SFCPerf: An Automatic Performance Evaluation Framework for Service Function Chaining

Igor Jochem Sanz, Diogo Menezes Ferrazani Mattos, and Otto Carlos Muniz Bandeira Duarte
Universidade Federal do Rio de Janeiro – GTA/COPPE/UFRJ – Brazil

Emails: {sanz,menezes,otto}@gta.ufrj.br

***Abstract*—Network Function Virtualization allows the provisioning and composition of on-demand network function chains tailored to an application or a service. Repeatable compliance tests and performance comparison of network functions and the whole function chains are required for virtual network function manufacturers and telecommunication operators. In this paper, we propose and develop SFCPerf, a framework for an automatic performance evaluation of service function chaining. SFCPerf describes all experimental configuration as a single workflow and provides automatic: (i) environment creation and setup; (ii) network configuration; (iii) experimental data measurement; (iv) experiment execution and control; and (v) data preliminary analysis. Our framework provides repeatability for experimenting different network functions and virtualization infrastructures. To demonstrate SFCPerf functionality, we design and implement a prototype of a service function chain that complies with the Network Service Header (NSH). We show the results of an SFCPerf experiment that evaluates the performance of our prototype, composed of an intrusion detection system (IDS) and a firewall, running on top of the open platform for network function virtualization (OPNFV).**

I. Introduction

Network function virtualization (NFV) is gaining notable prominence in telecommunications as it reduces hardware expenditures and network operational complexity. By deploying network services as software, NFV improves network flexibility combined with optimized resource allocation [1]. In this model, network functions, such as network address translation (NAT), firewall, intrusion detection systems, and load balancers, migrate from dedicated hardware middle-boxes to software functions on virtual machines. Therefore, network services can be extended and controlled in a centralized manner, dynamically migrated and deployed in the network, and tailored for each application. Concerning security features, the NFV flexibility allows to deploy new policies, to promote fast updates, to define security zones, to steer traffic and to isolate compromised network components [1]–[4]. Furthermore, service function chaining (SFC) is a key enabler to flexible traffic management of a service or application [5]. High latency or incorrect ordering of virtual network functions imply failure of packet handling policies, increase of vulnerabilities or occurrence of security incidents. Therefore, performing repeatable and comparable experimentations through a infrastructure-agnostic framework is essential to identify and avoid performance bottlenecks on NFV and SFC platforms, as well as to correctly define resources constraints [6].

978-1-5386-3416-5/18/$31.00 © 2018 IEEE

In this paper, we propose SFCPerf, a framework for automating experimentation of service function chaining. The main goal of the framework is to provide repeatability to experimentation through the definition of a testing workflow. Thus, results obtained by the framework enable comparison to any other service function chain configuration, as the scenario and the probes are strictly defined by a workflow description file. SFCPerf automates environment creation and network configuration during the setup phase. Then, during the experimental phase, SFCPerf configures data measurement, performs data collection and controls the experiment. In the post-experiment phase, acquired data is pre-processed and sent to preliminary analysis. We develop and evaluate the performance of a security function chaining prototype, based on service function chaining, to demonstrate the functionality of our framework. The prototype is composed of two security network functions: an intrusion detection system based on machine learning techniques and stream processing; and an adjustable firewall with a RESTful interface. We build our prototype on top of the European Telecommunications Standards Institute (ETSI) NFV MANO architecture [7], and we analyze network function chaining in compliance with RFC 7665 [8], provided by the Internet Engineering Task Force (IETF). The prototype meets the specifications of Network Service Header (NSH) for the SFC encapsulation. Furthermore, we use SFCPerf to evaluate the performance of VNFs chaining over different topologies and the current development level of NSH to identify major bottlenecks. We adopt the Open Platform for Network Function Virtualization (OPNFV) as an NFV infrastructure for our evaluation experiments. Results show that chaining multiple functions incurs a throughput decrease and a linear end-to-end delay increase, which are independent of the deployed topology.

The remainder of this paper is organized as follows. Section II presents related work. Section III addresses the state-of-the-art of network function virtualization and service function chaining. Section IV exposes the SFCPerf framework and Section V describes implementation details. Section VI describes the prototype and the virtual security functions. Section VII discusses the performance evaluation. Section VIII concludes the paper.

II. Related Work

Different network function virtualization architectures have been proposed with their own service function chaining approaches. Likewise, middle-box chaining can be performed by using software-defined networking techniques. FlowTags is an SFC proposal for middle-boxes that is capable of tagging headers and passing context information to the subsequent middle-box while enforcing traffic policies [9]. The StEERING proposal uses multiple tables, presented in the OpenFlow 1.0 standard, instead of adding tags [10]. It creates hierarchical forwarding rules while adding metadata to the packet handling on each step and defining the next hop in the chain.

Most proposals consider an infrastructure that combines software-defined networking with network function virtualization [11], [12]. The ESCAPE tool is built upon network function virtualization standardized by ETSI [11]. The key idea is to use ClickOS as the basis for the development of virtual network function prototypes [13]. On the other hand, Cloud4NFV presents an architecture for network function virtualization based on four planes: infrastructure, virtual infrastructure management, orchestration, and service [12]. Though ESCAPE and Cloud4NFV are closely related to ETSI architecture, they do not comply with the IETF service function chaining [8].

The NetBricks proposal develops network-function packet forwarding with zero copy [14]. Hence, the chaining of network functions uses shared memory with reference passing in the memory area. This proposal brings considerable gains in performance of virtual functions in terms of bandwidth and latency. Performance of virtual functions chaining is also evaluated analytically and is considered as a constraint for optimization problems, such as the placement of VNFs over the physical infrastructure. Lopez *et al.*, in turn, argue that the location of virtual functions on the physical infrastructure follows a trade-off between accepting a larger number of VNFs and the delay of chaining on more distant physical nodes [15].

Emmerich *et al.* evaluate the performance of virtual switches during VNF chaining. They conclude that optimizations in core operating system configurations and dedicated CPU utilization for the network interfaces are essential to increase the performance of virtual switches [16]. Callegati *et al.* present a comparison of the performance of network virtualization and the main components of the OpenStack cloud operating system [17]. Bonafiglia *et al.* compare the performance of different network function virtualization technologies [18]. They consider configurations of virtual switches with and without data plane development kit (DPDK). The authors also compare the performance of network chaining performed in virtual machines versus containers. The results show the performance achieved by virtual machines is superior to the performance of containers when using switches with DPDK support and dedicated processing cores.

Although most works focus on evaluating performance of NFV and SFC on a given scenario, there is a need for solutions to evaluate performance of NFV use cases, regarding the interoperability problem of the early stage of NFV [19]. In this sense, DETER is a testbed proposal for network testing and monitoring that focuses on security experiments [20]. Similarly, RIO is an experimentation platform to emulate denial of service (DoS) attacks using NFV technology. The objective is to investigate DoS attack patterns and potential mitigation mechanisms. RIO also automates the configuration and setup of network elements and proposes a language to describe a given test scenario. Riggio *et al.* propose Scylla, a descriptive language to describe virtual network functions orchestration regardless the underlying infrastructure [21].

Unlike all aforementioned works, in this paper we propose SFCPerf, a framework for automating the experimentation of performance evaluation of service function chaining. The framework aims to be used to compare performance of service function chains composed of virtual network functions from different manufacturers and running on distinct NFV-SFC platforms.

III. Virtual Network Function Chaining

A key aspect of service function chaining (SFC), also referred as network function chaining (NFC), is to extend the functionality of the network service by chaining specific functions into the network layer. Virtual network functions (VNF) are virtual machines that perform functions on the network layer to replace the numerous hardware-specific middleboxes that currently exist, such as intrusion detection system, firewall, proxy etc.

The basic architecture of service function chaining, according to RFC 7665 [8], is shown in Figure 1. Considering a multi-service environment, a classifier is configured with specific rules to identify and specify the chain of VNFs that each service flow must traverse. The chain is defined by a logic sequence of VNFs called service function path (SFP). Hence, flows from different micro-services can simultaneously traverse the same VNF, but each coursing its own service function path. Virtual network functions may also be hosted on different physical nodes. Therefore, the service function forwarder (SFF) is a mandatory element in each network function virtualization infrastructure (NFVI) node to provide the virtual links to its guest VNFs.

In the SFC architecture, isolation between micro-service flows in the same VNF is performed through packet encapsulation. VNFs may be aware or unaware of the SFC encapsulation. VNFs that are unaware of encapsulation require a precedent element to decapsulate packets, called SFC Proxy, whereas SFC-aware VNFs need either a kernel module or a software switch compatible with the SFC encapsulation. The most consolidated proposal of encapsulation that meets the specification of SFC architecture is the network service header (NSH) [22]. NSH performs correct packet forwarding to the next VNF and isolation between micro-services flows thanks to two main fields, the service index (SI) and the service path identifier (SPI). The SI is an 8-bit index representing the relative position of the current VNF in the chain. The SPI is a 24-bit identifier associated to a specific service function

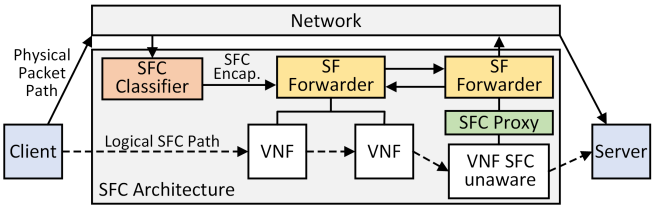


Fig. 1. Basic elements of the network functions chaining architecture. The SFC classifier encapsulates the incoming traffic and the SF forwarder redirects the traffic to the correct chain based on the encapsulation headers. An SFC proxy enables the implementation of functions unaware of the SFC encapsulation.

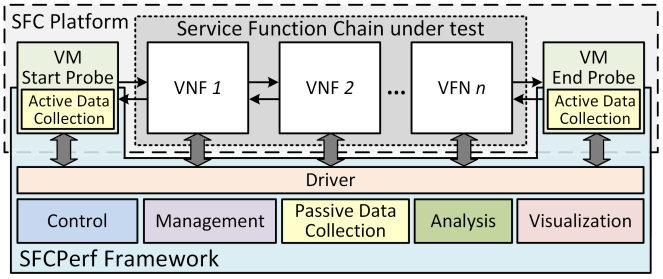


Fig. 2. The proposed SFCPerf framework. The driver module provides an adaptive interface between the framework and different SFC platforms. The control module coordinates other modules to provide the environment setup, data collection and data delivery to user.

path described in a software switch. All forwarding decisions on packets tagged with NSH header are defined by rules in the software switch and pro-actively orchestrated by the SDN controller, without resulting in delay for the packet forwarding.

To demonstrate the functionalities of our proposed SFCPerf framework, we develop and evaluate a service function chain prototype, considering different scenarios. We use the open platform for network function virtualization (OPNFV) with an SDN and NFV hybrid architecture to implement service function chaining. The chaining is built upon rules in the software switch Open vSwitch, using the OpenFlow protocol. These rules are managed by the SDN OpenDaylight controller coupled with a VNF manager and orchestrator, named Tacker.

IV. The Proposed SFCPerf Testing Framework

Different open source NFV platforms are currently being proposed, such as OPNFV [23], OpenMANO [24], ClickOS [13]. There are also other approaches for service function chaining, such as container-based chaining [25], NetBricks [14], Dysco [26], etc. We modularly design the SFCPerf framework to allow automated SFC testing, agnostic to the underlying NFV-SFC infrastructure. Our testing framework provides experiment repeatability for all scenarios, which is essential to compare different approaches for VNF. When testing different SFC infrastructure providers, we are able to perform exactly the same experiment over the different infrastructures and, thus, we assure repeatability required to compare the performance. The SFCPerf framework is illustrated in Figure 2 and comprises the following main components: control, management, driver, passive and active data collection, analysis, and visualization modules.

Control module is the main component of the architecture. It configures and coordinates the other SFCPerf modules. The central goals of the control module are to handle all configuration parameters setup and calibration knobs, to check the experiment health, and to deliver the testing results to user. The control module receives a unique document as input data containing a detailed JSON description of the experiment. The document contains all parameters necessary to perform an automated testing of a service function chain. The control module also assigns the correct driver regarding compatibility into different NFV infrastructures.

Management module governs the NFV-SFC platform and is responsible for orchestrating operations, such as instantiating and resizing VMs or VNFs. These operations are instructions to the management module of the NFV-SFC platform intermediated by the driver module. The management module also manages sensing tools in the NFV-SFC platform by coordinating the passive data collection module. The main difference between management and control modules is that the former handles NFV-SFC related operations, while the latter controls the experiment workflow.

Passive data collection module senses physical and virtual resource usage when performing experimental evaluations of VNFs. The data include information from telemetry modules of the NFV platform, such as disk, processing and memory usage of virtual resources, and from a set of data gathering applications executed on top of VNFs, e.g., tpcdump, top, iotop, free, etc.

Active data collection module consists in a set of probe applications, such as iperf, ping and httperf, that actively measures the network performance. The applications are hosted in the endpoint VMs and generate traffic that traverses the chain of VNFs. Thus, traffic is processed in the other termination to obtain the evaluation metric.

Analysis module performs operations over experimental data, such as data preprocessing, merging, correlating, and enriching. The module allows to combine multiple sources of data in a customized manner and also to enrich data with additional information, such as geo-tagging.

Visualization module provides a user-readable interface for data visualization. The module plots graphs and compares experimentation approaches. We conceive the visualization module with Kibana[1], a third-party data visualization application which is a component of the Elastic stack. Kibana implements a web interface that allows the visualization of data and the design of new observation scenarios in a simple and fast way. Moreover, Kibana benefits from a high-performance execution of queries over large volumes of data to provide visualization with low-latency.

[1]Available at https://www.elastic.co/products/kibana.

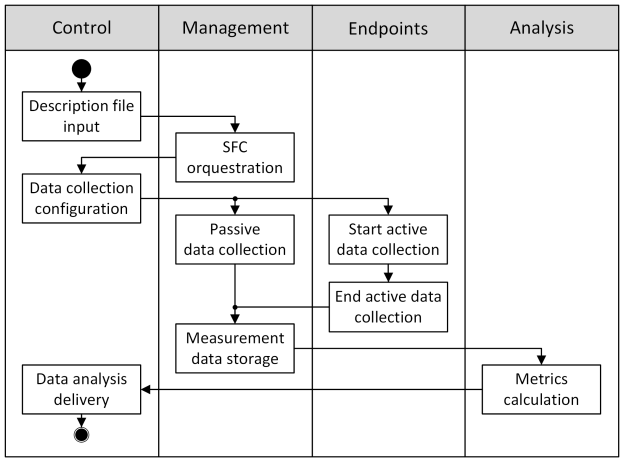


Fig. 3. UML activity diagram of the proposed SFCPerf framework. The framework user interacts with the control module, which sets up the environment, controls the experimentation, and delivers the obtained data.

Driver module is an essential component to provide compatibility of the framework to any NFV-SFC infrastructure. It provides an RPC interface between framework modules and orchestration components of the NFVI. In our prototype, we use OPNFV as NFVI platform, thus, the driver sends requests directly to Openstack, OpenDaylight, and Tacker APIs. For OPNFV, we implement this communication via RESTful HTTP requests.

The execution of an experiment over SFCPerf is exemplified on the activity diagram, Figure 3. We highlight the relationship among modules and their functions. First, the framework user loads the workflow description file into the control module, which parses and interprets the test description and calls the appropriate functions on the management module. All modules deal with the SFC platform through the driver module, omitted from Figure 3 for the sake of simplicity. The control module also instantiates all data collection modules. The SFC platform hosts tools that monitor the resource usage and feed the passive data collection module. Moreover, active data collection takes place between source and destination of the SFC. Depending on the tested SFC platform, active data collection is implemented as active measuring tools instantiated in virtual machines, or on physical machines that behave as SFC endpoints. After the accomplishment of measures, the data collection modules report the gathered data to the management module. The management module also keeps a local data repository, which is implemented as a non-structured database (NoSQL) based on the open source Elasticsearch [2]. The analysis module computes the final result and reports the final metrics to the control module. Finally, control module exports experimental results to the framework user and publishes it on the visualization module.

We define a JSON-based description language to represent the detailed configuration of an environment and the workflow

[2]Available at https://www.elastic.co/products/elasticsearch.

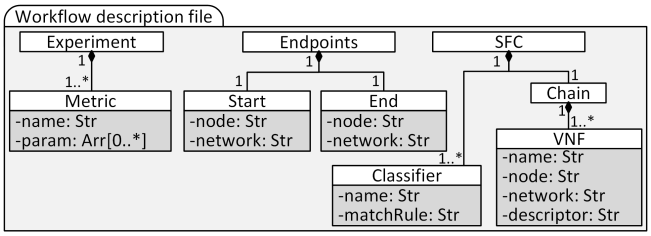


Fig. 4. UML class diagram of the JSON structure of the workflow description file. The description file contains all parameters to set up the environment, configure parameters knobs and obtain the desired metrics.

of an arbitrary experiment. The workflow description file is shown in Figure 4 as a UML class diagram. The diagram represents the JSON structure of the parameters defined in the workflow description file. The parameters include information about endpoints, resource allocation, resource placement, VNF descriptors, VNF types, VNF chaining order, and the measurements to evaluate the SFC. The VNF descriptor parameter is the path to a VNF description file based on TOSCA standards and YAML language. Moreover, the parameters list is adaptable to comprise adjustable knobs of different VNFs.

V. The SFCPerf Framework Implementation

The SFCPerf framework is written in Python language and its source code is available at https://github.com/ijochem/SFCPerf. We conceive the framework as a modular object-oriented software project in which new features are achieved by adding new modules to the framework. Thus, the SFCPerf is built upon some basic parent classes that provides the API (Application Programming Interface) for any new module. The driver module, for instance, assures the infrastructure-agnostic property. We conceive this module as a translation module from the experiments provided by SFCPerf and the NFV platform. For each different platform, we develop a new driver module as an API between SFCPerf and the CLI (command line interface) of the new platform. The new driver module inherits the main functions from the parent driver module. In our prototype, we develop a driver for the Tacker VNF manager used in OPNFV. Moreover, automating the experimentation is achieved by implementing a workflow control mechanism, which follows the experimentation definition on a JSON file. The experiments are also implemented as separated independent modules which are imported and executed in run-time according to the parameters and the sequence in the JSON file. The automation also assures the repeatability property of the conducted experiments, since the JSON input file fully describes the topology and conditions necessary to orchestrate the SFC and obtain the desired metrics. Furthermore, performance comparison between different tests and scenarios are obtained through the execution of a new round of experiments while inputing the same JSON description file with slightly changes on the test or scenario configuration. It assures equality of the experiment

conditions as the workflow of the SFCPerf configures the new experimentation scenario and keeps the same experiment parameters. Thus, comparing the results of different SFC platform resides into running the same JSON file calling the correspondent driver module of the SFC platform. Repeatability property is achieved by a new execution of a experiment-definition JSON file. Besides, experiment reproducibility is achieved by providing a copy of the experiment-definition JSON file with all modules used during the experimentation. It is worth noting that visualization of the results is also handled by an other module that is imported and executed in run-time. The default visualization module of the SFCPerf exports the experiment results to the ElasticSearch database and enables visualization through Kibana.

VI. The Tested Service Function Chaining Prototype

We develop a service function chaining prototype and evaluate it using SFCPerf framework. The prototype is a chain composed of two security functions, an intrusion detection system and a firewall. The former analyzes packets, generates alarms, and defines packet filtering rules, while the latter implements the rules, creating security perimeters on the network. Intelligent chaining of these virtual functions provides the flexibility to instantiate complex real-time security mechanisms at any point of the network. The key idea of the prototype is to associate the processing scalability provided by a streaming processing cloud, realized by Spark[3], with the flexibility of security functions chaining, in order to enable a real-time response to malicious traffic. Figure 5 shows the architecture of our service function chaining prototype.

The main element of the architecture is the intrusion detection system that employs stream processing techniques to perform real-time traffic analysis. The packets are captured through traffic mirroring by the IDS module, which acts directly in the chain. These packets are abstracted into flows, which are defined as a sequence of packets with the same source IP, destination IP, source port, destination port, and transport protocol, during a time window. In total, 46 flow characteristics are extracted and published in a publish/subscribe message service of Apache Kafka [27]. This service operates as a filtering and data flow manipulation system at low latency, in which flow statistics are queued and, then, consumed by the classification module. The classification module, in turn, is instantiated in a dedicated cloud for classification and contains Apache Spark as its main processing core. Spark is implemented in a cluster of virtual machines on a master/slave model, where slaves have a capacity for expansion and reduction of resources.

The implemented IDS uses the lambda architecture [28], composed of three layers: a data processing layer, a batch processing layer, and a service layer. The data processing layer handles real-time flow data. A batch processing layer analyzes a large amount of data stored through distributed computing techniques, such as map-reduce. Finally, a service layer aggregates information obtained from the two previous layers to create outputs of the analyzed data. The IDS classifies malicious or benign flows through machine learning algorithms based on a decision tree classification algorithm.

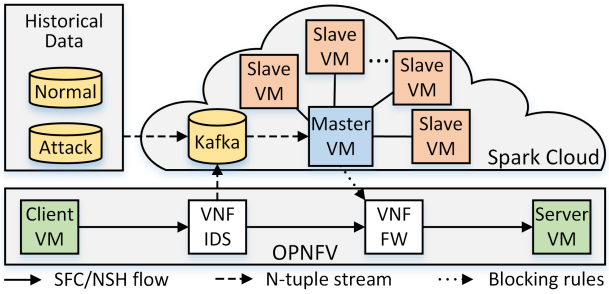


Fig. 5. The architecture of a service function chain use case, composed of a virtual intrusion detection system and a firewall, evaluated with the proposed SFCPerf framework. The intrusion detection system VNF uses a distributed stream processing cloud to analyze data traffic and set blocking rules in real time on the subsequent firewall VNF.

A historical traffic measurement database with entries labeled as attack or as normal is used to train the machine learning algorithm in an off-line manner [27]. The parameters calculated during the off-line historical data processing adjust the classification model in real-time. Thus, the system is adaptive, as parameters can be updated and adjusted to new using behaviors. After training, the IDS allows real-time traffic classification and, by detecting malicious flows, is able to send blocking rules to the firewall.

Our VNF firewall implementation is composed of a module capable of encapsulating and decapsulating NSH packet, built upon the open-source Python application *vxlan_tool*[4]. This application is extended to support a JSON object that stores blocking rules based on the 5-tuple of a packet. The 5-tuple stores the source and destination IP addresses, the source and destination ports and the transport protocol type. Before forwarding the packet to the subsequent VNF in the chain, the 5-tuple is checked over the existing set of rules. A RESTful interface is implemented to enable rule insertions and deletions in the set.

At last, we develop an SFC proxy, shown in Figure 6. The main goal of the proxy is to enable the deployment of VNFs unaware of the SFC encapsulation. Hence, we hide the NSH encapsulation from the VNF application by instantiating a pair of user-space interfaces. The VNF application accesses the user-space interfaces like a common network interface. It is worth noting that OPNFV implementation of NSH also applies a VXLAN encapsulation to forward packets between the service function forwarder and the VNF. Our proxy enhances the VXLAN packet handling by using a software switch in kernel space, which only forwards NSH-encapsulated packets

[3]Available at https://spark.apache.org/.

[4]Available at https://github.com/opendaylight/sfc/blob/master/sfc-test/nsh-tools/vxlan_tool.py.

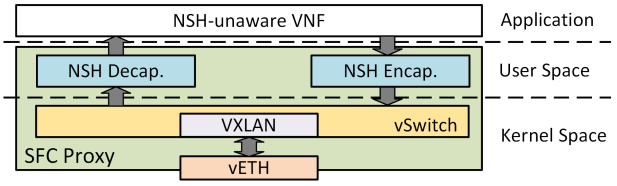
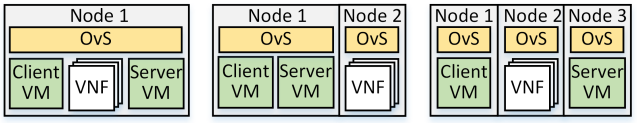


Fig. 6. Our SFC proxy implementation, in which an NSH-unaware VNF runs on top of a pair of user-space network interfaces that decapsulate incoming traffic and encapsulate outgoing traffic with NSH.



(a) Topology 1. (b) Topology 2. (c) Topology 3.

Fig. 7. Topologies of the performance evaluation scenarios of the network virtual network chaining: a) client, server, and chain of VNFs in the same node; b) client and server on a separated node from the node that hosts the chain; c) client, server, and chain on three separated nodes.

to the user-space network interfaces. Our proxy deployment is based on Open vSwitch version 2.8.9 and the NSH encapsulation and decapsulation are performed by OpenFlow 1.4 rules implemented on user-space Open vSwitch datapaths.

VII. Evaluation and Results

We use the SFCPerf framework to evaluate the service function chaining prototype shown in Figure 5. The performance and efficiency are evaluated for different topologies and configurations. We implement the experiments over OPNFV Danube 2.0[5]. The OPNFV platform deploys the NFV-MANO reference architecture based on the cloud operating system OpenStack[6]. The OPNFV is built through the Fuel installer in a scenario that provides the service functions chaining architecture referenced in RFC 7665. This scenario also deploys the SDN controller OpenDaylight, which manages the data link layer, the VNF manager Tacker, and the software switch Open vSwitch compatible with NSH.

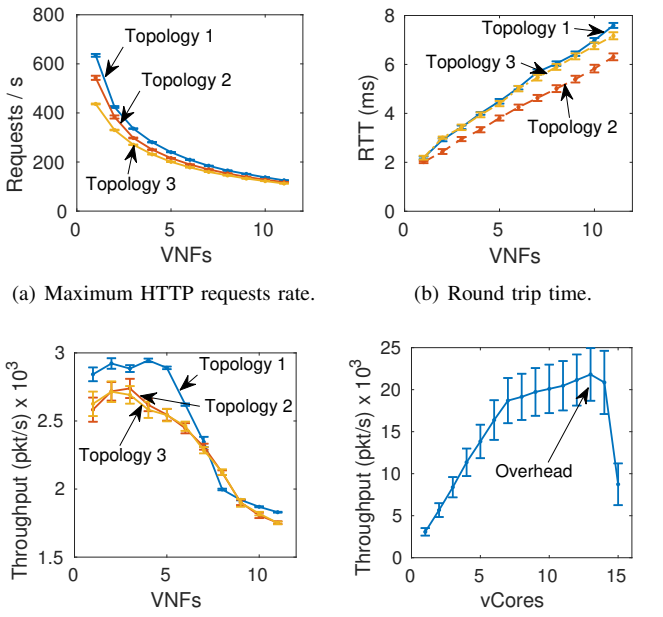
The hardware environment consists of a controller node Intel 8-Core i7-4770 CPU 3.40 GHz processor with 32 GB RAM, and three compute nodes, Intel Xeon processor X5570 2.93 GHz with 96 GB RAM (node 1), Intel 8-Core i7-6700 CPU, 3.40 GHz with 64 GB RAM (node 2) and Intel 8-Core i7-2600 CPU, 3.40 GHz with 32 GB RAM (node 3). All machines are interconnected through a top-of-the-rack switch over 1 Gb/s network interfaces that comprise the five VLANs needed for OPNFV cloud: public, private, management, storage and Preboot Execution Environment.

The performance evaluation of the service function chaining is based on the RFC 7665 architecture using the NSH protocol, as shown in Figure 1. Therefore, SFCPerf analyzes several topologies for different parameters that affect the performance of the chaining. The evaluated metrics include the placement of the endpoint VMs and of the chain of VNFs, defined by each given topology, the overhead that each network function introduces in the chaining, and the number of virtual processing cores requested by a VNF. In addition, the implemented service function chaining, as well as each developed VNF, are evaluated by SFCPerf. The evaluation considers latency, throughput and number of responded HTTP requests. Finally, we evaluate the implemented chaining within a real scenario. We inject real network traffic, which is analyzed, classified and filtered in real time. This evaluation compares an on-line classification with the off-line classification out of the chain for accuracy, and for the efficiency of real-time detection and automatic blocking of malicious flows.

The chosen topologies for evaluation are illustrated in Figure 7. The first topology (Topology 1), illustrated in Figure 7(a), uses only one compute node to instantiate the virtual function chain and the endpoint VMs. Thus, the endpoint VMs and the VNFs compete for network resources of the same physical node, and there are no physical link hops between nodes. Figure 7(b) shows Topology 2, in which the endpoint VMs are instantiated on the same node, and the chain of VNFs on another node. In the third topology (Topology 3), shown in Figure 7(c), the server VM, the client VM, and the chain of VNFs are instantiated in three distinct nodes. Thus, Topologies 2 and 3 require the use of more software switches, since each node has a local software switch (Open vSwitch – OvS) to control its network resources.

Figure 8 shows the impact on performance results obtained in each topology relative to the number of VNFs in the chain and considering a 95% of confidence interval. It is worth mentioning that the VNFs in this specific scenario are extremely simple network functions that only forward incoming packets to the next VNF in the chain. The forwarding consists in decrementing the service index field by 1 in the NSH protocol header, and then the packet is returned to the Open vSwitch, which sends it to the next virtual function. Thus, the processing and the delay are practically only due to the the overhead of handling the NSH protocol. The application responsible for these operations is a tool written in Python (*vxlan_tool*) for tests with the NSH header. Figure 8(a) compares the three topologies in relation to the rate of HTTP requests performed from a client VM to a server VM that traverses the chain. Note that Topology 1 provides the best rate of HTTP requests for short chains of VNFs. The difference, however, becomes negligible when the chain length exceeds 8 VNFs. It demonstrates that for short chains, the overhead introduced by spreading the client, server and VNFs on different nodes is the performance limiting factor. Nevertheless, longer chains introduce an overhead that exceeds this factor. Figure 8(b) shows that the round-trip time in all topologies grows linearly with the increase of the chain length. Topology 2 presented a significantly lower latency increase than the other two, because the client and server VMs are on the same physical node, which decreases the packet round-trip time. The node does

[5]Available at https://www.opnfv.org.
[6]Available at https://www.openstack.org.



(a) Maximum HTTP requests rate.

(b) Round trip time.

(c) Maximum throughput.

(d) Maximum throughput according to the number of virtual cores.

Fig. 8. Impact on the performance of network function chains considering: a) HTTP request rate supported by the chain; b) chain round-trip time; c) chain throughput; and d) throughput of a single VNF on Topology 1 as a function of the number of assigned virtual cores.

not share resources with other VNFs. We conclude that the increase of physical link hops, as well as the sharing of resources on the same node are factors that compromise the end-to-end delay. Hence, Topology 2 presents a fair trade-off of these factors. Figure 8(c), in turn, shows the maximum throughput in packets per second for each topology as a function of the chain length. The packet size used in the experiments is 1334 bytes, which represents the maximum transmission unit for the Ethernet packet sent by the client, which suffers no IP fragmentation with NSH encapsulation. Topology 1 presents a better throughput in relation to the others when chaining few VNFs, due to lower number of physical link hops between nodes. It is important to notice that the increase of the number of VNFs in all the topologies implies resources competition on the node that hosts the chain, which considerably compromises throughput.

The major limiting factor for throughput is the *vxlan_tool* application that decapsulates the NSH packets. This application, by default, operates sequentially in only one processing core, and we extended it for parallel execution on multiple cores. The effect of this change is observed in Figure 8(d), which shows the increased throughput of a VNF in a unit-length chain in which we allocate more dedicated virtual processing cores (vCPUs). In this way, VNF retains more processing power and is able to perform more packet operations per second until it reaches the hypervisor processing limit. It is expected, however, that in the next versions of OPNFV platform, the NSH encapsulation and decapsulation will be implemented as a kernel module of the operating system of virtual machines to assure performance gain.

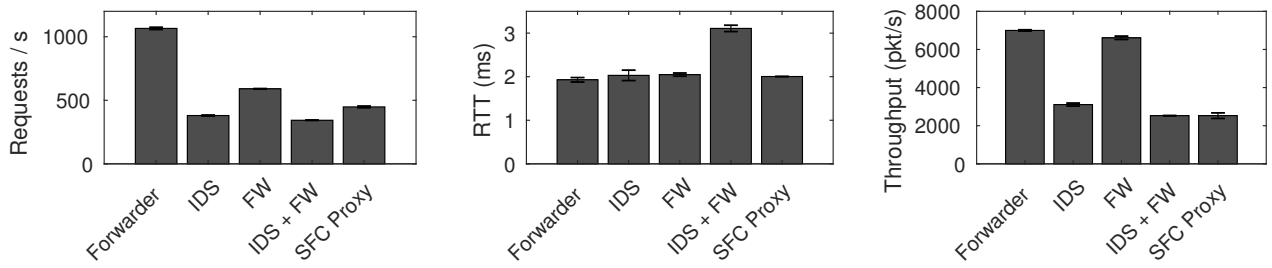
The second evaluated security function is the IDS, which consists of the two modules described in Section VI. The first module acts directly in the chain and performs two basic functions over the traffic, to extract characteristics of the flows in a two-second time window, and, at the same time, to forward the packets to the next VNF. In addition, the VNF publishes the extracted characteristics in the queuing and data flow manipulation system Kafka to be read by the second processing module, running outside the chain.

As SFCPerf assures repeatability, we designed a scenario and performed a comparison experiment of four VNF approaches. Figure 9 shows the performance of one-length chains of two different simple VNFs (a forwarder, and an SFC proxy), two virtual security functions (a firewall, and a IDS), and of the composition of the two virtual security functions. In all chains, Topology 1 is used as a reference, providing only one virtual core for each VNF. It can be observed that the performance of the VNF that only forwards packets is superior to the other virtual functions and, thus, it is used as a baseline. Figure 9(a) and Figure 9(c) show similar results regarding the maximum rate of HTTP requests and the supported throughput of each chain. The firewall VNF presents better results than IDS VNF in both metrics, thus the chain of both VNFs has the performance limited by the IDS, with a small overhead due to the extra hop between the two VNFs. Figure 9(b), however, shows that the latency overload introduced by each virtual security function is very low, remaining at a similar time to the baseline threshold. The chain of firewall and IDS increases by 50% the packet round-trip time, which was already predicted from Figure 8(b). Nevertheless, this value is 33% lower in the case where the firewall and IDS functions separately operate over the traffic. It is worth to highlight that the forwarder, which is implemented with the Python `vxlan_tool`, performed better than the SFC proxy VNF. Although we deploy the NSH-unaware VNF, preceded by the SFC proxy, as an Open vSwitch associated with a Linux Bridge, the overhead of copying packets between user-space and kernel-space contexts slows down the packet handling.

TABLE I
CONFUSION MATRIX OF REAL-TIME FLOW CLASSIFICATION AND BLOCKING. THE FLOWS THAT REACH THE SERVER ARE REPRESENTED AS NORMAL, WHILE THOSE BLOCKED BY OUR VNF AS ATTACK.

	TP	**FP**	**TN**	**FN**
Attack	430	2277	4412795	1658973
Normal	4412795	1658973	430	2277

Finally, we use the SFCPerf framework to evaluate the chain for real-time on-line classification of network traffic dataset of a Brazilian telecommunications operator [27]. We implement the decision tree machine learning algorithm for the traffic classification [27]. First, a fraction of the traffic



(a) Maximum rate of HTTP requests. (b) Average round-trip time. (c) Maximum throughput.

Fig. 9. Impact on the performance introduced by each virtual security function and by the chaining of the two VNFs in relation to: (a) maximum rate of HTTP requests; (b) packet round trip time; and (c) maximum throughput.

(20%), labeled as attack or normal by Suricata IDS[7], was used to train the classifier in an off-line manner. The remainder fraction is injected by the client VM in direction to the server and traverses the two virtual security functions. Table I shows the results of the on-line classification[8], comparing the flows that reach the server with those that were blocked in the core of the network by the virtual firewall. The results show an accuracy of 72.7% for the classification and that 0.02% of the malicious flows were blocked before reaching the server. However, since the proposed firewall is a reactive software defense, there is a real reduction of 15% of the total volume of malicious traffic that reaches the server. The reduction derives from the time required for a flow classification using machine learning. Short-duration attacking flows, or mice flows, have a shorter duration than the analysis window and, thus, are not possible to be blocked in real time. This measurement shows that the prototype was effective against attacks with large data volumes such as denial of service.

VIII. Conclusion

Service function chaining enables the network administrator to provide complex network services comprised of features developed by different VNF manufacturers. Hence, mechanisms that automate SFC benchmarking and accelerate the development of VNFs are essential. In this paper, we proposed SFCPerf[9], an automatic performance evaluation framework for service function chaining. SFCPerf assures repeatability for testing and for comparing virtual network function chains. We also presented a performance evaluation of a service function chain, described by coherent chaining of virtual security functions. We employed the SFCPerf framework to evaluate the chain, composed of an intrusion detection system based on stream data processing in the cloud and a reactive firewall. The results provided by our proposed SFCPerf framework showed the impact in terms of throughput, round-trip time, and request rate of several VNF chaining scenarios. These performance measures were obtained for different chain implementation topologies, chain lengths, and varying number of virtual cores offered to VNFs. Our framework allowed the comparison of the overhead introduced by each virtual function individually. The results showed that the main impact factors on the performance were the number of physical link hops between nodes and the competition for resources at shared physical nodes. Furthermore, the SFC throughput is directly related to the number of cores assigned to the virtual functions, which determines the number of packets that each VNF is able to process.

As future work, we will evaluate the performance of the network function chaining on new topologies and over different NFV platforms, to assess the performance bottlenecks of other network function virtualization approaches.

[7]Available at https://suricata-ids.org.

[8]TP: True Positives; FP: False Positives; TN: True Negatives; FN: False Negatives.

[9]The source code of SFCPerf is available at https://github.com/ijochem/SFCPerf.

Acknowledgment

This research is supported by CNPq, CAPES, FAPERJ, and FAPESP (2015/24514-9, 2015/24485-9, and 2014/50937-1).

References

[1] M. Pattaranantakul, R. He, A. Meddahi, and Z. Zhang, "SecMANO: Towards network functions virtualization (NFV) based security management and orchestration," in *IEEE Trustcom/BigDataSE/ISPA*, Aug. 2016, pp. 598–605.

[2] A. S. da Silva, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho, "ATLANTIC: A framework for anomaly traffic detection, classification, and mitigation in SDN," in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, Apr. 2016, pp. 27–35.

[3] F. Reynaud, F. X. Aguessy, O. Bettan, M. Bouet, and V. Conan, "Attacks against network functions virtualization and software-defined networking: State-of-the-art," in *IEEE NetSoft Conference and Workshops (NetSoft)*, Jun. 2016, pp. 471–476.

[4] I. J. Sanz, M. Andreoni Lopez, D. M. F. Mattos, and O. C. M. B. Duarte, "A Cooperation-Aware virtual network function for proactive detection of distributed port scanning," in *2017 1st Cyber Security in Networking Conference (CSNet'17)*, Rio de Janeiro, Brazil, Oct. 2017.

[5] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, and T. Magedanz, "Service function chaining in next generation networks: State of the art and research challenges," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 216–223, Feb. 2017.

[6] M. C. Luizelli, D. Raz, Y. Sa'ar, and J. Yallouz, "The actual cost of software switching for nfv chaining," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, May 2017, pp. 335–343.

[7] ETSI, "ETSI GS NFV-MAN 001: Network functions virtualisation; management and orchestration," Tech. Rep., 2014.

[8] J. Halpern and C. Pignataro, "Service Function Chaining (SFC) architecture," Internet Requests for Comments, RFC Editor, RFC 7665, October 2015, available at http://www.rfc-editor.org/rfc/rfc7665.txt.

[9] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul, "Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions," in *II ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 19–24.

[10] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patneyt, M. Shirazipour, R. Subrahmaniam, C. Truchan, and M. Tatipamula, "Steering: A software-defined networking for inline service chaining," in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, Oct. 2013, pp. 1–10.

[11] A. Csoma, B. Sonkoly, L. Csikor, F. Németh, A. Gulyas, W. Tavernier, and S. Sahhaf, "ESCAPE: Extensible service chain prototyping environment using Mininet, Click, NETCONF and POX," in *ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 125–126.

[12] F. Callegati, W. Cerroni, C. Contoli, and G. Santandrea, "Dynamic chaining of virtual network functions in cloud-based edge networks," in *1st IEEE Conference on Network Softwarization (NetSoft)*, Apr. 2015, pp. 1–5.

[13] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "ClickOS and the art of network function virtualization," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 459–473.

[14] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker, "Netbricks: Taking the V out of NFV," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*. USENIX Association, 2016, pp. 203–216.

[15] M. Andreoni Lopez, D. M. F. Mattos, and O. C. M. B. Duarte, "Evaluating allocation heuristics for an efficient virtual network function chaining," in *7th International Conference on the Network of the Future (NoF)*, Nov. 2016, pp. 1–5.

[16] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle, "Performance characteristics of virtual switching," in *IEEE 3rd International Conference on Cloud Networking (CloudNet)*, Oct. 2014, pp. 120–125.

[17] F. Callegati, W. Cerroni, C. Contoli, and G. Santandrea, "Performance of network virtualization in cloud computing infrastructures: The OpenStack case," in *IEEE 3rd International Conference on Cloud Networking (CloudNet)*, Oct. 2014, pp. 132–137.

[18] R. Bonafiglia, I. Cerrato, F. Ciaccia, M. Nemirovsky, and F. Risso, "Assessing the performance of virtualization technologies for NFV: A preliminary benchmarking," in *2015 Fourth European Workshop on Software Defined Networks*, Sep. 2015, pp. 67–72.

[19] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[20] J. Mirkovic, T. V. Benzel, T. Faber, R. Braden, J. T. Wroclawski, and S. Schwab, "The DETER project: Advancing the science of cyber security experimentation and test," in *Technologies for Homeland Security (HST), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–7.

[21] R. Riggio, I. G. B. Yahia, S. Latré, and T. Rasheed, "Scylla: A language for virtual network functions orchestration in enterprise WLANs," in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, Apr. 2016, pp. 401–409.

[22] P. Quinn and U. Elzur, "Network service header," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-sfc-nsh-12, February 2017, available at http://www.ietf.org/internet-drafts/draft-ietf-sfc-nsh-12.txt.

[23] OPNFV, "Open platform for NFV," https://www.opnfv.org/, 2017, accessed on 10-06-2017.

[24] D. R. Lopez, "OpenMANO: The dataplane ready open source NFV MANO stack," *Proc. IETF Meet.*, pp. 1–28, Mar. 2015.

[25] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, "NFP: Enabling network function parallelism in NFV," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM'17. New York, NY, USA: ACM, 2017, pp. 43–56.

[26] P. Zave, R. A. Ferreira, X. K. Zou, M. Morimoto, and J. Rexford, "Dynamic service chaining with Dysco," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM'17. New York, NY, USA: ACM, 2017, pp. 57–70.

[27] M. Andreoni Lopez, R. S. Silva, I. Alvarenga, G. Rebello, I. J. Sanz, A. Lobato, D. Mattos, O. C. M. B. Duarte, and G. Pujolle, "Collecting and characterizing a real broadband access network traffic dataset," in *2017 1st Cyber Security in Networking Conference (CSNet'17)*, Rio de Janeiro, Brazil, Oct. 2017.

[28] N. Marz and J. Warren, *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*, 1st ed. Greenwich, CT, USA: Manning Publications Co., 2013.