ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

ARCHIVIO ISTITUZIONALE
DELLA RICERCA

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

Unified and standalone monitoring module for NFV/SDN infrastructures

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

Published Version:
Unified and standalone monitoring module for NFV/SDN infrastructures / Borylo, Piotr; Davoli, Gianluca; Rzepka, Michal; Lason, Artur; Cerroni, Walter. - In: JOURNAL OF NETWORK AND COMPUTER APPLICATIONS. - ISSN 1084-8045. - ELETTRONICO. - 175:(2021), pp. 102934.1-102934.19. [10.1016/j.jnca.2020.102934]

Availability:
This version is available at: https://hdl.handle.net/11585/783754 since: 2021-05-03

Published:

DOI: http://doi.org/10.1016/j.jnca.2020.102934

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/).
When citing, please refer to the published version.

(Article begins on next page)

24 April 2024

Journal of Network and Computer Applications

Unified and Standalone Monitoring Module for NFV/SDN Infrastructures

--Manuscript Draft--

Manuscript Number:	JNCA-D-20-00462R1
Article Type:	Research Paper
Keywords:	monitoring module; unified NFV/SDN; universal tool; ICT softwarization
Corresponding Author:	Piotr Borylo, PhD AGH University of Science and Technology Krakow, POLAND
First Author:	Piotr Borylo, PhD
Order of Authors:	Piotr Borylo, PhD
	Gianluca Davoli
	Michal Rzepka
	Artur Lason
	Walter Cerroni
Abstract:	In the ongoing process of telecommunications systems softwarization , a key role is played by Virtual Network Functions deployed in Cloud environments and interconnected through Software Defined Networks. The separation between data, control and service logic changes the way in which those systems are designed, operated, and capitalized, and also enables elasticity, flexibility and cost-efficiency in the whole ICT ecosystem. However, dedicated and reliable monitoring tools are required to take full advantage of software-based telecommunication systems. Such tools are expected to effectively combine metrics about various assets to provide a global view over the infrastructure, offload control plane from monitoring tasks to improve scalability, and be seamlessly integrated with the existing control plane through the interface. In this paper, we propose a monitoring module that meets the requirements imposed by the softwarization process. We present the reference architecture for the proposed solution, along with the fundamental assumptions we have based our work upon, and a Proof-of-Concept implementation. Our solution has been designed as a standalone Virtual Network Function, fully decoupled from the existing control plane. As a result, new features and degrees of freedom are available, especially in terms of adaptability to the heterogeneous softwarized infrastructure, with no modifications needed in the existing software or hardware components. The proposed modular architecture, supported by proper interfaces, can be integrated with a variety of tools to collects statistics from different assets, relieving the control plane from the burden of computational-intensive monitoring tasks. In order to carefully validate the design of the proposed unified and standalone monitoring module numerous scenarios addressing various aspects and potential impediments were considered, in a series of experiments run on our internal testbeds and on an auxiliary public cloud environment. The obtained results prove that our monitoring tool provides significant advantages with respect to the existing solutions which are integrated in the control plane. Therefore, it is able to cooperate with sophisticated traffic steering and cloud management mechanisms operating on the combined network and computing resources. Furthermore, we demonstrate that our solution is easily portable, for instance to a public cloud environment.

The final published version is available online at: https://doi.org/10.1016/j.jnca.2020.102934

Powered by Editorial Manager® and ProduXion Manager® from Aries Systems Corporation

REVISED Manuscript (text UNmarked)

Unified and Standalone Monitoring Module for NFV/SDN Infrastructures

Piotr Borylo[a], Gianluca Davoli[b], Michal Rzepka[a], Artur Lason[a], Walter Cerroni[b]

[a] *AGH University of Science and Technology, Department of Telecommunications, Poland*
[b] *Department of Electrical, Electronic and Information Engineering (DEI), University of Bologna, Italy*

Abstract

In the ongoing process of telecommunications systems *softwarization*, a key role is played by Virtual Network Functions deployed in Cloud environments and interconnected through Software Defined Networks. The separation between data, control and service logic changes the way in which those systems are designed, operated, and capitalized, and also enables elasticity, flexibility and cost-efficiency in the whole ICT ecosystem. However, dedicated and reliable monitoring tools are required to take full advantage of software-based telecommunication systems. Such tools are expected to effectively combine metrics about various assets to provide a global view over the infrastructure, offload control plane from monitoring tasks to improve scalability, and be seamlessly integrated with the existing control plane through the interface.

In this paper, we propose a monitoring module that meets the requirements imposed by the *softwarization* process. We present the reference architecture for the proposed solution, along with the fundamental assumptions we have based our work upon, and a Proof-of-Concept implementation. Our solution has been designed as a standalone Virtual Network Function, fully decoupled from the existing control plane. As a result, new features and degrees of freedom are available, especially in terms of adaptability to the heterogeneous softwarized infrastructure, with no modifications needed in the existing software or hardware components. The proposed modular architecture, supported by proper interfaces, can be integrated with a variety of tools to collects statistics from different assets, relieving the control plane

Email address: borylo@agh.edu.pl (Piotr Borylo)

from the burden of computational-intensive monitoring tasks.

In order to carefully validate the design of the proposed unified and standalone monitoring module numerous scenarios addressing various aspects and potential impediments were considered, in a series of experiments run on our internal testbeds and on an auxiliary public cloud environment. The obtained results prove that our monitoring tool provides significant advantages with respect to the existing solutions which are integrated in the control plane. Therefore, it is able to cooperate with sophisticated traffic steering and cloud management mechanisms operating on the combined network and computing resources. Furthermore, we demonstrate that our solution is easily portable, for instance to a public cloud environment.

Keywords:
monitoring module, unified NFV/SDN, universal tool, ICT softwarization.

---

1. Introduction

The ongoing *softwarization* process in telecommunications is changing the way networks are designed, operated, and capitalized [1]. Software-Defined Networking (SDN) [2], Network Function Virtualization (NFV) [3] and cloud computing [4] are outstanding examples of a trend toward the full separation of all network services and functions from the underlying physical infrastructures. The features of each of those concepts represent a catalyst for the others, and foster the transition of ICT ecosystems into software-based solutions. Network Functions (NFs) are intended to be dynamically deployed as Virtual Machines (VMs) or containers, and interconnected via SDN within a cloud environment. The resulting architecture is expected to be flexible, easily scalable, reliable, and thus, cost-efficient [5].

However, reliable resource monitoring tools are required to take full advantage of the *softwarization* process by adjusting the infrastructure to the changing conditions. In particular, considering the convergence of SDN, NFV and cloud computing technologies, such a monitoring system should be able to collect metrics that are typical of network domains and of a variety of computing assets [6]. Moreover, the possible impact on the performance of existing control plane solutions, and the integration with the related, often heterogeneous components, must be thoroughly considered. Fortunately, the aforementioned challenges, further confirmed in a recent survey [7], can be faced in a way that takes advantage of the features offered by the softwarized

environment itself: global view over combined resources, on-demand resource provisioning, and integration between control planes of different domains are not only enabling, but also demanding the design of a unified and standalone monitoring tool.

In this work, we propose a monitoring module designed as a standalone Virtual Network Function (VNF) and compatible with a generic SDN/NFV infrastructure deployed in a cloud environment. The solution is separated from the existing control plane components and collects measurements for different assets. Decoupling the monitoring module from the control plane and considering it as a standalone VNF not only spares critical elements such as the controllers from additional workload, but also brings further advantages in terms of new features and degrees of freedom. For example, such a modular architecture enables flexible integration with a variety of control plane components or tools gathering particular metrics on behalf of the proposed platform. No modifications are needed in the existing software or hardware solutions for the purpose of such integration, as only well-known and widely implemented protocols and interfaces are used. Finally, updates in the monitoring module do not trigger any changes in other components. The proposed standalone monitoring module could be used to feed monitoring information to a service orchestrator, or to a traffic steering mechanism operating over complex infrastructures comprising both network and computing resources (as, for example, presented in [8]). Additionally, our solution may be useful to discover the limitations and real performance of a virtualized infrastructure managed and controlled by a third-party provider. The proposed system has been implemented, deployed, and thoroughly validated in three different experimental environments. Two of them are real testbeds, the first one based on the OpenStack (OS) cloud platform and the second one built on container technology. Furthermore, we have also run our solution in a public cloud environment using an automation tool to prove portability.

The rest of the paper is organized as follows. Section 2 presents the current state of the art on network/compute resource monitoring as well as on integrated monitoring. Section 3 provides the reference scenario of combined SDN/NFV and discusses the background for the design of a unified and standalone resource monitoring module. Section 4 presents the architecture of the unified monitoring module proposed in this paper, whereas the implementation of the proposed solution is described in detail in Section 5. Section 6 reports results and analysis of the experimental validation performed under numerous scenarios. Section 7 concludes the paper.

2. Related Work

A very broad and comprehensive survey on network data collection has been proposed in [9] and also numerous solutions have been proposed for the purpose of monitoring SDN networks and most of them are surveyed in [10] and [11]. On the other hand, isolated NFV monitoring frameworks have been widely studied and analyzed in the literature, e.g. both the most recent survey [12] and also the older one [3] address this issue. Additionally, the most popular Virtualized Infrastructure Managers (VIMs) and hypervisors comprise metering modules able collect information about utilization of host machines and VMs. Furthermore, in general purpose public cloud computing infrastructures detailed monitoring of resource utilization is a mandatory feature for cloud providers to ensure precise billing with high granularity [13].

However, as the main scope of this paper is a unified approach to monitor NFV/SDN infrastructures, we will not concentrate on the solutions designed for general purpose and separated SDN and NFV entities. Instead, we refer to the approaches suitable for NFV/SDN infrastructures, which are pointed out to be one of the most important challenges in a very recent survey [7]. Solutions that are based on both network- and computing-originated metrics are presented in Section 2.1, as they are the most similar to our approach. However, also works that, in contrast to our solution, assume collecting solely network or computing metrics are presented in Sections 2.2 and 2.3, respectively.

2.1. Integrated monitoring

A generic framework aimed at constructing service chains in NFV/SDN infrastructure is presented in [14]. Unified monitoring for both networking and computing resources is designed. However, due to the tutorial nature of this work, implementation details are not provided to make the concept useful from a practical point of view. Furthermore, authors in [14] assumed that monitoring modules are integrated with NFV manager and SDN controller, which may deteriorate the performance of the control plane.

The monitoring framework that represents the closest solution to the one proposed in this paper was presented in [15] and released as an open-source project[1]. In terms of architecture design, the authors assumed that

[1] `https://github.com/T-NOVA/vim-monitoring`

SDN is one of the most important enablers for the Virtual Network Function Chain (VNFC). Regarding the monitoring module, it was assumed that any candidate monitoring framework should limit the amount of information being exchanged with the control plane in order to offload it, as scalability is a crucial issue in NFV and Data Centre (DC) environments. Thus, a monitoring solution should be effective, accurate, and is expected to utilize basic mechanisms in order to be as lightweight as possible. However, despite the mentioned similarities, two significant architectural differences exist between the framework presented in [15] and our work. First of all, we use the dedicated but well known and widely implemented sFlow protocol to collect network metrics instead of gathering OpenFlow statistics, which may lead to network control plane overloading. OpenFlow statistic accuracy and efficiency strongly depend on the network load, imposing higher utilization of the SDN controller and switches. The second difference regards the fact that in [15] the authors considered the deployment of a monitoring agent module within the VNF instance to collect VNF-specific metrics as an addition to existing cloud platform monitoring facilities (e.g., the OpenStack Ceilometer module). This makes that solution more challenging in terms of deployment and integration with existing infrastructures. In [15] some limited experimental results are presented and are slightly extended in the project deliverable [16]. Basic evaluation was performed and was focused on two aspects: load reduction in the control channel, and accuracy of an anomaly detection system implemented on top of the monitoring module. Differently, in this work we present a detailed and comprehensive experimental study of the effectiveness of our framework.

Edge data centers interconnected through SDN networks, controlled by OpenStack and hosting VNFs are considered in [17]. Dedicated probes are assumed to be installed on servers to report VNF-specific parameters to the VIM. At the same time, the SDN controller collects information about network utilization from software switches and also provides it to the VIM. The strongest aspect of the paper is a Proof of Concept (PoC) being similar to the prototype proposed in this work. However, the approach engages both SDN and VNF control planes in the monitoring activities imposing additional resource utilization, which is the main drawback compared to our solution.

2.2. Network-based monitoring

Solutions such as [18], [19], and [20] that are based solely on network metrics suffer from the fact that NFV-specific metrics are neglected. In [18]

a monitoring module was proposed as an integrated component of the Floodlight[2] SDN controller. It performs periodical network measurements using OpenFlow statistics with the aim to monitor bandwidth usage and identify congested links. Then, based on those network metrics VM migration is performed by the VIM according to the proposed algorithm. Neglecting information about, for example, CPU or RAM utilization during those actions seems to be a significant limitation in terms of VM deployment. Additionally, such an architecture adds further overhead to the network control layer, differently from our proposal which is to separate monitoring and control functions.

In [19] the integrated orchestration of cloud and SDN network is being considered. Monitoring is mentioned as one of the three core functions of the orchestrator, which is especially important in DC environments where complex traffic patterns occur. However, two important limitations must be mentioned. The first one regards the fact that solely network measurements are performed while cloud-specific metrics are neglected. It may have negative impact on optimization performed in further steps. The second disadvantage is related to the validation of the solution. It is performed using simulation techniques and, differently from our work, no PoC or prototype on a realistic experimental environment is proposed.

Finally, the authors in [20] developed a distributed network monitoring framework leveraging information collected from software switches to identify bottlenecks between tiers in cloud-scale applications. Again, no measurement is performed in the computing domain, and the collector is integrated with the control plane component, which imposes additional resource utilization. Both aspects make the proposed solution different from our monitoring framework.

The computing infrastructure is not considered at all in [21], where authors described a solution dedicated to data center networks and focused on efficient flow-oriented counters. Despite this significant difference with respect to our work, the approach is worth to be mentioned due to some similarities, including the design of the monitoring module as a separate entity for scalability and independence of the SDN architecture.

[2]http://www.projectfloodlight.org/

2.3. Computing-based monitoring

Similarly to [15], the work reported in [22] is the result of the T-Nova project[3] and is partially based on the framework designed in [15]. Thus architectural similarities to our work still exist. The aim of the SDN-based monitoring system proposed in [22] is to detect and report abnormal conditions in the NFV infrastructure. For that purpose CPU and memory utilization as a function of traffic handled by VNF is analyzed. However, network metrics are not being considered, which is a significant limitation when compared to our work. A second issue regards the fact that technical details provided in [22] are limited. Conversely, we thoroughly report the experimental assessment of our framework in a realistic testbed. The most recent publication from the T-Nova project [23] summarizes all of the project achievements, but does not include any novel results or concepts.

Cloud-related metrics are also considered in [24], where a monitoring framework is proposed to control probes in both network and cloud infrastructures. The aim is to use information originating from layers 4-7 assuming it is valuable from the NFV point of view. However, fundamental network-based metrics are neglected. Furthermore, the solution is based on the emerging Network Service Header (NSH) protocol, recently defined by the IETF [25]. Unfortunately, production-level implementations of NSH nodes are still limited. On the contrary, in our approach we use well-known and widely deployed protocols (such as sFlow), and universal communication interfaces.

3. Unified Monitoring in SDN/NFV Infrastructure

The very basic principles of SDN [2] can be summarized as follows: (1) the data plane (carrying user data traffic) is decoupled from both control and management planes, (2) control plane functionalities are logically centralized and performed by the entity known as the SDN controller, and (3) the SDN infrastructure can be effectively programmed through the controller itself by means of a northbound interface offered to the application layer. On the other hand, NFV [3] adopts virtualization techniques in the process of Network Function (NF) deployment. Numerous types of NFs can be virtualized, depending on the specific service being deployed and infrastructure being

[3] `http://www.t-nova.eu/`

used. A set of VNFs connected in the desired order, according to the service to be provisioned, defines a VNFC, which falls under the more general class of Service Function Chain (SFC). Those VNFs are deployed in a cloud environment [4] where computing, storage and networking resources are provisioned on-demand and are managed by a cloud orchestration/controller entity.

The features of each of the SDN, NFV and cloud computing paradigms enhance the potentialities of the others. The mutual advantages between these three building blocks of network softwarization are summarized in Fig. 1. More specifically, on-demand cloud resource provisioning makes it possible to elastically deploy both VNFs and SDN control plane components, preserving multi-tenancy capabilities [26]. The cloud-based approach allows scaling the allocated resources according to current requirements, ensuring performance and reliability, and eliminating the need for over-provisioned expensive dedicated hardware (costs reduction). Additionally, SDN takes advantage of a global view over the network infrastructure to easily implement sophisticated and programmable traffic steering mechanisms that can reduce the overall load in a cloud environment and ensure efficient connectivity between VNFs. Finally, NFV enables the development of application-aware architectures combining computing and networking functions in a cloud-like environment, with great elasticity and cost-efficiency. The softwarized nature of these three building blocks allows for close integration and cooperation among them.

Each of the three building blocks (i.e., SDN, NFV, and cloud computing) offers some kind of monitoring capabilities. Regarding the cloud infrastructure, it is possible to monitor any specific physical or virtual machine deployed in the cloud and hosting VNFs. Additional monitoring agents may report more detailed metrics specific for each VNF (e.g. rate of packets being inspected and their average size). Finally, the SDN infrastructure should collect data about network load in a flow-aware manner, mapping flows into the relevant VNFC.

However, to fully benefit from the described ecosystem, the monitoring module has to be able to conduct unified measurements over distributed heterogeneous resources. Such a monitoring tool should be able to effectively combine all the metrics and formulate comprehensive conclusions on resource usage. Furthermore, to fully decouple control plane from monitoring tasks, the monitoring tool is expected to be separated from SDN, VNF and cloud controllers/orchestrators. Thus, it should be considered as a special case of VNF, taking advantage of virtualization capabilities enabled by the cloud

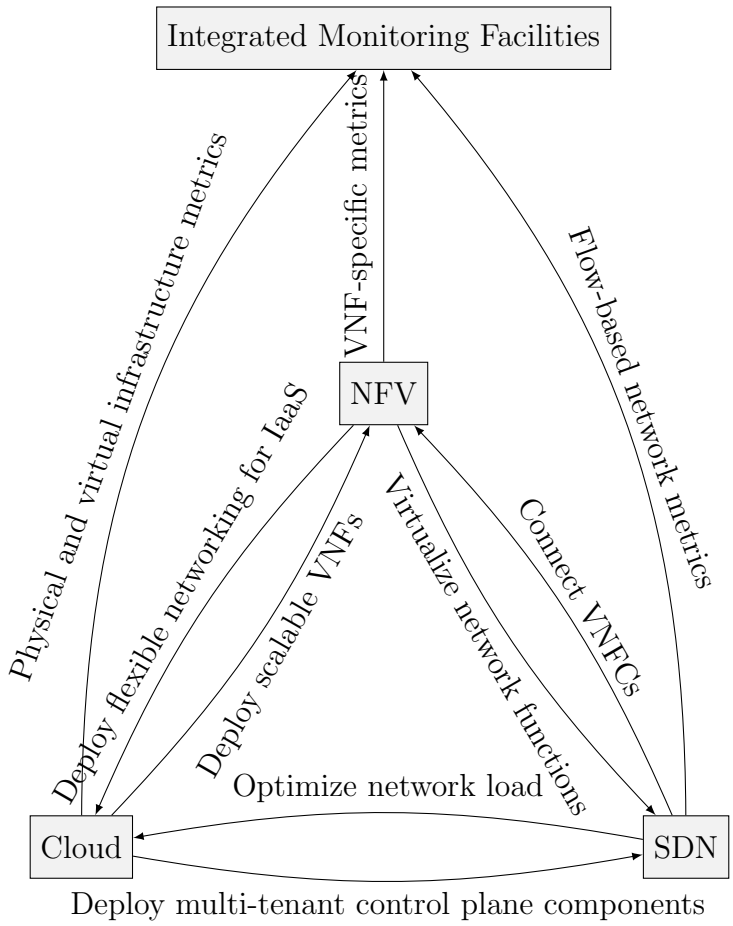


Figure 1: Mutual advantages of SDN, NFV and cloud computing paradigms and respective monitoring capabilities.

infrastructure. Finally, the monitoring module should utilize well-known protocols to allow seamless integration with existing infrastructures and to introduce some redundancy. The unified monitoring system proposed in this paper is designed to meet the aforementioned requirements and, as a result, be advantageous in the perspective of new features and degrees of freedom compared to existing solutions.

As a result, the monitoring module can enable several features, including optimization of complex infrastructures from the global point of view, full automation of service deployment combined with the capability to react to the changes in the data plane, on-demand placement of VNFs [27] or dy-

namic modification of VNFCs combined with SDN-enabled traffic steering mechanisms to ensure the desired Quality of Service (QoS) level [6]. Furthermore, energy-aware issues in terms of joint online resource provisioning can be effectively addressed, as proved in [28] and [29]. The same approach can also be extended to the fog computing paradigm, which offers latency-aware traffic management based on the global knowledge of networking and computing resource conditions [30].

4. Unified Monitoring System Architecture

The description of the proposed unified monitoring system is divided into two parts. The first part, reported in this Section, describes the architecture of the solution and the fundamental assumptions, highlighting advantages with respect to previously existing solutions. The second part, reported in Section 5, describes the PoC implementation. The description shows the compatibility of our solution with generic SDN/NFV infrastructures, and emphasizes the usage of existing protocols for monitoring, which do not operate in the control plane. Additionally, versatility and portability were proven by deployment in three different environments: two real-life testbeds and a public cloud. These features are key to achieving the improvements demonstrated by experimental validation and described in Section 6.

One of the most significant properties of the proposed unified and standalone monitoring module is the separation of the monitoring functions from other control and management plane components. As such, the monitoring module is a software element that can be deployed as one of the VNFs, with all the advantages that this choice implies, as discussed in section 3. First of all, decoupling it from the control and management plane components makes our solution independent of the specific cloud and network controller technology. Any interaction needed with the control/management plane can take place through the northbound interfaces offered by the majority of SDN controllers, e.g., Ryu[4], ONOS[5] or OpenDayLight[6], as well as by different cloud orchestrators/Virtual Network Function Managers (VNFMs)/VIMs,

[4]https://osrg.github.io/ryu/
[5]http://onosproject.org/
[6]https://www.opendaylight.org/

such as OpenStack[7] or Kubernetes[8]. Thus, there is no need for complex, time consuming and technology-dependent integration with specific control planes. Moreover, such an approach offloads the existing control plane from monitoring tasks and, consequently, from the decisional burden over possible actions to take. Additionally, from the perspective of possible commercial implementations, any update to the monitoring module will not require integration efforts in the customer's SDN controller or VNF orchestrator. Thanks to the unified design approach, such a standalone monitoring module can provide consistent information to automatically trigger traffic steering mechanisms that jointly take into account network and computing resources, interacting with both the network controller and cloud orchestration software in a coordinated manner.

Finally, modularity facilitates potential extensions of the monitoring module. For example, in the current version, the proposed solution is focused on passive monitoring. However, active monitoring features may be provided by adding separate components responsible for the active monitoring orchestration, i.e. deployment of traffic generators, configuration of the infrastructure to handle monitoring traffic (e.g. setup network paths or deploy VNFs), and measurements scheduling. Thus, the existing component of the monitoring tool will be only extended with an interface to communicate with these new components.

Figure 2 shows the architecture of the proposed unified monitoring system for SDN/NFV infrastructures, as well as the possible interactions with the SDN controller and the VIM. The dark gray components belong to the control/management plane, whereas the white ones refer to data plane modules. The latter include VMs implementing the VNFs, virtual switches (e.g., Open vSwitch (OvS) instances[9]) used to interconnect them within the same physical Host Machine (HM), and hardware switches interconnecting the physical hosts. The unified and standalone monitoring module is represented by the light gray box. It is able to collect monitoring data from different sources, including the VNF orchestrator and the SDN controller through their respective northbound Application Programming Interfaces (APIs), as well as directly from the network infrastructure elements using existing and widely

[7]https://www.openstack.org/
[8]https://kubernetes.io/
[9]http://openvswitch.org/

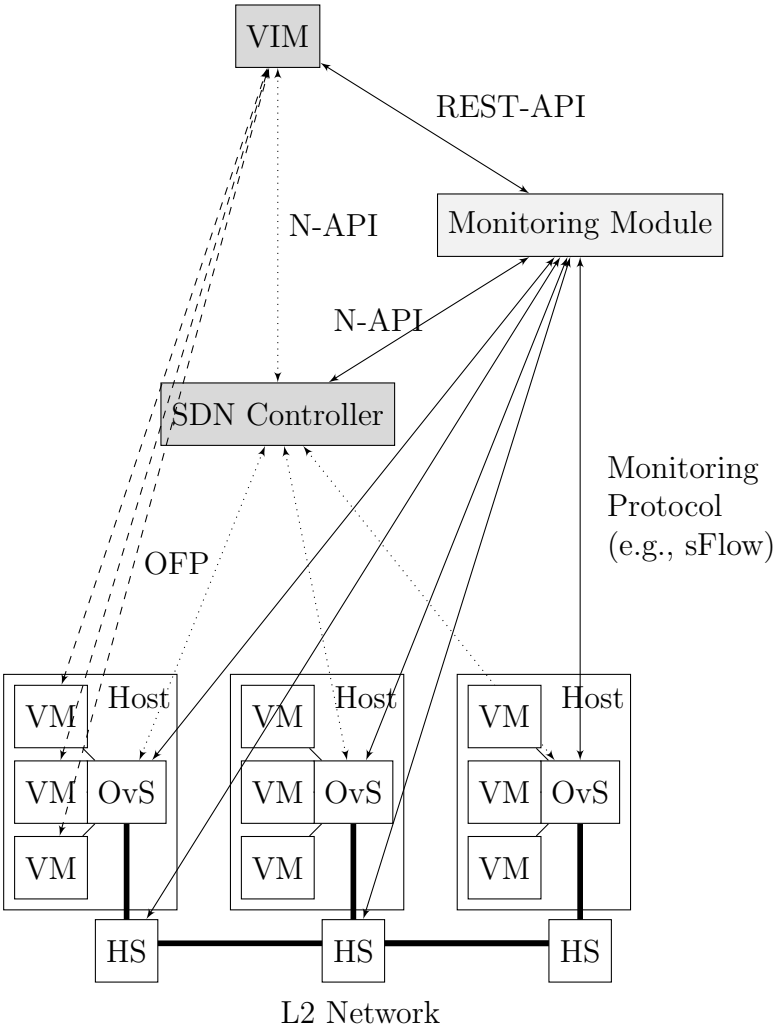


Figure 2: Architecture with dark gray control plane elements, light gray monitoring module and white data plane elements. VIM - Virtualized Infrastructure Manager, N-API - Northbound-API, OFP - Open Flow Protocol, VM - Virtual Machine, OvS - Open vSwitch, HS - hardware switch. Solid, dashed and dotted lines are used to denote communication-related to the monitoring module, SDN controller (northbound and OFP), and VIM, respectively.

supported standard protocols, such as sFlow.

The modular nature of the proposed monitoring system offers an additional degree of freedom over which data sources should be used and how often they should be queried. For instance, based on the present conditions

of a given infrastructure, it is possible to decide whether to rely on detailed flow statistics collected by the SDN controller through OpenFlow, or to get sampled flow data from the physical and/or virtual switches using sFlow, resulting in different granularity levels of the monitoring information but also different loads imposed to the control plane. Therefore, whenever the data collected directly from the SDN/NFV infrastructure is sufficient for the monitoring module to understand how the resources are being used and whether any corrective action should be put in place, the proposed approach allows to offload the control/management planes from the burden to deal with such operations. Such a redundancy can be also advantageous for uninterrupted statistics gathering in case of malicious behavior aimed at disrupting monitoring infrastructure. The separate interesting byproduct is that the monitoring module could also be programmed to react to certain conditions and trigger either the controller or orchestrator to perform corrective actions aimed at avoiding performance deterioration.

5. Prototype Implementation

We developed a prototype of the unified and standalone monitoring module and deployed it in three experimental setups that implement different parts of the system architecture displayed in Fig. 2. We first discuss the technical details of the prototype that are common to all experimental environments. Then, in the following subsections, we focus on environment-specific matters.

The monitoring module itself has been developed as a software application written in Python, due to the suitability of that language for prototyping purposes and ease of implementation of a REST API. The monitoring module was designed to collect and aggregate information gathered from multiple sources. In the prototype, the module implements data collection from Openstack Ceilometer/Gnocchi as well as flow sampling directly from the network devices by means of an sFlow collector[10]. The sFlow protocol was selected among a number of flow statistics gathering tools, and custom-implemented in the form of a simple Python application, running as a plugin of the monitoring module. As an alternative, the NetFlow protocol was deemed not suitable as it is designed as a L3 solution (Layer-3 in the OSI model).

[10] http://www.sflow.org/

Another point in favor of the choice of sFlow has been its widespread support by network devices, including OvS, thus making it readily available, for example in OpenStack's virtual switches, eliminating the need to install sFlow agents on purpose. The main protocol parameters that can be used to tune the behavior of sFlow are the *sampling ratio* and the *sample aggregation interval*. The sampling ratio N denotes that, on average, one out of N packets handled by the node will be sent by the sFlow agent to the sFlow collector for the purposes of statistics gathering. The higher the value of N, the lower the communication and computational overhead. However, by increasing N the measurements are available with a higher delay and worse sensitivity. Actually, by default the sFlow agent only sends the first B bytes of the sampled packet to the sFlow collector, together with an indication of the total packet size. In fact, the first bytes include the packet headers, which are then processed for statistical purposes by the collector, whereas the rest of the packet would contain only payload content without meaningful information for flow monitoring purposes. This way the sFlow protocol overhead is reduced, without impacting the statistical accuracy. The sample aggregation interval C denotes the interval in which individual sFlow samples are aggregated, meaning that all samples received within a time window of C seconds are combined and treated as single measurement entry. This aggregation aims to attribute a group of asynchronous samples to a given moment in time, thus achieving a tunable "quantization" of the time axis. Similarly to the sampling ratio, increasing the value of the sample aggregation interval C reduces the monitoring overhead and deteriorates the monitoring capabilities in terms of measurements sensitivity and time needed to obtain the collected monitoring data.

The monitoring module was also equipped with a REST-API able to provide information, gathered by means of sFlow and Ceilometer, regarding resource utilization in the VMs, as well as data flows observed among them. In our implementation, a flow was defined by the pair of its source and destination IP addresses. For each flow, we collect both an instantaneous data rate estimate value, and an Exponential Weighted Moving Average (EWMA) value according to the recursive formula

$$e_n = \alpha s_n + (1 - \alpha) e_{n-1} \tag{1}$$

where e_n is the EWMA value computed when the n-th instantaneous sample s_n is received and $0 < \alpha < 1$ is a weighting coefficient. The EWMA was introduced to smooth out measurements and make them robust against load

fluctuations. The α coefficient expresses how fast historical measurements lose importance: the higher its value, the heavier the weight of instantaneous samples.

It is important to quantify the overhead caused by a monitoring protocol such as sFlow, i.e. the amount of additional signaling traffic exchanged between network nodes and the sFlow collector to perform the monitoring operations. For each sFlow packet p received by the collector, we can define the relative overhead as the ratio of the size of the sFlow packet $L_{\text{sflow},p}$ over the total amount of data to which that packet refers. The latter quantity includes the size of sFlow packet p, plus the full size of each sampled data packet carried by sFlow packet p, plus the size of $N-1$ data packets not being sampled for each sampled packet in p. Assuming that, on average, the total size of the $N-1$ non-sampled packets is $N-1$ times the size of the sampled packet,[11] we can approximate the relative overhead of packet p as:

$$O_{\text{sflow},p} = \frac{L_{\text{sflow},p}}{L_{\text{sflow},p} + N \sum_{i=1}^{n_p} L_i} \tag{2}$$

where n_p is the number of samples carried by sFlow packet p and L_i is the full size of the i-th sampled packet.

The size of a generic sFlow packet is variable and depends on the number and size of the carried samples, including meta-data associated to each sample and carried in a sample header. Recalling that sFlow packets are transported by UDP datagrams, in an Ethernet network we have:

$$L_{\text{sflow},p} = h_{\text{eth}} + h_{\text{ip}} + h_{\text{udp}} + h_{\text{sflow}} + \sum_{i=1}^{n_p} (h_{\text{sample},i} + B) \tag{3}$$

where h_{eth} is the Ethernet header size, h_{ip} is the IP header size, h_{udp} is the UDP header size, h_{sflow} is the sFlow header size, $h_{\text{sample},i}$ is the i-th sample header size, and B is the sample data size.

While we considered sFlow suitable for the purpose of our prototype, it may not be sufficient to perform detailed real-time monitoring tasks. Thanks to the modular architecture of the monitoring system, the sFlow protocol can always be replaced with a different data plane monitoring solution [10].

[11] This assumption is equivalent to the assumption made by the sFlow protocol, which considers one sample out of N as an estimation of the monitored bit rate.

5.1. Testbed based on private cloud platform

A first experimental evaluation of our prototype was performed in a testbed running in a private cloud environment. The software platform chosen to implement the SDN/NFV infrastructure is OpenStack (Newton version), one of the most popular cloud computing platforms used to deploy NFV. Recent versions of Neutron (the OpenStack networking module) natively integrate SDN solutions in order to increase flexibility and improve performance of both the control and data planes [31]. Therefore, such a platform represents the ideal playground to experiment with the proposed unified monitoring module. The OpenStack cluster used in our implementation consisted of:

1. a controller node, where all the required cloud services are running, including the endpoints of the relevant APIs; as such, the OpenStack controller acts as the VIM in our architecture;
2. two compute nodes, where the VNFs have been implemented as dedicated VMs, managed by the KVM hypervisor[12] and supported by the *libvirt* library[13];
3. a network node, where all networking services to provide external connectivity and traffic filtering features are located.

The telemetry facilities natively provided by OpenStack include metering, monitoring, and alarming functionality. In particular, the Ceilometer component is responsible for polling events and monitoring data notified by different OpenStack services, and for publishing collected data to suitable data stores and message queues. In our experimental setup we used Ceilometer to collect selected metrics related to HMs and VMs resources directly from the compute service. Gnocchi was chosen as metric storage service and time-series indexing and aggregation tool for data collected by Ceilometer. Its REST API was used by the monitoring module to retrieve monitoring data from the VNF orchestrator. With Ceilometer it is possible to define specific metrics so as to target the specific parameters the monitoring application is interested in. By default, Ceilometer provides a measurement for each defined metrics every 10 minutes. However, for real-time monitoring purposes we reduced this period to 10 seconds and also analyze other values in this work.

[12]https://www.linux-kvm.org/
[13]https://libvirt.org/

The SDN controller adopted in our experimental setup is Ryu. The reason behind this choice is that Ryu is the OpenFlow controller natively integrated in each OpenStack compute node for managing the virtual network infrastructure [31]. With a simple customization of the Ryu instance provided by OpenStack Neutron we were able to make it expose its Northbound API, thus enabling the interaction with our monitoring module. The Ryu Northbound API allows retrieving data about network topology and its current state, including flow and meter statistics. Since in our experiments we used only the virtual switches deployed by OpenStack, we did not have to deploy an external SDN controller, which would be necessary in case OpenFlow monitoring data were to be collected also from physical switches.

A more detailed representation of the prototype monitoring module and of the interactions between monitoring components is depicted in Fig. 3. The sFlow agent in each OvS samples one packet out of N for a particular flow that the OvS forwards in the data plane network – a flow being defined by the source and destination of the traffic. Sampled packets are assembled in sets comprising two to six units, and sent to the sFlow collector in a sFlow datagram packet. The rate of arrival of sFlow datagram packets to the sFlow collector is not steady nor predictable, as it depends on the intensity of the data plane traffic through the sampling ratio N. Therefore, there is no synchronization between sFlow agent and sFlow collector. The collector aggregates sFlow datagrams over temporal windows spanning C seconds, where C is the aforementioned sample aggregation interval, generating one measurement entry every C seconds, on the closing instant t_n of the current temporal window. Fig. 3 also visualizes how N and C parameters affect the delay after which measurements are available and sensitivity. Each measurement entry coming from the sFlow collector is the result of the combination of the most recent sFlow sample with the recent previous ones through an EWMA, as detailed in the following paragraph. Parallel to the sFlow data sampling and collection, the other main component of the monitoring module, the Ceilometer poller, periodically polls the time series database (Gnocchi) of the Ceilometer module in the OpenStack controller to retrieve data measured from there, and generates measurement entries of its own, to be stored in the measurement database along with the entries generated through sFlow.

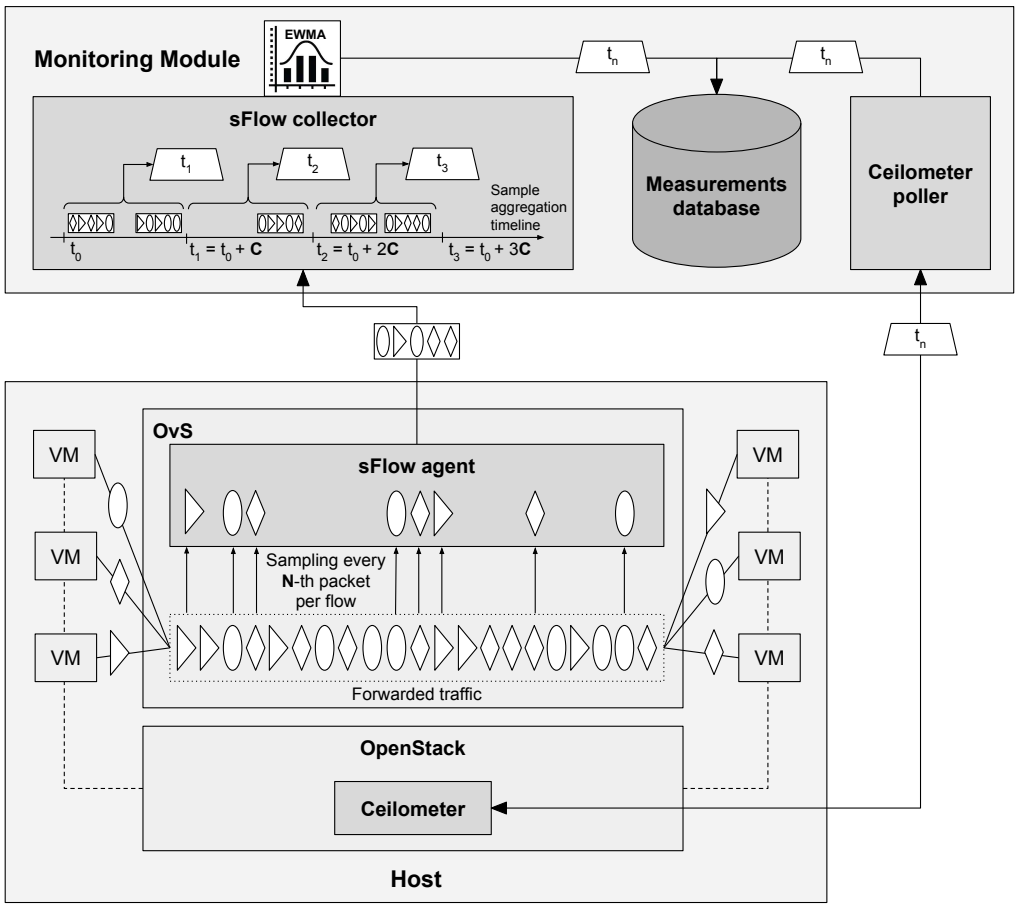


Figure 3: Inside-view of the prototype monitoring module with a representation of the main mechanisms. Data packets are represented with ovals, triangles and diamond shapes, each representing a different traffic flow; sFlow datagrams are represented by rectangles; measurement entries referred to instant t_n are represented with trapezoids.

5.2. Testbed based on container technology

Container technology has gained immense popularity since its introduction, owing to the diverse benefits it can offer to a vast range of applications and services. Containerization enables lightweight and scalable deployments of service functions in the network, facilitating dynamic service provisioning and management. This makes the inclusion of container technology meaningful for the thorough evaluation of the capabilities of a monitoring module such as the one proposed in this paper. Therefore, we evaluated our prototype in a testbed based on container technology. The container management software chosen for this work was Docker, one of the most popular tools for container management, for its high level of automation in the control and management processes of containerized resources, and availability of APIs, making it a valuable instrument for the experimental validation of the monitoring approach. In this scenario, Docker acted as VIM, handling requests for deployment of new VNFs, managing their networking and overseeing their lifecycle.

Our container-based testbed consisted of four physical machines, all equipped with 4 CPUs (Intel i5-4460 up to 3.2 GHz) and 8 GB of RAM. All of them were connected to the same local private network, which served as control and management network only. Additionally, the machines were arranged in two pairs, by connecting the first two and the last two of them directly, and connecting the second and third machines via a physical SDN switch (the OpenFlow-enabled HP Aruba 2920-48G switch). The first machine hosted the running instances of the monitoring module and of the SDN controller (Ryu), as well as the source endpoint of the generated traffic flows. The second and third machines hosted the container management software (Docker), as well as the virtual switches (OvS) required to complete the desired overall logical topology. The fourth and last machine hosted the destination endpoints of the generated traffic flows. The links between each pair of machines had a physical capacity of 1 Gbit/s. However, some links in the final logical topology were limited to different values for evaluation purposes that will be addressed in Section 6.4. A depiction of the physical testbed, along with a representation of the intended logical topology, is given in Fig. 4.

5.3. Public cloud environment

As a third evaluation environment for our prototype, three VMs were deployed in the public cloud infrastructure provided by Amazon Web Services (AWS). Each VM was a *t3.medium* instance with 2 vCPUs (2.5 GHz Intel

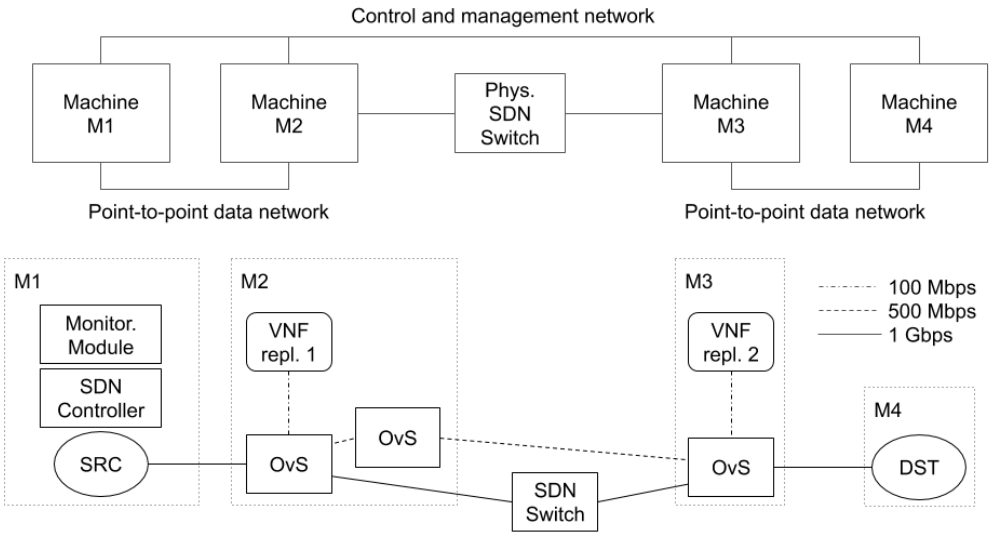


Figure 4: Containerized testbed topology: physical (upper) and logical (lower) setups.

Scalable Processor) and 4 GB of RAM allocated. The monitoring module was housed by one of the VMs, while the two others served as the endpoints of the generated traffic. As stated in the provider's documentation, the instances were eligible to generate traffic bursts reaching up to 5 Gbit/s. However, the baseline data rate observed during the experiment was well below 1 Gbit/s. In contrast to the other testbed environments deployed as a part of the research presented in this paper, the public cloud exhibited a number of restrictions typical of such kind of services. Both networking, virtualization and orchestration layers exposed only a subset of features for the end-user to manage, while some of the core components remained fully transparent and solely under the provider's control. Under these circumstances, our monitoring module can be used to discover the real performance parameters of the cloud network, as demonstrated in 6.5.

As users of public cloud are usually provided with a set of defined interfaces, i.e., web console, command line, and programming interface, additional tools can be used to simplify the process of setting up the infrastructure. In the experiment, the *Terraform* tool was chosen to define and deploy the testbed components in accordance with the *Infrastructure as Code* concept. A representation of the described scenario is given in Fig. 5.

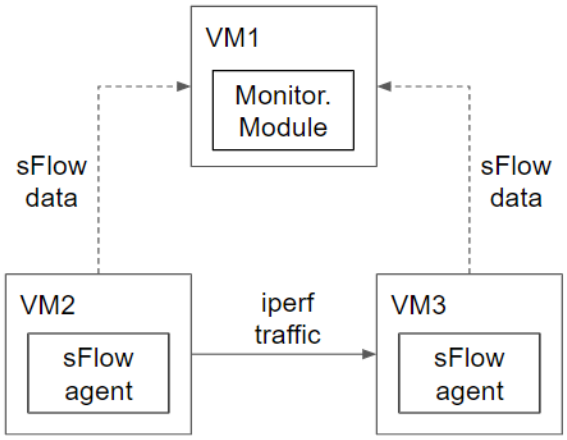


Figure 5: Public cloud testbed topology

6. Experimental Validation

In order to carefully validate the design of the proposed unified and standalone monitoring module for SDN/NFV infrastructures, numerous scenarios addressing various aspects and potential impediments were considered in a series of experiments run with our prototype implementation in the three different environments described in Sec. 5. The following subsections refer to those scenarios, describing the experiments and presenting the results of quantitative analysis, proving the advantages of using existing protocols outside the SDN control plane.

In our experiments we used the iperf[14] software tool to generate traffic in the data plane and measure the throughput between VNF instances. Each quantitative experiment was repeated 10 times for statistical accuracy and each iperf session lasted at least 30 seconds in order to be able to gather a "stable" bit rate value from the monitoring module. For each measurement we calculated 95% confidence intervals to ensure statistical correctness of the performance evaluation, but we decided not to show them in every figure for readability reasons. Instead, we discuss the accuracy of our validation campaign in subsection 6.7. Since the throughput measured by iperf refers to transport layer payload, we reported the link-level bit rate after multiplying

[14]https://software.es.net/iperf/

the values measured by iperf by a suitable corrective constant [32]. The iperf tool was chosen because of its stability and low computation resources utilization. In the private cloud testbed the endpoints of the generated traffic flows were placed on virtual machines within the same network segment, and the traffic never had to cross an external network. In the container-based testbed, physical machines connected to the same network segment as the machines hosting the VNFs served as endpoints of generated traffic. This proves the generality of the approach, as the employed monitoring technology does not require specific hardware or network configuration choices for the monitored devices. In fact, the communication between agent and collector monitoring modules is independent of the forwarding technology of the network.

In the following presentation of the results we first focus on the crucial feature of our monitoring module. Thus, subsection 6.1 shows that traffic monitoring performed at the data-plane with sFlow is strictly correlated to compute resource consumption measured by the OpenStack telemetry facilities, i.e. Ceilometer + Gnocchi. This proves that our module can actually be considered "unified" and capable of monitoring both SDN and NFV metrics of the infrastructure. To further prove the feasibility of our solution on the SDN/NFV infrastructure, some constraints related to the measurement latency of the Ceilometer module are considered in subsection 6.2, whereas in subsection 6.3 we compare sFlow and Ceilometer data collection capabilities in terms of traffic flow granularity. The general message of those experiments is that use of existing protocols outside the SDN control plane jointly with a modular architecture provides significant advantages in terms of new features and degrees of freedom with respect to existing solutions that are integrated in the control plane. One must also note that both sFlow and OpenStack telemetry facilities are solely examples of tools that can be easily replaced thanks to the loose coupling assumed in our solution.

Then in subsection 6.4 we present the potential advantages of the proposed solution by applying suitable traffic steering based on the real-time measurements collected by the monitoring module. It is therefore shown that the monitoring module is able to cooperate with the employed SDN controller and VIM to react to changes in the utilization of monitored resources. The fundamental features of the proposed monitoring solution deployed in a public cloud were also tested, as described in subsection 6.5. Results in both subsections 6.4 and 6.5 justify the portability of the module by effortless integration with a variety of networking and virtualization solutions. Due to

Table 1: Summary of experiments, evaluation environments and corresponding result subsections.

Experiment	Environment	Subsection
Unified monitoring for NFV/SDN	Private cloud	6.1
Ceilometer response time	Private cloud	6.2
Traffic flow granularity	Private cloud	6.3
Monitoring-based traffic steering	Containers	6.4
Monitoring in public cloud	Public cloud	6.5
sFlow protocol analysis	Private cloud	6.6

the popularity of sFlow, in subsection 6.6 we focus on assessing the quality of this specific data-plane monitoring solution we adopted. In particular, we study the impact of sFlow configuration parameters on the accuracy of the measurements as well as on the overhead and sensitivity of sFlow. An auxiliary subsection 6.7 proves the statistical correctness of the results presented here, whereas a final discussion on how to choose the monitoring parameters is reported in subsection 6.8.

For the sake of clarity, Table 1 summarizes which experiments were conducted in which environment and the corresponding subsection reporting the results.

6.1. Unified monitoring for NFV/SDN infrastructure

The main aim of this section is to present the unified monitoring approach for both NFV and SDN resources and demonstrate that our solution is feasible and effective. Two traffic patterns were considered in this case, and are presented in Figs. 6 and 7, respectively. The first scenario assumes periodical traffic spikes of the same intensity, resulting from TCP sessions starting at times 5, 65, 125 seconds, lasting 30 seconds, and having unlimited throughput ("unlimited" by the application; the limit is a result of the saturation of link bandwidth). The second scenario considers traffic spikes with increasing intensity, given by TCP sessions starting at times 5, 65, 125, 185, 245, 305, 365, 425 seconds, lasting 30 seconds, and having throughput limited by the application that generates the traffic. The limits are set to 0.1, 0.2, 0.5, 1, 2, 5, 10 Gbit/s respectively to span the most typical transmission rates, while the last TCP session is again throughput-unlimited.

The figures present both the number of bytes received (histogram) and the CPU utilization (line) of a selected VNF. The received bytes are measured by aggregating values found in sFlow samples, corresponding to data forwarded

by the switch the VNF is attached to. The CPU load metric shows the percentage of time the CPU is busy in the specific machine running the selected VNF. As VNFs are running on separate virtual machines with dedicated resources, CPU denotes virtual CPU associated with a particular instance, that can be referred to as guest CPU.

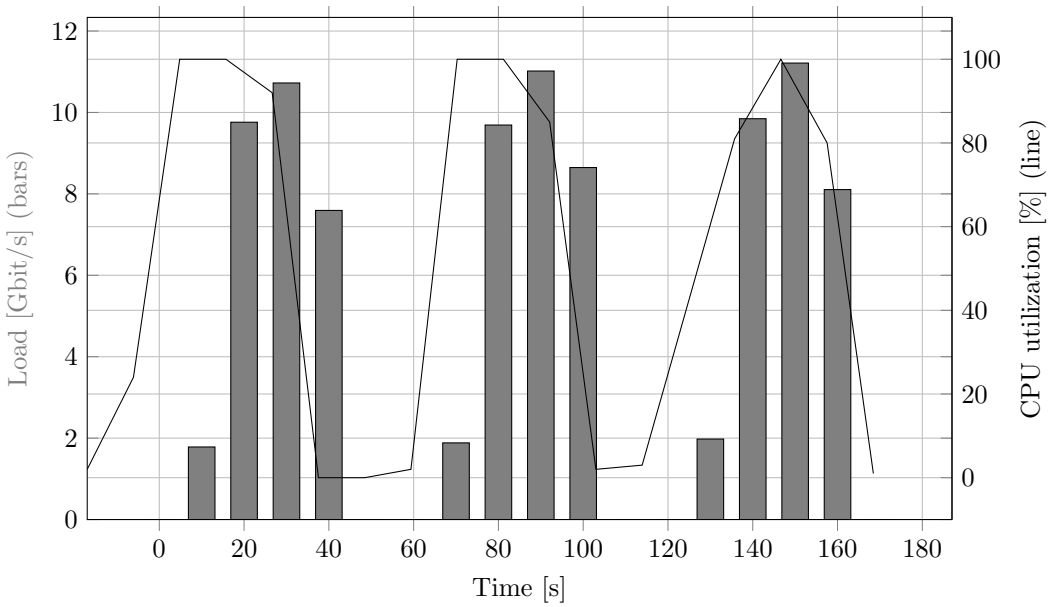


Figure 6: The amount of bytes received (histogram) and the percentage of time the CPU is busy (line) for a selected VNF stressed by traffic spikes of the same intensity.

The unified monitoring tool should be aware of the fact that network and computing resource utilization are usually strongly correlated, as indeed proved by the results shown in Figs. 6 and 7. Examples of cloud metrics as a function of network traffic for different VNFs were reported in literature [33]. In our experiments, we did not aim at analyzing specific VNF behavior, so the CPU load is originated only from handling packet forwarding operations on the generated flows. Such a simple approach demonstrates that our monitoring module is able to measure the load on both NFV (computing) and SDN (network) infrastructures in a unified way.

Similarly to the CPU load, the unified monitoring tool can measure other computing metrics provided, for example, by Ceilometer, such as memory or storage utilization, and expose those data to an orchestrator. This way, a unified and coordinated monitoring of traffic load and resource utilization

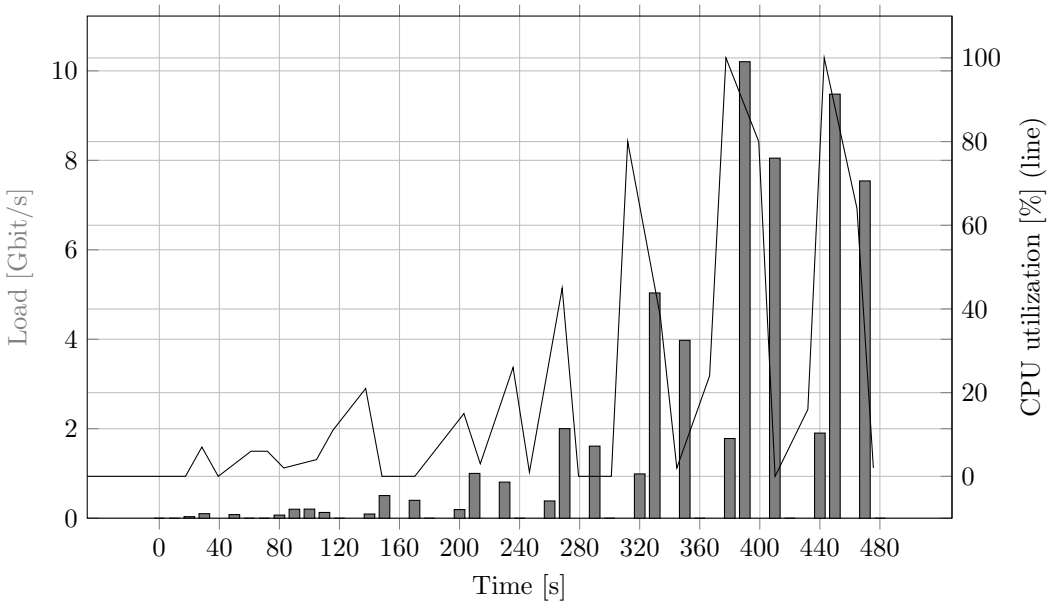


Figure 7: The amount of bytes received (histogram) and the percentage of time the CPU is busy (line) for a selected VNF stressed by traffic spikes with increasing intensity.

can ensure optimization from the global point of view.

6.2. Ceilometer response time

Collecting monitoring data through Ceilometer has some drawbacks in terms of response time, due to the multiple layers and interactions needed to obtain the measurements from the OpenStack platform. In our prototype implementation, when we tried to increase the update frequency of the measured values, the metrics provided by Ceilometer were available to the monitoring module even several minutes after they had actually been measured. Such a significant latency was, most probably, caused by the excessive queuing delay introduced by the data accumulators and the dispatcher used by the OpenStack components.

The latency with which measured data were accessible through the Ceilometer APIs is presented in Fig. 8 as a function of the experiment time. The default configuration of Ceilometer makes it collect measurements every 10 minutes. We changed this parameter to 10 seconds, which caused the continuous increase of the latency. The latency always increases linearly, with the slope displayed in the graph. However, the initial latency of the first sample

reflects the history of the OpenStack controller. Basically, after a reboot, the first measurement suffers of almost no additional latency, then the latency increases with time. The two curves show the difference in terms of latency between the scenario where the OpenStack controller has just been rebooted, and the one where the controller has been operational for some time. The slope of the latency curve depends on the processing power of the server hosting the OpenStack controller and its overall resource utilization. However, independently of slope and offset, this latency issue makes Ceilometer useless for the purpose of a real-time monitoring in the NFV environment. Therefore, monitoring data collected from Ceilometer can be used only for long term statistics purposes.

The last finding proves that the deployment of our monitoring solution as a standalone instance in modular architecture is reasonable. Thanks to such an approach, we can easily integrate the monitoring module with any other solution collecting metrics from the underlying NFV infrastructure. Furthermore, it is possible to integrate more than one monitoring sources. For example, we can use one tool (e.g. Ceilometer) to collect a wide variety of available metrics for latency-agnostic, long-term analysis purposes, and any other lightweight and fast tool which can measure simple CPU, RAM or network utilization for the purpose of real-time traffic steering. An example of such a tool, in the network traffic domain, is sFlow, which we adopted and carefully studied in our PoC implementation.

6.3. Traffic flow granularity

Implementing our monitoring module outside of the control plane enables freedom in terms of integration with external tools. It is an important advantage over existing solutions that are part of the control plane. However, to fully benefit from this fact, those tools must be selected according to the particular needs. Therefore, in this section we present experiments aimed at studying the difference, in terms of granularity of the traffic flow measurements, between the monitoring based on sFlow and Ceilometer.

The sFlow agent was configured with sampling ratio $N = 10$ and sample aggregation interval $C = 10$ s while the EWMA was run with weighting coefficient $\alpha = 0.3$. Three flows were activated in the network with a fixed delay of 60 seconds between consecutive flows. The throughput generated by each flow was equal to 30, 60 and 90 Mbit/s, respectively. However, two scenarios were considered. In the first one, all three flows were exchanged

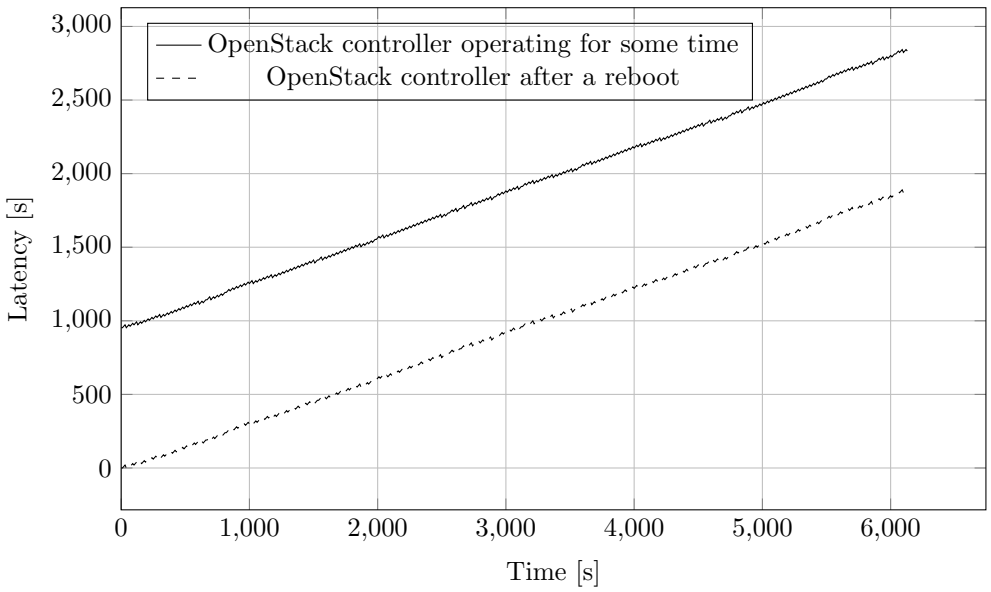


Figure 8: The time after which Ceilometer data is available for collection.

between different pairs of VMs. As can be observed in Fig. 9, the measurements provided by both sFlow and Ceilometer match quite well to the actual network load. sFlow configured with the given parameters is able to reflect traffic changes faster but, at the same time, it is slightly less precise.

However, this traffic scenario does not allow to observe the differences between sFlow and Ceilometer in terms of flow granularity. This is why the second scenario also assumes that the aforementioned three flows are sent between a single pair of VMs. Results are presented in the Fig. 10. On one hand, sFlow is able to distinguish different flows based on the IP address of the endpoints. On the other hand, Ceilometer monitors traffic with the granularity of the network interface and it is not able to distinguish every single flow, providing thus cumulative results. The monitoring modules of numerous cloud orchestrators operate in the same manner. Therefore, it is an important conclusion that cumulative interface-based monitoring tools are less effective when it is needed to distinguish different flows between a single VM pair. Conversely, the modular architecture of our monitoring module shows its advantages in terms of new features and degrees of freedom in an NFV infrastructure where multiple VNFs are running on a single machine.

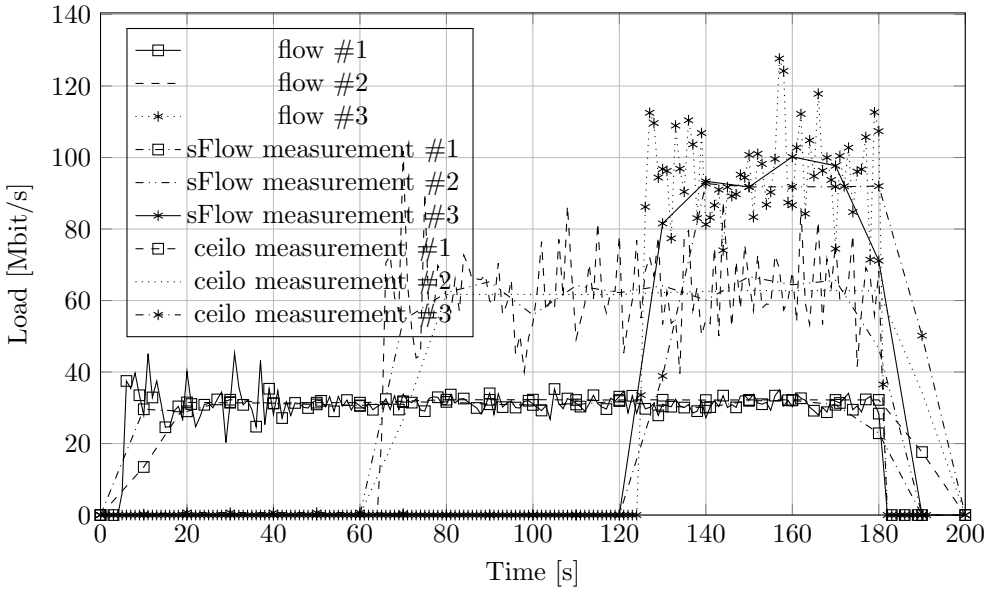


Figure 9: Granularity of Ceilometer and sFlow with $N = 10$, $\alpha = 0.3$ and $C = 1$ s, compared to the actual network load between different pairs of VMs.

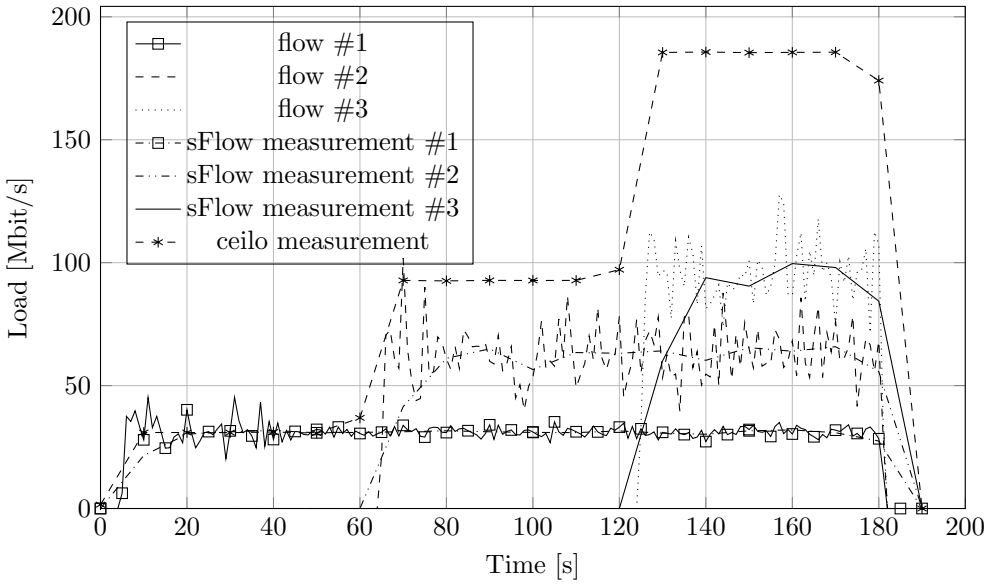


Figure 10: Granularity of Ceilometer and sFlow with $N = 10$, $\alpha = 0.3$ and $C = 1$ s, compared to the actual network load between single pair of VMs.

6.4. Monitoring-based traffic steering

The testbed presented in Section 5.2 was employed to perform proof-of-concept experiments on the ability of the proposed monitoring module to constructively interact with an SDN controller in order to support the dynamic steering of traffic flowing in the network, aimed at optimizing the utilization of physical and virtualized networking and computing resources. Traffic is generated with iperf using TCP with a specific throughput target, depending on the experiment.

Two case studies are examined. In the first one, called *choose VNF*, traffic steering is performed to mitigate congestion of a particular VNF instance. This is possible thanks to the monitoring capabilities of the proposed module in a NFV domain. The desired action is to distribute traffic among different replicas of the same VNF, no matter what is their location considered from the network perspective. In the second case study, called *choose path*, the aim of traffic steering is to avoid network congestion based on the measurements performed on a network node representing an SDN domain. The desired action in this scenario is to change the path of traffic to distribute the load between different network links.

In both cases, the objective is to achieve the maximum overall throughput from source to destination, while respecting service policies, such as the requirement to cross a given VNF. This is achieved in the *choose VNF* scenario by finding the VNF replica with sufficient computing resources even if the traffic is directed through the same network path. On the contrary, in *choose path* the goal is achieved by finding alternative network links with sufficient resources even if the destination node is the same. Therefore, the monitoring module is able to trigger traffic steering based on the measurements performed in both NFV and SDN domains, proving that it is unified and takes advantage of the involved cooperating paradigms.

The logical topology, shown in Fig. 4, aims at providing a testbed that can be used to run experiments in both case studies. As results we consider load distribution in both scenarios, comparing it to the generated load pattern. Our monitoring module was configured with $N = 10$, $\alpha = 0.4$ and $C = 1$ s.

6.4.1. Case study: choose VNF

All traffic is required to cross a replica of a given VNF. Any network function can be considered, for example, traffic shaping, packet inspection for an intrusion detection system, or transcoding for multimedia traffic flows. In our PoC we measure the traffic load on the interface of the VNF, which is

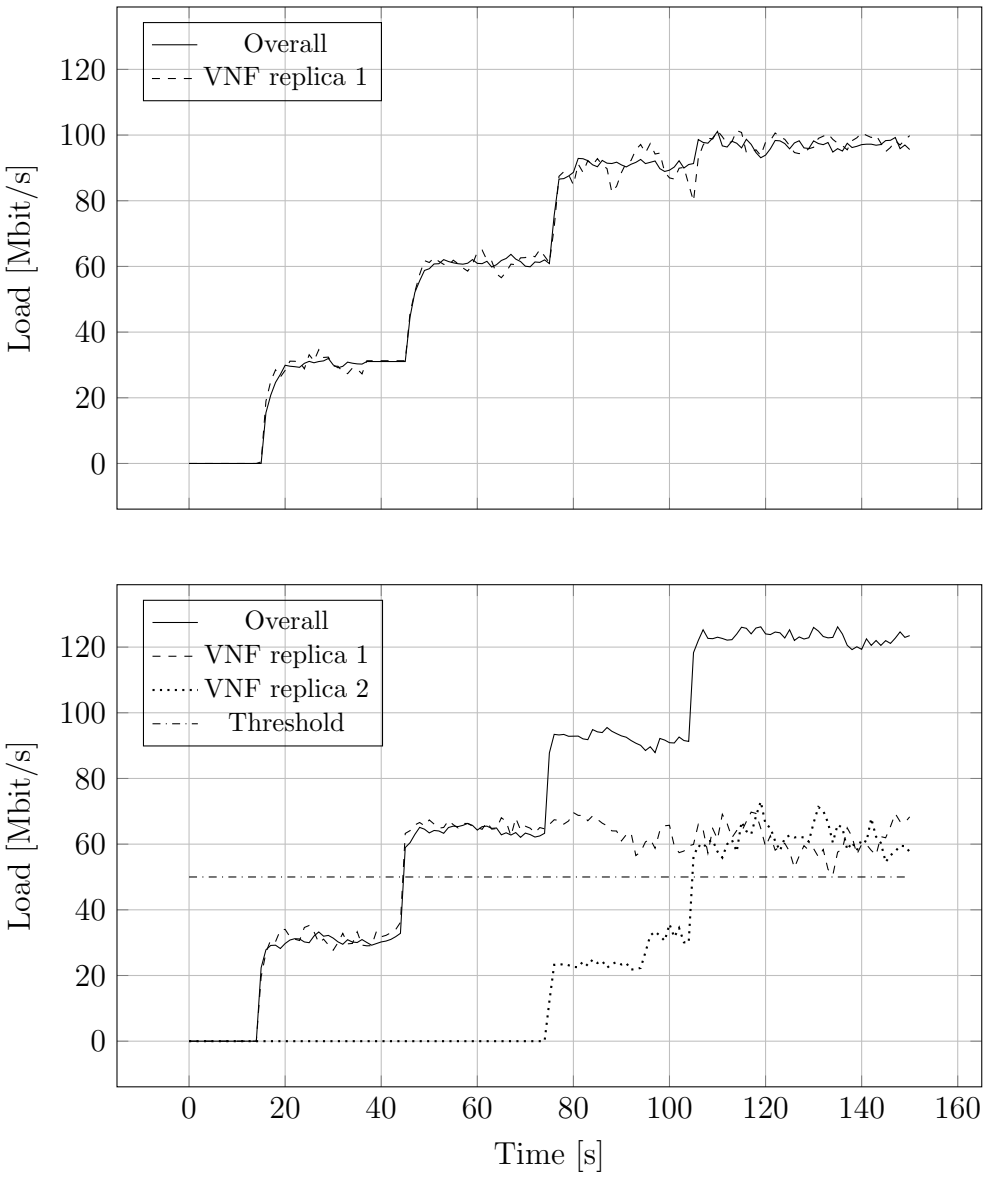


Figure 11: Case study: *choose VNF*. Load of VNFs without traffic steering (upper) and with traffic steering based on the monitoring in NFV domain (lower).

strongly correlated to the load in computing delay as it was shown in Fig. 7. We do not consider any particular service and a simple packet forwarding function is performed. This simplification should be considered as a best case

because any other function requires more computing resources for each packet received. Therefore, it is reasonable to assume that the measurements taken at network level can also give an insight of the computational burden on the VNF, justifying the actions taken to optimize the load. In these experiments, a total of four iperf sessions were launched from SRC to DST, all of them using TCP and aiming at a throughput of 30 Mbit/s. These sessions were activated sequentially at intervals of 30 seconds, and configured to last until the end of the experiment.

To begin with, a baseline experiment was run, in order to assess the behavior of the system when no steering was applied. All the generated traffic was crossing the same replica of the VNF. The evolution in time of the data rate of the traffic flow from the source host to the destination host is shown in Fig. 11. As expected, after the fourth flow started, i.e., at $t = 105$s, the traffic saturated the capabilities of the VNF instance, and the flows needed to compete for the resources, causing the total throughput to be limited by the capacity of the link to the VNF, i.e., 100 Mbit/s. In the second experiment, this potential deterioration was avoided by taking advantage of the proposed monitoring module. The traffic was initially steered through the first replica of the VNF. The monitoring module kept tracking of the increase in resource utilization in the VNF, verifying that the traffic was below a predetermined *warning threshold* set at 50 Mbit/s. When this threshold was exceeded, i.e., at $t = 45$s, the monitoring module interacted with the VIM to find out the location of a second replica. Based on that, the monitoring module instructed the SDN controller to steer the traffic accordingly, in order for it to cross the second replica. This way, even when the four flows are running at the same time, they do not have to compete for the shared resources, and the full overall throughput can be achieved.

6.4.2. Case study: choose Path

This case study aims at highlighting the benefits of dynamic traffic steering over multiple paths in the network, in case switches are overloaded or not utilized optimally.

Similarly to the previous case, a total of four iperf sessions were launched from SRC to DST, all of them using TCP, this time aiming at a throughput of 200 Mbit/s. Once again, the sessions were activated sequentially at intervals of 30 seconds and configured to last until the end of the experiment. Initially, the traffic crossed the *upper path*, with reference to the logical topology in Fig. 4, where the link capacity is 500 Mbit/s.

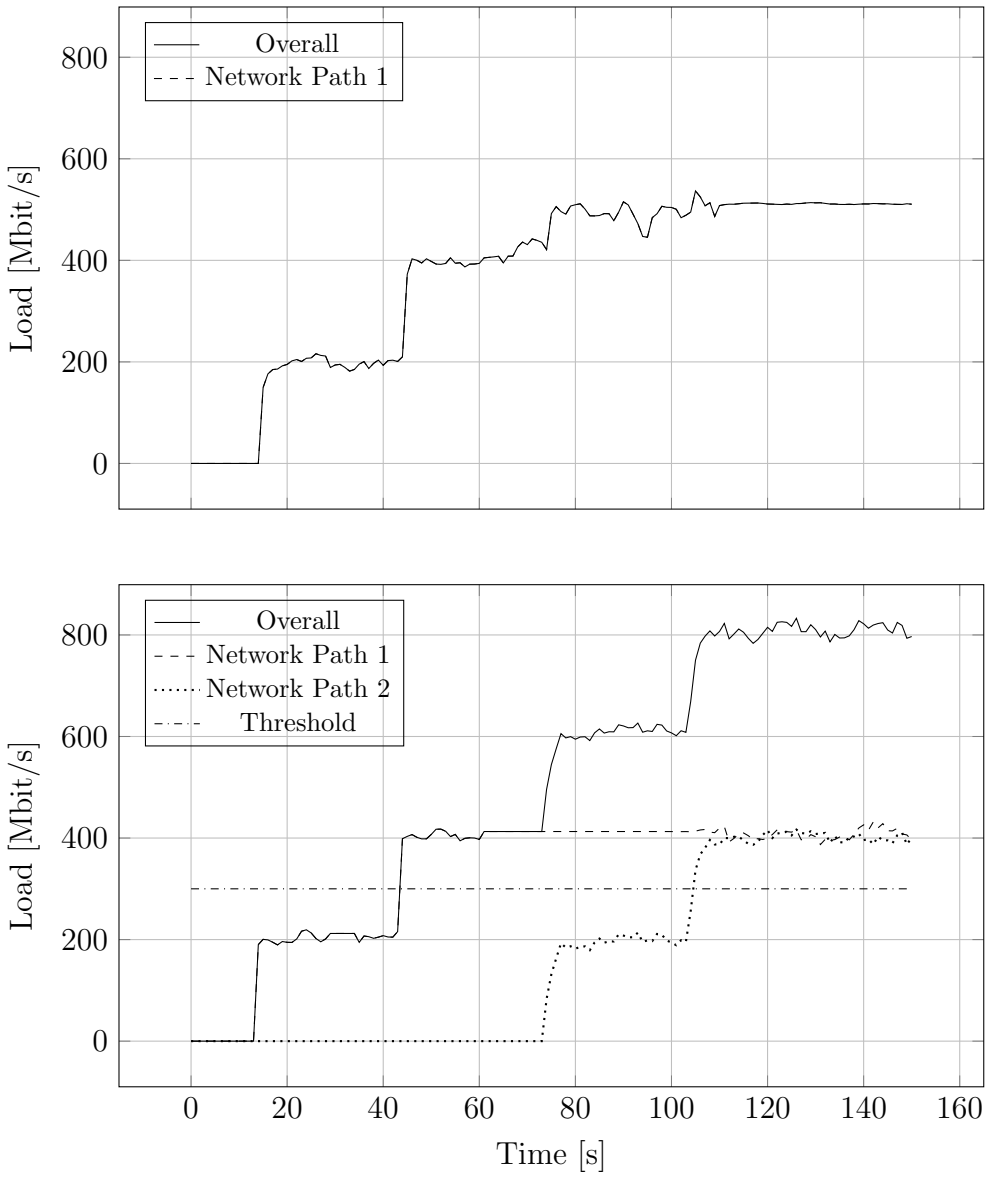


Figure 12: Case study: *choose Path*. Load of switch interfaces without traffic steering (upper) and with traffic steering based on the monitoring in SDN domain (lower).

Similarly to the case examined in Section 6.4.1, the temporal evolution of the data rate of source-to-destination traffic flow is shown in Fig. 12. The baseline experiment, without traffic steering, shows that the links became

saturated after the third traffic flow started, at $t = 75$s, then the combined throughput of flows was limited to the capacity of the link with smaller capacity, i.e., 500 Mbit/s, which acted as a bottleneck. In fact, the impact of the fourth flow, starting at $t = 105$s, is practically invisible. The proposed monitoring module enables avoiding this service degradation. In the second part of the experiment, the monitoring module kept tracking of the increase in network resource utilization, considering the load on the ports of the switch. Again, when a predefined threshold was exceeded, after the second flow started at $t = 45$s, the monitoring module triggered traffic steering mechanisms. This time it instructed the SDN controller to steer the traffic through the alternative path. The traffic was then directed through the not congested *lower path* and throughput of flows was no longer limited. The overall throughput reached its maximum value, equal to the sum of the throughput of the four flows.

6.5. Monitoring in public cloud

The fundamental aim of the deployment of our module in the public cloud is to prove its versatility and portability. Results presented below not only confirm these features, but also reveal additional advantages in terms of valuable module's applications in the public cloud. By successfully deploying our module we understood that we were able to seamlessly migrate our solution towards a general purpose environment not fully managed by us. We also validated our module and verified that the traffic was properly sampled by comparing the measurements obtained against those provided by the renowned sFlow collector sFlow-RT[15]. sFlow-RT is supported by the company which invented the sFlow protocol and is widely used in numerous open-source projects. Thus, the sFlow-RT is a reliable tool to justify that both steady traffic and periodic bursts were correctly measured by our module[16].

To carefully validate and present some additional advantages of the monitoring module two research scenarios were conducted. Both traffic patterns were identical to those generated in the private cloud testbed (Section 6.1). The rationale is to ensure correspondence between different deployment environments. Just as a reminder, the first scenario assumes periodical traffic

[15]https://sflow-rt.com/index.php

[16]The results of this validation are not reported here due to space limitation.

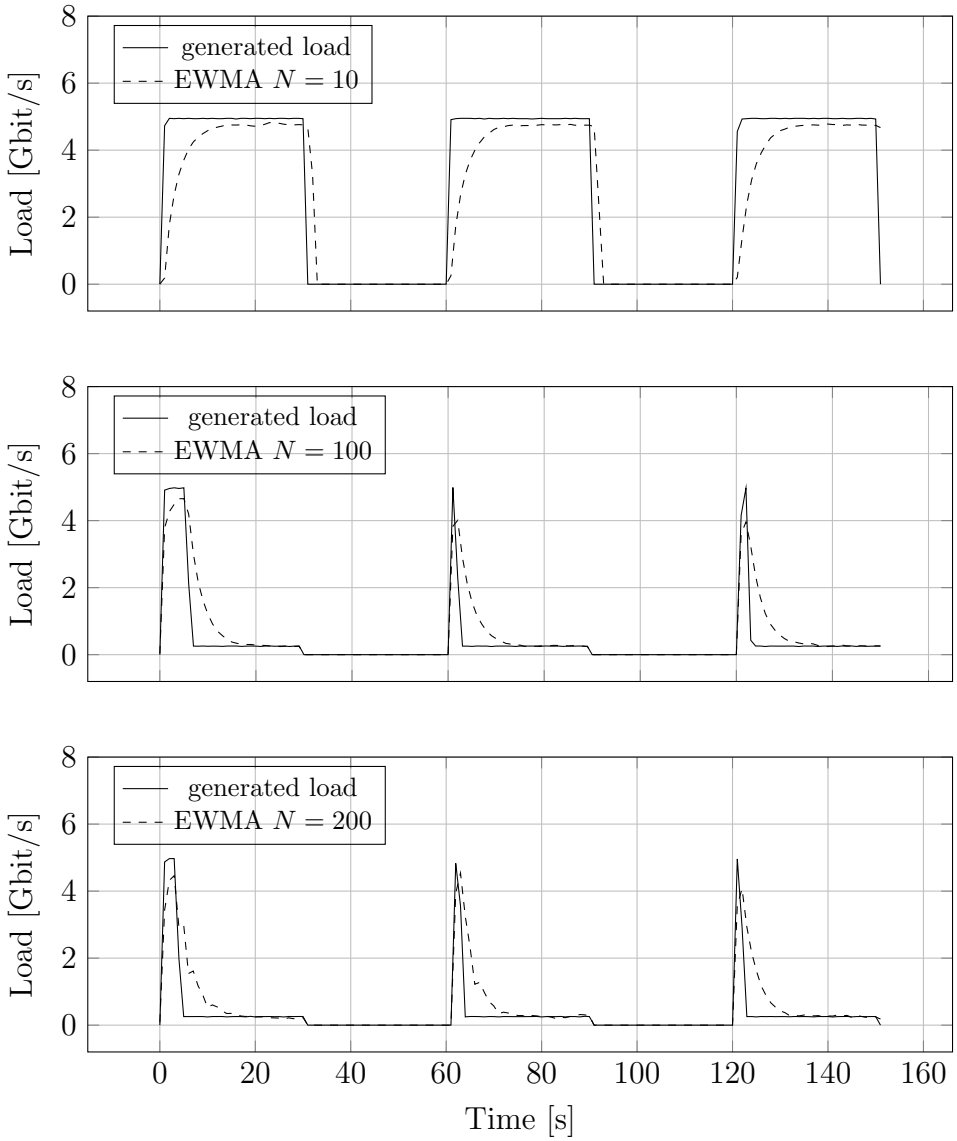


Figure 13: Accuracy of the monitoring module in the first scenario and under different sampling ratio values N, with $\alpha = 0.3$ and $C = 1$ s, compared to the generated load pattern.

spikes of the same intensity, resulting from TCP sessions starting at times 5, 65, and 125 seconds, lasting 30 seconds, and having unlimited throughput ("unlimited" by the application; the limit is imposed by the public cloud

environment). The second scenario considers traffic spikes with increasing intensity, given by TCP sessions starting at times 5, 65, 125, 185, 245, 305, 365, 425 seconds, lasting 30 seconds, and having throughput limited by the application that generates the traffic. The limits are set to 0.1, 0.2, 0.5, 1, 2, 5, 10 Gbit/s respectively to span the most typical transmission rates, while the last TCP session is again throughput-unlimited.

Figs. 13 and 14 show the accuracy of the module in the two scenarios, respectively. We investigate the effect of different values of N (equal to 10, 100, and 200), with $\alpha = 0.3$ and $C = 1$ s, comparing it to the generated load pattern. The most important conclusion is that the measurements collected by the module correctly follow the generated traffic pattern in both scenarios. This proves the applicability of the proposed solution in the public cloud environment. Similarly to what reported for the private cloud testbed, in the public cloud environment changing the value of N creates an opportunity to balance the tradeoff between measurement accuracy and additional resource utilization (detailed studies on that issue are presented in Section 6.6.2). The higher N, the smoother and more precise the measurements, but also the monitoring process becomes more demanding.

Additionally, thanks to the collected results, we can formulate interesting insights regarding the public cloud environment. The cloud platform limits the throughput of the iperf session at two levels. Namely, virtual machines are eligible to generate traffic bursts reaching up to 5 Gbit/s. This is fully compatible with the specifications of the selected types of virtual instances. However, persistent traffic bursts are further limited to a bitrate much lower than 1 Gbit/s.

The plots in Fig. 13 show how N impacts the maximum time that bursty traffic can be generated before the second-level bitrate limit is applied. However, the real reason of the differences lies in a limitation imposed by the cloud provider. Being a proprietary solution, such limitation is not publicly revealed. Comprehensive investigations led us to the following conclusions: a certain amount of credits is given to each virtual machine to send traffic with the maximum throughput declared in the specifications. Once the granted credits are consumed, more restrictive limits are applied. Since credits are replenished with a rate unknown to the user, the longer an interface is idle between traffic bursts, the longer it will be eligible to generate again traffic at full rate.

The graph at the top of Fig. 13 presents the results of the experiment that was conducted after a long idle period, thus when all credits were available.

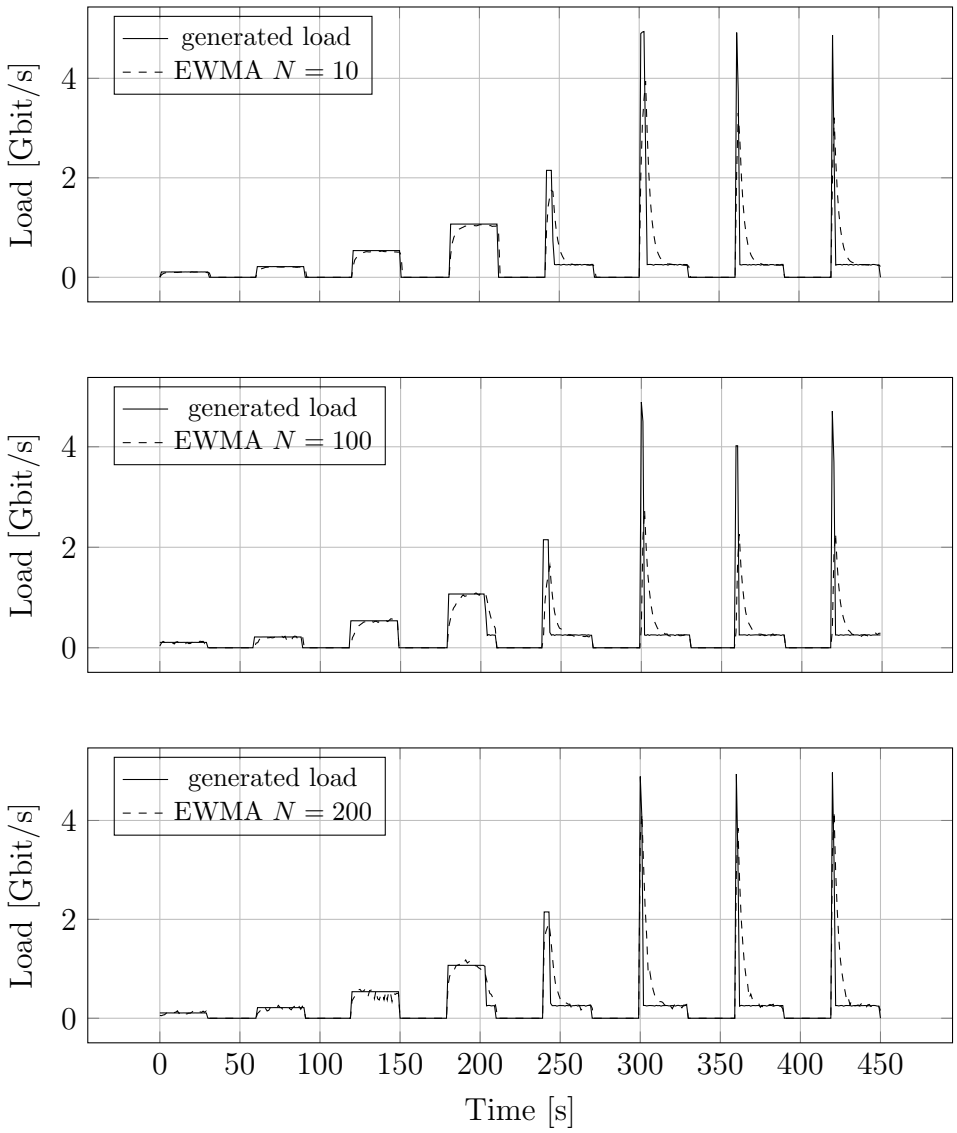


Figure 14: Accuracy of the monitoring module in the second scenario and under different sampling ratio values N, with $\alpha = 0.3$ and $C = 1$ s, compared to the generated load pattern.

This was certainly not the case for the experiments presented in the other graphs, where after a few seconds of transmission at full rate the throughput dropped to a few hundreds of Mbit/s, clearly showing a traffic shaping effect imposed by the public cloud infrastructure provider. To eliminate this effect

a proper idle period between experiments should be introduced. However, we intentionally present this example to show that the real performance of the public cloud network is much more sophisticated and less predictable than the performance of an experimental environment completely under our control. We validated our findings conducting extensive experiments that we do not present here for the sake of brevity.

Results collected in the second scenario, presented in Fig. 14, confirm the existence of two levels of bitrate limits and the fact that N does not impact the allowable duration of the traffic bursts. Additionally, it can be concluded that the burst throughput directly affects the credits consumption rate. The mechanism intuitively allows for longer bursts, if only the throughput of the burst is lower.

It was shown that our monitoring module may help discover the real performance of the cloud network. Experimenting with different traffic patterns would allow to obtain the amount of credits available for each instance, the credit renewal pace, and the impact of the transmission rate on the credit consumption. This is especially important as cloud providers are usually vague in documenting the available network resources while consumers may be interested in actual limitations and SLA fulfilment.

As a final note, to deploy our PoC we used the Terraform[17] tool and its HashiCorp Configuration Language (HCL). This proves that our module can be integrated with widely used solutions aimed at automation of deployment in various environments, like public, hybrid, or private clouds.

6.6. Experiments regarding the sFlow protocol

In our prototype implementation, the sFlow protocol was selected, among a number of tools gathering flow statistics, to sample directly from network devices. The choice has been already justified, however, due to the popularity of sFlow it is reasonable to carefully assess the quality of this specific data-plane monitoring solution.

6.6.1. Impact of sFlow parameters

A first experimental scenario was aimed at investigating the fundamental properties of the sFlow protocol as a function of three parameters: the sFlow sampling ratio (N), the sample aggregation interval (C) and the α coefficient

[17] https://www.terraform.io/

of the EWMA. We generated a simple traffic pattern including three iperf data flows starting at times 5 s, 65 s and 125 s, respectively. Each flow lasted until the end of the experiment and generated a throughput of 30 Mbit/s. As a result, a stepwise increasing load curve was imposed.

Figure 15 shows the accuracy of the EWMA of the sFlow samples for different values of N, with $\alpha = 0.3$ and $C = 1$ s, comparing it to the generated load pattern and the instantaneous sFlow sample values collected for $N = 10$. Figure 16 reports the same measurements for $C = 10$ s. For $C = 1$ s, it can be seen that the EWMA shows higher variability when N is higher, because sFlow collects less samples and thus is more affected by temporal traffic fluctuations. Furthermore, the higher the generated load, the wider the fluctuations, for any value of N. Thus, the sampling ratio should be adjusted not only to the required accuracy, but also to the absolute traffic load. In the case presented in Fig. 15, choosing $N = 20$ gives quite stable values when the load is small (30 Mbit/s), but when the load increases (90 Mbit/s) even $N = 5$ results in some significant fluctuations. Changing the sample aggregation interval to $C = 10$ s reduces the number of sFlow samples, smoothing out all curves (Fig. 16). The obtained measurements are therefore less sensitive to load fluctuations, but at the cost of worse responsiveness. Even in the relatively static case we are considering, where new flows arrive only every 60 s, sampling each packet ($N = 1$) does not converge sufficiently fast to provide reliable monitoring results. The first sFlow samples collected after a significant load change (cross-shaped markers at 70 s and 130 s in Fig. 16) provide incorrect values as the collected aggregate is affected by historical values. Thus, increasing the sample aggregation interval is reasonable only in case of almost static network load. Furthermore, as for $C = 10$ s the accuracy does not significantly depend on the sampling ratio, higher values for N should be considered. The first general conclusion is that any tool considered for flows statistics gathering should be adjustable to the expected load and particular needs.

Based on the results presented in the Figs. 15 and 16 it is also possible to draw conclusions regarding the delay after which the collected measurements are available and sufficiently credible. This delay may be critical, for example, when monitoring is expected to feed a control plane that dynamically reoptimizes the network configuration. The delay comprises both communication time between the monitoring module and control plane as well as the delay introduced by the monitoring module itself. The former component depends on the infrastructure and, in most cases, is expected to be negligible

when compared with the second one. The monitoring module introduces a significant delay that depends on the configuration parameter. Changing the value of C from 1 to 10 intuitively increases by a factor of 10 the interval with which samples are generated by the sFlow collector (as it appears by comparing the density of sFlow samples presented in Figs. 15 and 16). Thus, the value of C represents a lower-bound of the delay introduced by the module. One must also note that receiving sFlow counters does not necessarily mean that they are always useful, for example to perform network reoptimization. Namely, as presented in Fig. 16 the first sFlow counters after each increase of traffic load report worthless and stale values that will not trigger the control plane to reoptimize the network. Considering the EWMA with increasing value of N makes this issue even more important.

For both $C = 1$ s and $C = 10$ s, sampling every packet ($N = 1$) provides accurate results until the third flow is injected in the network. At that point, the testbed is no longer able to measure network load accurately. This is caused by several reasons. Firstly, the implementation of the sFlow collector software is sub-optimal and causes some unnecessary load on the research infrastructure. Secondly, $N = 1$ means that the monitoring traffic doubles the amount of data traffic and all that communication must be handled in our testbed. Finally, the testbed has some computational limits as we operate in a virtualized infrastructure. Anyway, the scenario of sampling every packet presents potential overutilization of the monitoring infrastructure and should be considered as an unrealistic case which defeats the purpose of applying packet sampling at all. It was demonstrated in our work only for the sake of completeness, but it also proves that sFlow is not a good choice if the objective is to carefully analyse each packet.

Under particular circumstances, the widely-adopted sFlow protocol may not be able to meet monitoring requirements and other mechanisms should be considered. Such demanding use cases may be caused by applications imposing strict delay and jitter measurements. Also, this observation further confirms that the inter-component independence assumed in the proposed monitoring solution may bring significant and valuable advantages with respect to existing solutions that are integrated in the control plane.

As already stated, the EWMA was introduced as an alternative to the counter sampling mechanism aimed at making measurements less sensitive to temporal load fluctuations. Figures 17 and 18 show the sFlow EWMA accuracy for different values of α with $N = 10$, assuming $C = 1$ s and $C = 10$ s, respectively. When $C = 1$ s, smaller values of α cause the

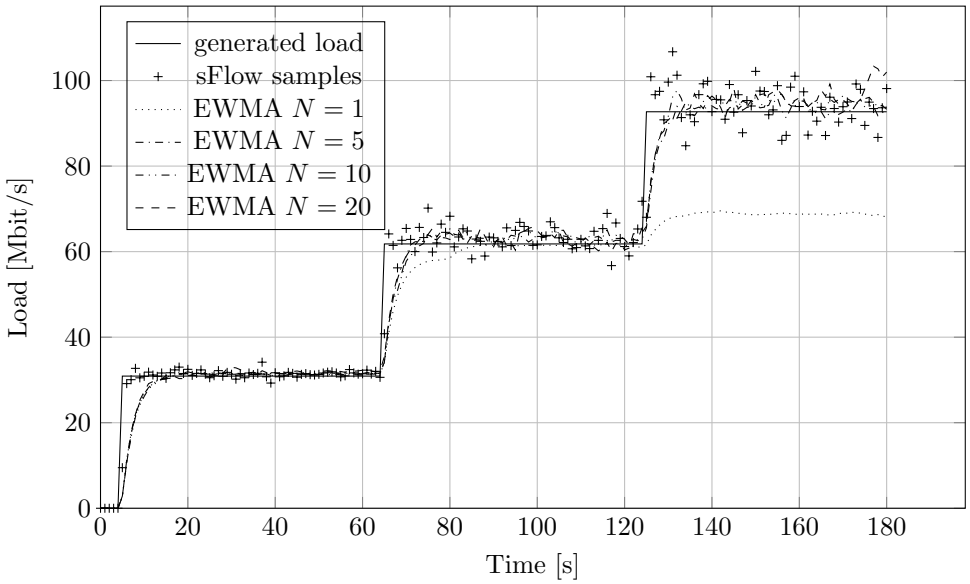


Figure 15: Accuracy of the sFlow EWMA under different sampling ratio values N, with $\alpha = 0.3$ and $C = 1$ s, compared to the instantaneous sFlow samples and the generated load pattern.

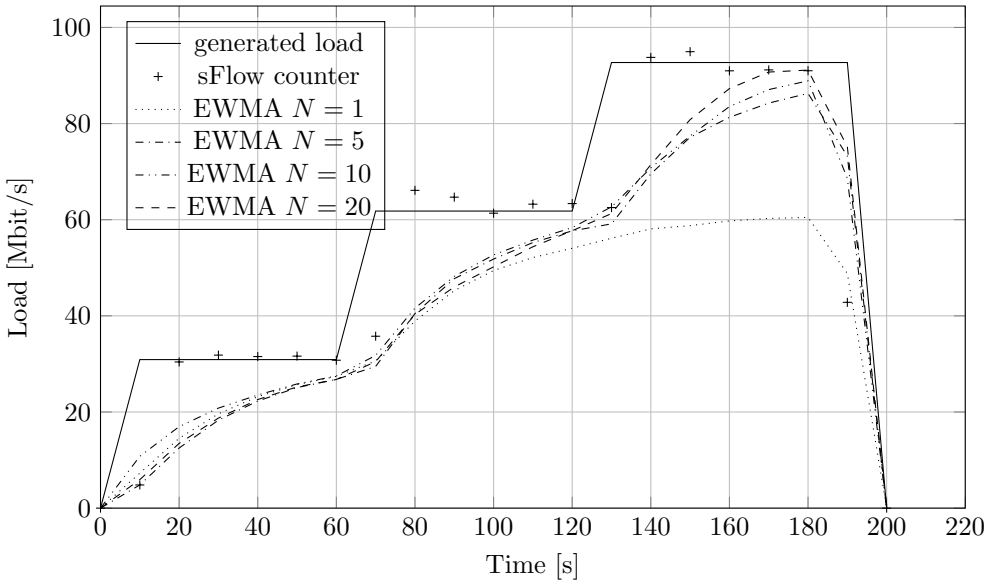


Figure 16: Accuracy of the sFlow EWMA under different sampling ratio values N, with $\alpha = 0.3$ and $C = 10$ s, compared to the instantaneous sFlow samples and the generated load pattern.

EWMA to converge slower to the current load as historical samples have higher weight. In particular, $\alpha = 0.02$ and $\alpha = 0.05$ give significantly inaccurate results even in a scenario with slow load changes. However, the α parameter may be adjusted to provide substantial improvement in comparison to the sFlow samples, which are heavily affected by load fluctuations, without harmful degradation in terms of responsiveness ($\alpha = 0.2$ is a good choice in our scenario). Analyzing the results obtained for $C = 10$ we can see that the measurements are averaged by both the EWMA and the sFlow counter sampling mechanisms. In such a case the measured values are practically unaffected by temporal traffic variations (including the sFlow samples that are aggregated by the counter sampling mechanism), but at the same time they converge very slowly to the current load levels. The results of our experiments proved that the EWMA can be an effective method to make sFlow results less sensitive to the temporal load variations while keeping good responsiveness, with a proper adjustment of the sFlow parameters to the specific traffic load dynamics. In particular, α should be assigned higher values if the sFlow counter sampling mechanism is used to aggregate the measured samples. These conclusions are generally valid, as the EWMA algorithm, similarly to our monitoring module, is universal and can be integrated with tools other than sFlow.

6.6.2. Evaluation of sFlow protocol overhead

The sFlow sampling ratio N affects not only the accuracy (as described in Section 6.6.1) but also the overhead of the sFlow-based measurement process, as computed in eq. (2). Network nodes and collector exchange samples of packets from the overall traffic, plus the sFlow protocol header and the sample headers, as reported in eq. (3). There is an additional parameter defining the size B of the packet sample to be included in the sFlow packet, and this parameter directly affects the overhead. Depending on the processing performed in the sFlow collector, different sample sizes may be required. However, in our case we only measure the amount of traffic at the flow level and do not need to analyze any extraordinary sFlow configurations. Therefore, we decided to send the first $B = 128$ bytes of each sampled packet. The presented results were collected with sample aggregation interval $C = 1$ s. One must also note that changing the α parameter does not impact the overhead of the sFlow protocol, as α affects only how samples are processed in the collector and does not change the amount of data that network nodes send to the collector. Therefore, we present the overhead results as a function

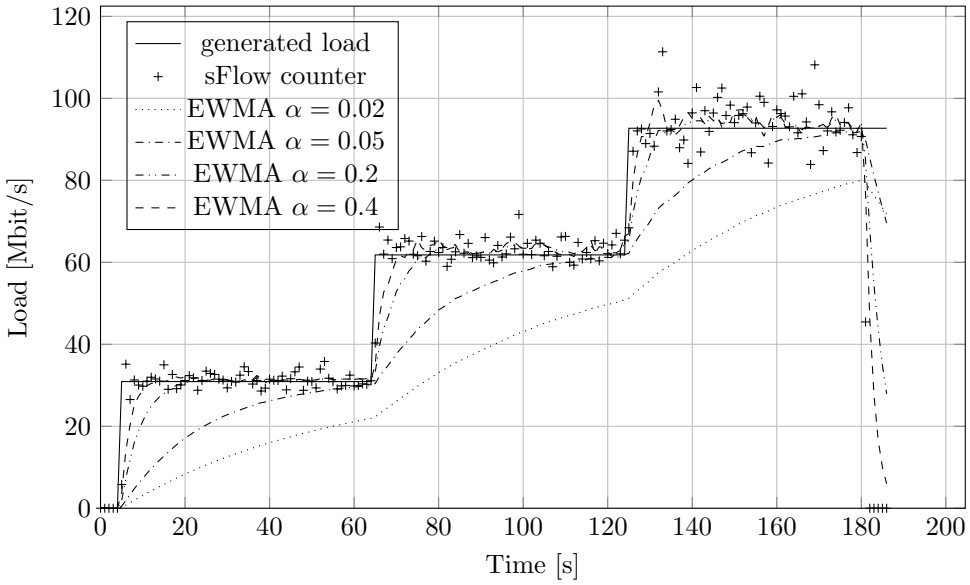


Figure 17: Accuracy of the sFlow EWMA under different moving average values α, with $N = 10$ and $C = 1$ s, compared to the instantaneous sFlow samples and the generated load pattern.

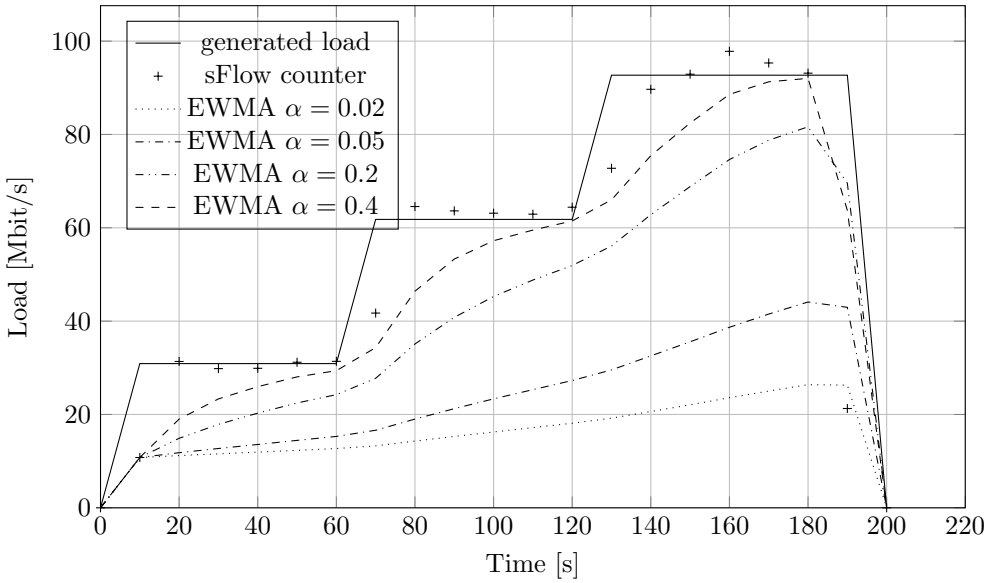


Figure 18: Accuracy of the sFlow EWMA under different moving average values α, with $N = 10$ and $C = 10$ s, compared to the instantaneous sFlow samples and the generated load pattern.

of the sFlow sampling ratio parameter (N).

The sFlow overhead is evaluated under a simple yet sufficient traffic pattern. Namely, it includes a single iperf session of 1 Mbit/s throughput ran for 60 seconds between two different virtual machines. Figure 19 presents the total absolute amount of sFlow signaling traffic being exchanged between network nodes and the sFlow collector. The results show how the sFlow protocol signalling decreases with the sampling ratio N.

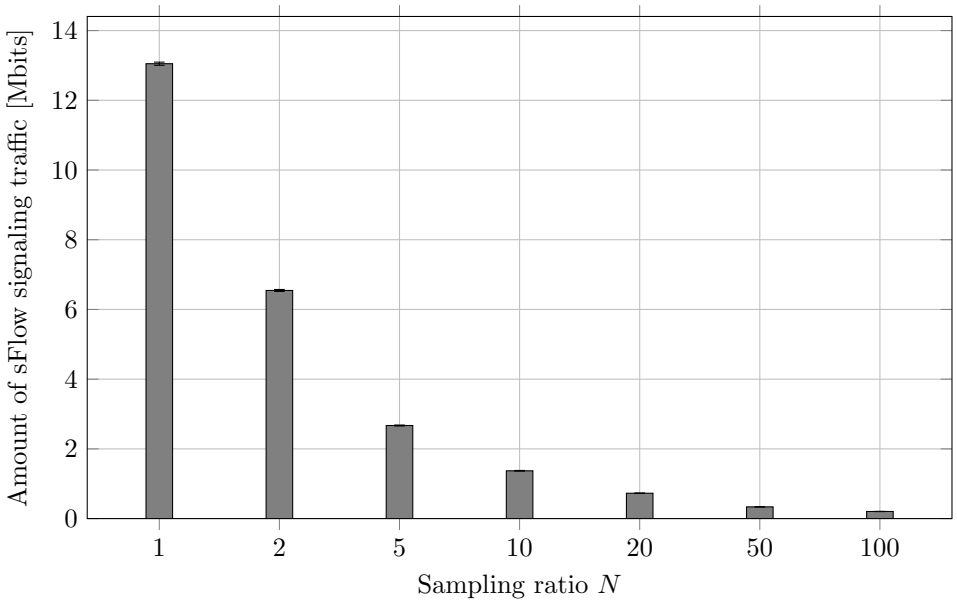


Figure 19: Total absolute amount of sFlow signalling traffic with a single 1 Mbit/s iperf session running for 60 seconds.

In Table 2 we report the relative overhead, as a percentage of the total traffic exchanged, for different values of N. The measured relative overhead was obtained by capturing the traffic at the output of a virtual switch and reporting the percentage of sFlow traffic over the total traffic captured. The estimated relative overhead was instead obtained by applying eq. (2) and assuming the traffic parameters generated by the iperf session, which was configured to send packets of $L_i = 1432$ bytes, $\forall i = 1, \ldots, n_p$. Considering the case when $N = 1$, each data packet was sampled by sFlow and the first $B = 128$ bytes were sent to the sFlow collector. We used the sFlow protocol version 5, which adds to each data sample some meta-data that amount to a sample header of $h_{\text{sample},i} = 88$ bytes. Therefore, each sample carried by

Table 2: Measured and estimated relative overhead introduced by the sFlow protocol under different sampling ratio values N

N	Measured relative overhead	Estimated relative overhead
1	15.07%	13.74%
2	8.17%	7.37%
5	3.50%	3.09%
10	1.83%	1.57%
20	0.98%	0.79%
50	0.46%	0.32%
100	0.28%	0.16%

the sFlow packet adds a total of 216 bytes to the packet size. Considering an sFlow packet carrying $n_p = 6$ samples and adding the standard protocol header sizes (i.e. $h_{\text{eth}} = 14$ bytes, $h_{\text{ip}} = 20$ bytes, $h_{\text{udp}} = 8$ bytes, and $h_{\text{sflow}} = 28$ bytes), then the size of the sFlow packet is $L_{\text{sflow},p} = 1366$ bytes. From eq. (2) we obtain $O_{\text{sflow},p} = 13.72\%$ for $n_p = 6$. While more than 90% of the captured sFlow packets carried 6 samples, the rest included from 1 to 5 samples. The estimated relative overhead reported in Table 2 shows the weighted average of $O_{\text{sflow},p}$ for $n_p = 1, \ldots, 6$. The approximate formula (2) is very close to but underestimates the measured relative overhead. The reason is that the captured traffic included also some packets due to background traffic and whose size was smaller than 1432 bytes, thus increasing the actual overhead. However, the formula is able to capture quite well the behavior of the overhead as a function of the sampling ratio, with an approximation error of 1.33% in the worst case ($N = 1$). Thus, one can easily estimate the expected overhead, consider the trade-off against accuracy, and properly configure the monitoring tool making it suitable to the requirements of a particular SDN/NFV infrastructure. Applicability and usefulness of the proposed formula are further increased by the fact that it can be adjusted to any other data plane monitoring solutions, making it possible to theoretically estimate the monitoring overhead.

We validated our conclusions under different traffic scenarios (including the one considered in Section 6.6.1). Based on the results we can confirm that changing the amount of traffic being exchanged in the network impacts only the absolute value of signaling traffic and does not affect the relative overhead.

6.6.3. Sensitivity study

As previously mentioned, the counter sampling parameter can be increased in order to reduce the sFlow overhead. However, careful considerations are required in doing so, due to the impact on sensitivity analyzed in this section. We performed a set of experiments where we generated two traffic flows. The first one was sending 30 Mbit/s for the whole experiment duration, while the second one produced traffic spikes of 60 Mbit/s starting at times 25, 45, 65, 85 and 105 seconds, and lasting 1, 2, 3, 4 and 5 seconds, respectively.

Figures 20 and 21 show the accuracy of the EWMA applied to sFlow sampled measurements for different values of α, with $N = 10$, and $C = 1$ s and $C = 10$ s, respectively. For $C = 1$ s, even the instantaneous sample values collected by sFlow are partially affected by this basic one-second aggregation performed by the protocol, as the measured spike intensity is higher when the spike duration is longer. Also, Figure 20 shows an apparent overshoot of the measured data in correspondence to the traffic spikes (the actual traffic never exceeds $30 + 60 = 90$ Mbit/s). However, that is only another effect of the mentioned basic aggregation that directly affects the interval between collected samples. In fact, in presence of abrupt variations of the traffic, the protocol will have to associate a large quantity of samples to the sampling instant that is closest to the traffic variation. This may result in associating to that instant a larger quantity of samples than necessary, resulting in an apparent overshoot of the measured data, which then is smoothed out in the following sampling instants. Moreover, the spikes are hardly detected by the EWMA with the α parameter equal to 0.02 and even 0.05. Choosing $\alpha = 0.2$ or 0.4 allows to detect traffic spikes, although with reduced accuracy. The measurement is even less sensitive when $C = 10$ s, as the increased sample aggregation interval makes historical samples more significant. In fact, with a 10 times longer aggregation period, the spike intensity value reported by all measurement configurations (including the sFlow instantaneous sample values) is lower than the case with $C = 1$ s. When designing a real-time monitoring system, the granularity of collected samples, expressed by the C parameter, should be carefully determined to properly detect traffic spikes. Therefore, choosing proper values of α and C allows to balance the accuracy, overhead and sensitivity of the sFlow protocol and the EWMA computation.

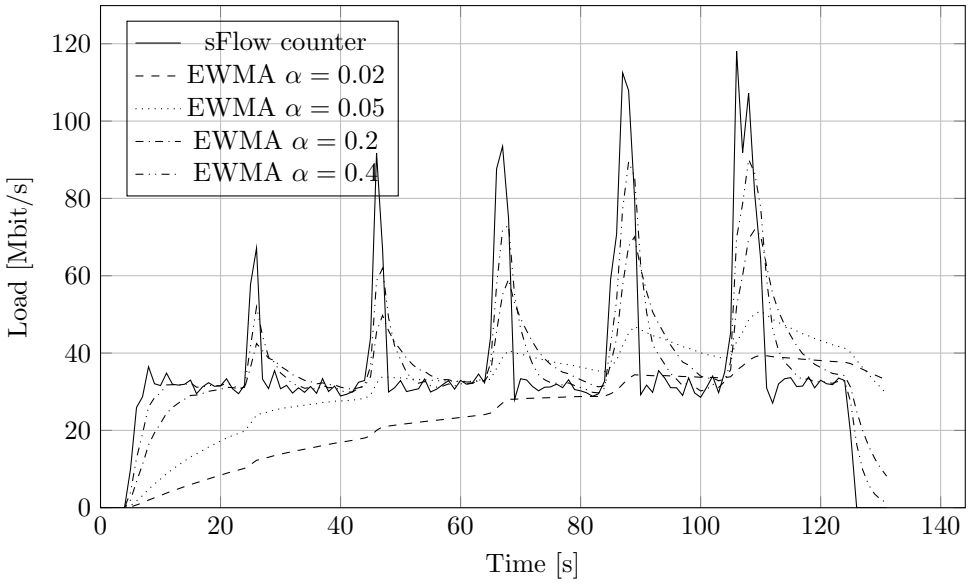


Figure 20: Accuracy of the sFlow EWMA under additional traffic spikes and different moving average values α, with $N = 10$ and $C = 1$ s, compared to the instantaneous sFlow samples.

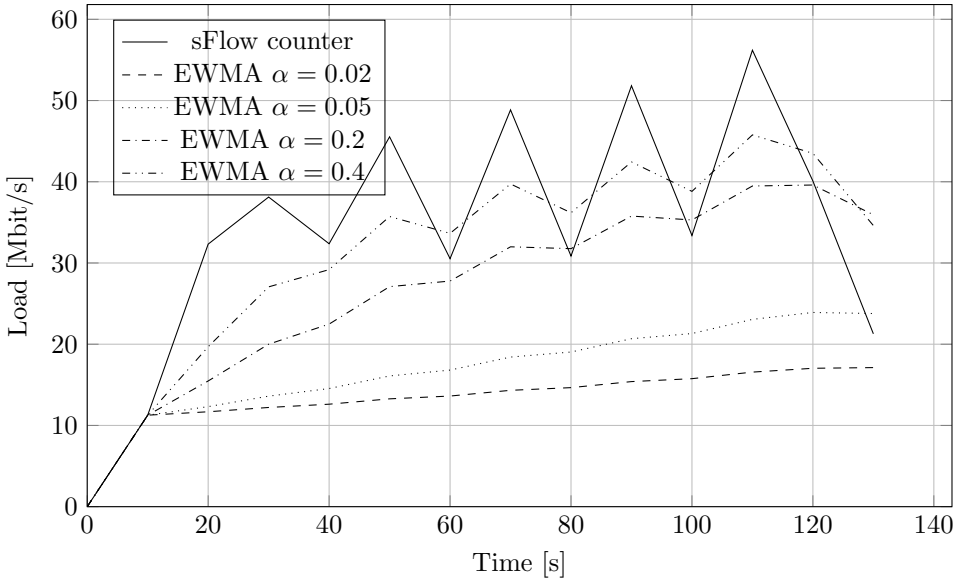


Figure 21: Accuracy of the sFlow EWMA under additional traffic spikes and different moving average values α, with $N = 10$ and $C = 10$ s, compared to the instantaneous sFlow samples.

6.7. Statistical correctness

Each quantitative experiment reported in the previous sections was run 10 times to ensure statistical correctness of the performance evaluation. Thus, all data presented in the figures above were obtained as the mean value calculated on those 10 repetitions. We also calculated the 95% confidence intervals but we decided not to show them in the figures for readability reasons. Instead, separate figures are provided in this section. Namely, the statistical correctness of sFlow measurements is displayed in Figs. 22 and 23. The former assumes $C = 1$ s and $\alpha = 0.3$ while N is variable. The latter assumes $C = 1$ s and $N = 10$ while α is variable. The considered scenarios are the same as those in Figs. 15 and 17, respectively. By selecting different combinations of α and N parameter values, we intend to present the extreme cases. The colored area corresponding to each curve represents the 95% confidence interval. We have validated all the results collected by sFlow in the same way and no anomaly was observed.

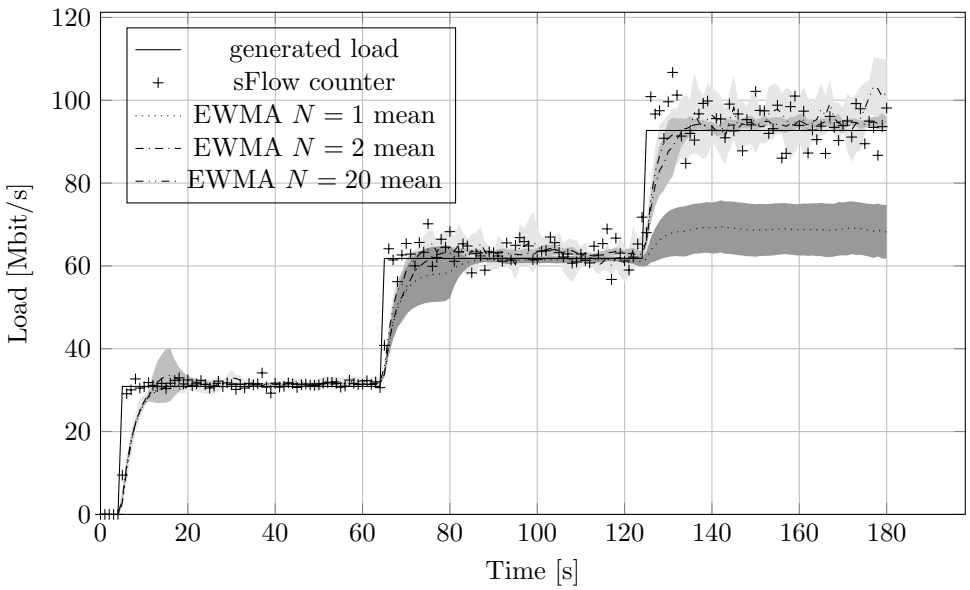


Figure 22: Statistical considerations for the sFlow EWMA under different sampling ratio values N, with $\alpha = 0.3$ and $C = 1$ s, compared to the instantaneous sFlow samples and the generated load pattern.

Similar statistical considerations are also provided for the measurements originating from Ceilometer. Figures 24 and 25 show the CPU utilization

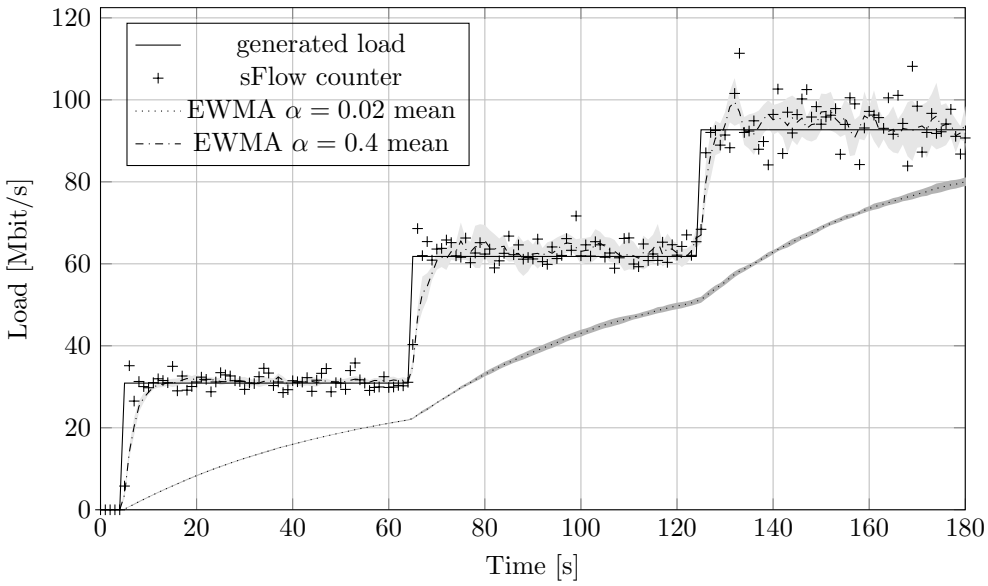


Figure 23: Statistical considerations for the sFlow EWMA under different moving average values α, with $N = 10$ and $C = 1$ s, compared to the instantaneous sFlow samples and the generated load pattern.

and network load metrics, respectively. The considered scenario is the same as the one presented in Fig. 7, with traffic spikes increasing in time. An anomaly in the CPU utilization can be observed between 150 and 225 seconds since the start of the experiment. It originates from the fact that in one of the experiment runs the CPU was additionally loaded by operating system activities not related to the traffic handling.

To sum up, the number of 10 repetitions for each of the quantitative experimental scenarios may seem to be too small. However, thanks to the system stability, the obtained narrow confidence intervals prove that our results are credible and justified conclusions can be drawn based on them.

6.8. Discussion

To sum up, the presented multidimensional evaluation of sFlow protocol for the purpose of efficient SDN/NFV infrastructure monitoring leads to the formulation of some best practises. The very first step is to determine limitations and our studies revealed two possible issues. The first one concerns the maximum acceptable signalling overhead. As we demonstrated in Section 6.6.2, if one can tolerate 15-20% of additional traffic in the control

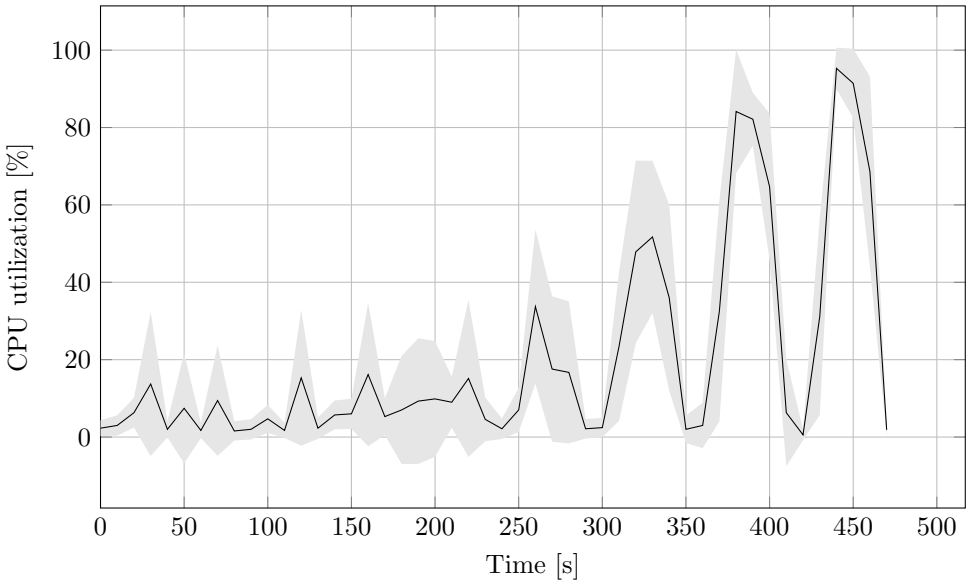


Figure 24: Statistical considerations for the Ceilometer measurements of time the CPU is busy for a selected VNF stressed by traffic spikes with increasing intensity.

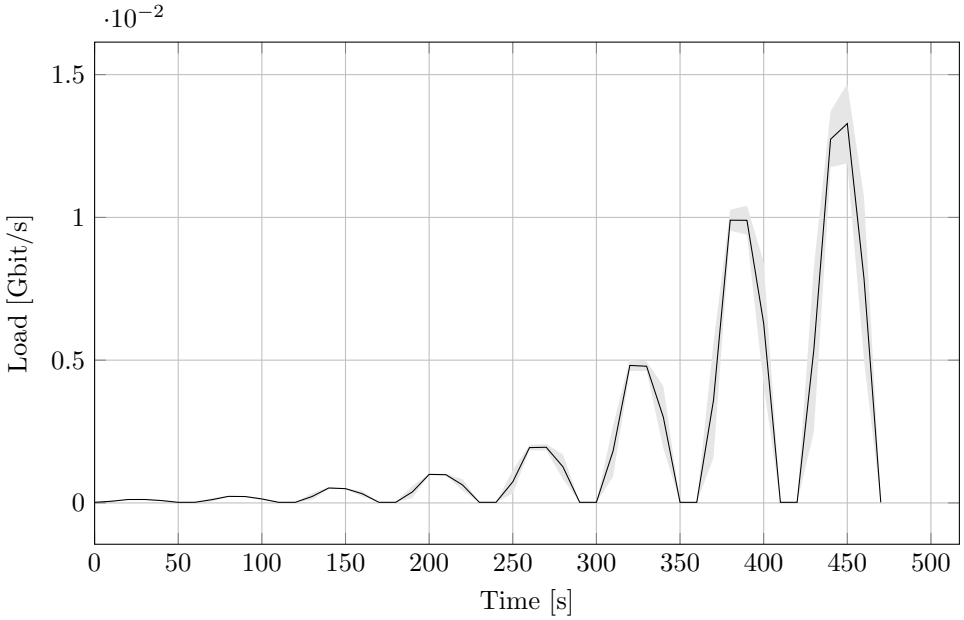


Figure 25: Statistical considerations for the Ceilometer measurements of network load for a selected VNF stressed by traffic spikes with increasing intensity.

channel, then there is no need to increase the sampling ratio above $N = 1$ and the sample aggregation interval above $C = 1$ with the aim to limit the signalling overhead, at the cost of accuracy and sensitivity. The second possible limitation was described in Section 6.6.1 as sFlow saturation, which can be much more invasive as it originates from the architectural constraints of the sFlow protocol. In our PoC implementation, we observed this issue for $N = 1$ regardless of the sample aggregation interval being $C = 1$ or $C = 10$. Those values depend on the expected network load and performance of computing infrastructure hosting the sFlow software. Thus, one should determine minimal values of sampling ratio and sample aggregation interval to avoid sFlow saturation for the highest possible traffic rates expected to be measured. On the other hand, increasing both C and N values increases also a delay after which credible measurement results are available. In case when immediate measurements and detailed analysis of each packet are needed, for instance, due to strict delay or jitter requirements, other network monitoring protocols should be used.

Next, when configuring the sFlow parameters, it must be determined whether the network load to be precisely measured is expected to be approximately constant (typical of static core networks) or highly variable (typical of data center networks and programmable infrastructures with numerous degrees of freedom). For static scenarios both sampling ratio and sample aggregation interval can be easily increased to a certain point for which accuracy and sensitivity are not deteriorated, while the signalling overhead is limited not to overload sFlow infrastructure. In our PoC, $N = 5$ and $C = 10$ can be considered as a good choice. On the other hand, for dynamic network loads one must consider two interdependent trade-offs. The first one concerns the question of how fast the monitoring tool should respond to the changing traffic, which is directly affected by the interval between reported measurements. If real-time measurements are required, for example, to perform online traffic engineering or analyze the delay of each packet, then the maximum reasonable value for the sample aggregation interval is $C = 1$. The sampling ratio may be reasonably assigned having in mind the identified limitations, signaling overhead, and the fact that smaller values of N will also bring stronger fluctuations of collected samples traffic. In our PoC, $N = 5$ should be considered as a choice that brings significant reduction in terms of signaling overhead, reasonably stable measurements, and sufficient response time to the traffic changes. The second trade-off is related to the question if transient network states and traffic spikes should be detected by

the sFlow tool. If so, C and weighting coefficient α can be properly adjusted to detect spikes of expected traffic volume and width while measurements overshoot should be considered. If the measurements are expected to feed traffic-engineering mechanisms, the granularity of the collected metrics is critical to achieve expected sensitivity. One must note that, although the EWMA approach has the same objective as the sample aggregation interval parameter, significant differences occur. The EWMA is computed by the monitoring module instead of being part of the sFlow protocol. As a result, changing the α parameter does not affect the signalling overhead, but it can simplify the dynamic adjustment of sensitiveness to changing network conditions. Furthermore, we designed our module to include the EWMA computation in order to integrate it with any network monitoring protocol or tool, which may not be equipped with configuration options analogous to the ones offered by sFlow. In the case of our PoC, reasonable values that can be applied are $C = 1$ and $\alpha = 0.2$.

The proposed monitoring module was deployed in three different evaluation environments without any modifications. For that purpose, it was easily integrated with an open-source cloud platform, basic container technology, and a public cloud environment. Furthermore, it was successfully integrated with a widely used infrastructure automation tool. All these facts prove that the solution is portable, universal, and thanks to the modular architecture, can be easily combined with a wide variety of external systems. The monitoring capabilities were presented in a testbed based on the private cloud platform, whereas in the testbed based on container technology we showed how the monitoring module can be used to apply traffic steering mechanisms and avoid potential service disruptions. Two different research scenarios show that both NFV and SDN domains can take advantage of the unified measurement solution. Finally, the real performance of the public cloud network may be discovered using the proposed monitoring module.

7. Conclusion

In this paper we proposed and verified a unified and standalone monitoring module designed to monitor combined SDN/NFV infrastructures in a cloud environment. The module communicates with the control plane elements through well-known and universal interfaces. Thus, the module can be easily integrated with a variety of controllers as well as tools aimed at collecting metrics specific to particular assets. The proposed approach does

not impose a significant load on existing control plane components, due to the full independence of it, and the possibility of deploying the module in the form of a VNF.

The proposed solution has been deployed and validated in real-life testbeds and public cloud environment under numerous scenarios addressing various aspects and potential impediments regarding accuracy, granularity, sensitivity, overhead, latency, versatility, and portability. With these prototype implementations, it was possible to prove the feasibility and effectiveness of our solution for various SDN/NFV infrastructures. Although we analyzed in detail the specific case of sFlow as a network monitoring protocol, our findings can be generalized to any other specific technical solution. The whole set of experiments confirms that the modular architecture provides significant advantages in terms of new features and degrees of freedom with respect to existing solutions integrated in the control plane. For example, some limitations of the selected tools can be overcome by proper configuration (e.g., sFlow adjustments to the expected load and particular needs), while others will require replacing those components (e.g. Ceilometer module for real-time monitoring), which is easily enabled by the loosely coupled architecture. Furthermore, deployment in a public cloud environment proves the versatility and portability of our solution, as well as compatibility with infrastructure automation tools. Additional advantages and usage scenarios of the proposed module were revealed in a virtualized cloud infrastructure managed by third-party providers.

As proved with the experiments performed in a container-based environment, we believe that the proposed solution may provide measurement data to traffic steering mechanisms aimed at avoiding congestion and improving service provisioning in *softwareized* ICT infrastructures. Therefore, in the nearest future, we plan to deploy selected virtual network functions, e.g., Deep Packet Inspection, WAN Accelerator, or Traffic Conditioner, and use the proposed monitoring module to feed optimization algorithms aimed at improving infrastructure utilization and avoiding congestion. Another possible research avenue is to develop mechanisms for adaptive adjustments of monitoring parameters. Finally, we also consider extending the monitoring module with separate components orchestrating active measurements in SDN/NFV infrastructures.

Acknowledgment

The contribution of Piotr Borylo was funded by the Dean of AGH Faculty of Computer Science, Electronics and Telecommunications under Grant No. 15.11.230.384. This work was partially funded by the University of Bologna under the AlmaIdea project "PRONE (Programmable Networks for Emergency Trials)."

References

[1] J. Soares, C. Gonalves, B. Parreira, P. Tavares, J. Carapinha, J. P. Barraca, R. L. Aguiar, S. Sargento, Toward a Telco Cloud Environment for Service Functions, IEEE Communications Magazine 53 (2) (2015) 98–106.

[2] F. Hu, Q. Hao, K. Bao, A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation, IEEE Communications Surveys Tutorials 16 (4) (2014) 2181–2206.

[3] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, R. Boutaba, Network Function Virtualization: State-of-the-Art and Research Challenges, IEEE Communications Surveys Tutorials 18 (1) (2016) 236–262.

[4] I. Foster, Y. Zhao, I. Raicu, S. Lu, Cloud Computing and Grid Computing 360-Degree Compared, in: 2008 Grid Computing Environments Workshop, 2008, pp. 1–10.

[5] A. Fahmin, Y.-C. Lai, M. S. Hossain, Y.-D. Lin, Performance Modeling and Comparison of NFV Integrated With SDN: Under or Aside?, Journal of Network and Computer Applications 113 (2018) 119 – 129.

[6] F. Callegati, W. Cerroni, C. Contoli, R. Cardone, M. Nocentini, A. Manzalini, SDN for Dynamic NFV Deployment, IEEE Communications Magazine 54 (10) (2016) 89–95.

[7] M. S. Bonfim, K. L. Dias, S. F. L. Fernandes, Integrated NFV/SDN Architectures: A Systematic Literature Review, ACM Comput. Surv. 51 (6) (2019) 114:1–114:39.

[8] M.-T. Thai, Y.-D. Lin, P.-C. Lin, Y.-C. Lai, Towards Load-Balanced Service Chaining by Hash-based Traffic Steering on Softswitches, Journal of Network and Computer Applications 109 (2018) 1 – 10. URL http://www.sciencedirect.com/science/article/pii/S1084804518300699

[9] D. Zhou, Z. Yan, Y. Fu, Z. Yao, A Survey on Network Data Collection, Journal of Network and Computer Applications 116 (2018) 9 – 23.

[10] D. Kreutz, F. M. V. Ramos, P. E. Verssimo, C. E. Rothenberg, S. Azodolmolky, S. Uhlig, Software-Defined Networking: A Comprehensive Survey, Proceedings of the IEEE 103 (1) (2015) 14–76.

[11] P. Tsai, C. Tsai, C. Hsu, C. Yang, Network Monitoring in Software-Defined Networking: A Review, IEEE Systems Journal 12 (4) (2018) 3958–3969.

[12] B. Yi, X. Wang, K. Li, S. k. Das, M. Huang, A Comprehensive Survey of Network Function Virtualization, Computer Networks 133 (2018) 212 – 262.

[13] G. Aceto, A. Botta, W. de Donato, A. Pescap, Cloud Monitoring: A Survey, Computer Networks 57 (9) (2013) 2093 – 2115.

[14] J. Zhang, Z. Wang, N. Ma, T. Huang, Y. Liu, Enabling Efficient Service Function Chaining by Integrating NFV and SDN: Architecture, Challenges and Opportunities, IEEE Network 32 (6) (2018) 152–159.

[15] G. Gardikis, I. Koutras, G. Mavroudis, S. Costicoglou, G. Xilouris, C. Sakkas, A. Kourtis, An Integrating Framework for Efficient NFV Monitoring, in: 2016 IEEE NetSoft Conference and Workshops (NetSoft), 2016, pp. 1–5.

[16] G. Gardikis, I. Koutras, G. Mavroudis, S. Costicoglou, G. Dimosthenous, D. Christofi, M. D. Girolamo, K. Karras, G. Xilouris, E. Trouva, M. Arnaboldi, P. Harsh, E. Markakis, T-Nova Project Deliverable D4.42: Monitoring and Maintenance, http://www.t-nova.eu/wp-content/uploads/2016/05/TNOVA_D4.42_Monitoring_and_Maintenance_Final_v.1.0.pdf, [Online; accessed 19-July-2018] (March 2016).

[17] C. K. Dominicini, G. L. Vassoler, L. F. Meneses, R. S. Villaca, M. R. N. Ribeiro, M. Martinello, VirtPhy: Fully Programmable NFV Orchestration Architecture for Edge Data Centers, IEEE Transactions on Network and Service Management 14 (4) (2017) 817–830.

[18] H. T. Chang, S. Y. Wang, Using SDN Technology to Mitigate Congestion in the OpenStack Data Center Network, in: 2015 IEEE International Conference on Communications (ICC), 2015, pp. 401–406.

[19] M. Gharbaoui, B. Martini, D. Adami, S. Giordano, P. Castoldi, Cloud and Network Orchestration in SDN Data Centers: Design Principles and Performance Evaluation, Computer Networks 108 (2016) 279 – 295.

[20] G. Liu, M. Trotter, Y. Ren, T. Wood, NetAlytics: Cloud-Scale Application Performance Monitoring with SDN and NFV, in: Proceedings of the 17th International Middleware Conference, Middleware '16, 2016, pp. 8:1–8:14.

[21] Y. Li, R. Miao, C. Kim, M. Yu, FlowRadar: A Better NetFlow for Data Centers, in: Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation, NSDI'16, USENIX Association, Berkeley, CA, USA, 2016, pp. 311–324.

[22] M. A. Kourtis, G. Xilouris, G. Gardikis, I. Koutras, Statistical-Based Anomaly Detection for NFV Services, in: 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2016, pp. 161–166.

[23] M. A. Kourtis, M. J. McGrath, G. Gardikis, G. Xilouris, V. Riccobene, P. Papadimitriou, E. Trouva, F. Liberati, M. Trubian, J. Batall, H. Koumaras, D. Dietrich, A. Ramos, J. F. Riera, J. Bonnet, A. Pietrabissa, A. Ceselli, A. Petrini, T-NOVA: An Open-Source MANO Stack for NFV Infrastructures, IEEE Transactions on Network and Service Management 14 (3) (2017) 586–602.

[24] M. Shirazipour, H. Mahkonen, M. Xia, R. Manghirmalani, A. Takacs, V. S. Vega, A Monitoring Framework at Layer4/7 Granularity Using Network Service Headers, in: 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), 2015, pp. 54–60.

[25] J. Halpern, C. Pignataro, Service Function Chaining (SFC) Architecture (Oct. 2015).

[26] F. Callegati, W. Cerroni, C. Contoli, Virtual Networking Performance in OpenStack Platform for Network Function Virtualization, Journal of Electrical and Computer Engineering 2016.

[27] B. Han, V. Gopalakrishnan, L. Ji, S. Lee, Network Function Virtualization: Challenges and Opportunities for Innovations, IEEE Communications Magazine 53 (2) (2015) 90–97.

[28] P. Borylo, A. Lason, J. Rzasa, A. Szymanski, A. Jajszczyk, Green Cloud Provisioning Throughout Cooperation of a WDM Wide Area Network and a Hybrid Power IT Infrastructure, Journal of Grid Computing 14 (1) (2016) 127–151.

[29] P. Borylo, et al., Fitting Green Anycast Strategies to Cloud Services in WDM Hybrid Power Networks, in: Proc. IEEE GLOBECOM, Austin, TX, USA, 2014, pp. 2633–2639.

[30] P. Borylo, A. Lason, J. Rzasa, A. Szymanski, A. Jajszczyk, Energy-Aware Fog and Cloud Interplay Supported by Wide Area Software Defined Networking, in: Proc. IEEE ICC, Kuala Lumpur, Malaysia, 2016.

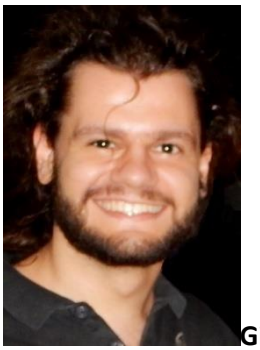
[31] F. Foresta, W. Cerroni, L. Foschini, G. Davoli, C. Contoli, A. Corradi, F. Callegati, Improving OpenStack networking: Advantages and performance of native SDN integration, in: 2018 IEEE International Conference on Communications (ICC), 2018, pp. 1–6.

[32] Does iperf tell white lies?, `https://iwl.com/idocs/does-iperf-tell-white-lies`, [Online; accessed 13-May-2019].

[33] A. Gupta, M. F. Habib, U. Mandal, P. Chowdhury, M. Tornatore, B. Mukherjee, On Service-Chaining Strategies Using Virtual Network Functions in Operator Networks, Computer Networks 133 (2018) 1 – 16.

*Author Biography

Piotr Boryło is working as assistant professor at the Department of Telecommunications, AGH University of Science and Technology. He received M.S. and Ph.D. degrees in Telecommunications from the same university in 2012 and 2016, respectively. He is interested mainly in traffic engineering and resource provisioning issues in Software Defined Networks, but also for the purpose of cloud services provisioning, energy efficiency, 5G networks, data centers, fog computing concept. Both static and dynamic optimization are within the scope of his interests, as well as research techniques like simulations, emulations and physical testbeds.



Gianluca Davoli received his M.Sc. Degree in Telecommunications Engineering from the University of Bologna in 2017, and he has been a PhD student and Research Fellow at the same University since then. His fields of interest revolve around communication networks, focusing on the new approaches to management and monitoring of network resources made available by the ongoing process of network softwarization.

Artur Lason is an assistant professor at AGH University of Science and Technology, Poland. He received M.Sc. and Ph.D. degrees from AGH University in 1992 and 1999 respectively. Artur Lason is the co-author of some books (in Polish) and numerous technical papers in the field of optical networks, green networking, edge and cloud networking, SDN as well as security issues in a cloud environment. He was involved in several international research projects related to the indicated research areas. Currently his main research interests are focused on optimised and secure virtualisation of network functions.

Michał Rzepka is a PhD student at the Department of Telecommunications, AGH University of Science and Technology, in Krakow, Poland. He received the MSc degree in ICT from the same university in 2017. His area of research covers Software Defined Networking, traffic engineering and network measurements. His interests also include cloud computing and mobile technologies.

Walter Cerroni is an assistant professor of communication networks at the University of Bologna, Italy. His recent research interests include software-defined networking, network function virtualization, service function chaining in cloud computing platforms, intent-based northbound interfaces for multi-domain/multi-technology virtualized infrastructure management, modeling and design of inter-data and intra-data center networks. He has co-authored more than 120 articles published in well renowned international journals, magazines, and conference proceedings. He serves/served as Series Editor for IEEE Communications Magazine, Associate Editor for IEEE Communications Letters, and Technical Program Co-Chair for IEEE-sponsored international workshops and conferences.

*conflict of Interest Statement

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Credit Author Statement

Piotr Borylo: Conceptualization, Methodology, Validation, Formal analysis, Resources, Writing - Original Draft, Writing - Review & Editing, Visualization, Supervision, Project administration, Funding acquisition

Gianluca Davoli: Methodology, Software, Validation, Investigation, Data Curation, Writing - Original Draft, Visualization

Michal Rzepka: Software, Validation, Investigation, Writing - Review & Editing

Artur Lason: Validation, Writing - Review & Editing

Walter Cerroni: Conceptualization, Methodology, Validation, Formal analysis, Writing - Review & Editing, Supervision