. Hannemann, and T. Kosse, "Flowgrind-a new performance measurement tool," in *IEEE GLOBECOM*, 2010.

[13] iPerf. [Online]. Available: https://iperf.fr/