An NSH-Enabled Architecture for Virtualized Network Function Platforms

Vinícius F. Garcia[1(✉)], Leonardo C. Marcuzzo[1], Giovanni V. Souza[2], Lucas Bondan[3], Jéferson C. Nobre[4], Alberto E. Schaeffer-Filho[3], Carlos R. P. dos Santos[1], Lisandro Z. Granville[3], and Elias P. Duarte Jr.[2]

[1] Federal University of Santa Maria, Santa Maria, Brazil
{vfulber,lmarcuzzo,csantos}@inf.ufsm.br
[2] Federal University of Paraná, Curitiba, Brazil
{gvsouza,elias}@inf.ufpr.br
[3] Federal University of Rio Grande do Sul, Porto Alegre, Brazil
{lbondan,alberto,granville}@inf.ufrgs.br
[4] University of Vale do Rio dos Sinos, Porto Alegre, Brazil
jcnobre@unisinos.br

Abstract. The proper execution of Virtualized Network Functions (VNFs) depends on the employment of platforms specifically created to fulfill multiple Network Function Virtualization (NFV) requirements (*e.g.*, performance, integration, and management). However, existing VNF platforms implement different architectures, thus resulting in proprietary or limited solutions that do not always support important NFV specifications, such as Network Service Header (NSH). In this work, we introduce a comprehensive architecture for VNF platforms that supports the NFV requirements defined by the European Telecommunications Standards Institute (ETSI), while also enabling the execution of NSH. We implemented a VNF platform prototype, on which we conducted a case study, and report a series of performance evaluation experiments. Results demonstrate the intrinsic advantages of supporting NSH and show the flexibility of our architecture in distinct NFV scenarios.

1 Introduction

Computer networks typically rely on dedicated equipment (*i.e.*, middleboxes) to perform common network functions (*e.g.*, NAT translation, intrusion detection and load balancing). However, despite the usual benefits of such middleboxes, including performance and reliability, they offer limited flexibility regarding the design and deployment of advanced network services. Network Functions Virtualization (NFV), in turn, is a networking paradigm that leverages virtualization technologies to decouple network functions from the physical equipment and run such functions as software that executes on commodity hardware [5].

The adoption of NFV presents several benefits, including higher customization and the reduction of Capital and Operational Expenditures

L. Barolli et al. (Eds.): AINA 2019, AISC 926, pp. 376–387, 2020.
https://doi.org/10.1007/978-3-030-15032-7_32

(CAPEX/OPEX). Multiple efforts are being conducted to foster the adoption of NFV technologies with the ultimate goal of encouraging the development of solid foundations that support advanced NFV solutions.

Essentially, a VNF is divided into two main parts: the Network Function (NF) itself and the VNF platform. NF corresponds to the software implementation responsible for packet processing, while the VNF platform is the environment that supports the execution of NFs. VNF platforms are designed taking into account the need to enable the creation of multiple network functions while consuming few computing resources. However, existing VNF platforms (*e.g.*, ClickOS [12] and OpenNetVM [15]) are not created using standardized architectures, thus resulting in solutions that are either proprietary or present serious limitations, such as the lack of support for advanced NFV specifications, in particular Network Service Header (NSH) – a packet header that enables the creation of dynamic service planes [13].

In this paper, we introduce a comprehensive architecture for VNF platforms that strictly adheres to ETSI requirements and provides support for NSH. Our key contributions are: (i) the development of a VNF platform prototype based on ETSI requirements; and (ii) the identification of critical features provided by NSH that enables the development of advanced network services.

The remaining of this paper is organized as follows. Section 2 presents the background on NFV along with its main components, basic requirements, and a review of the literature. In Sect. 3, we propose a architecture for developing VNFs with support for NSH, and instantiate this architecture by describing a running prototype platform implemented to support virtualized network functions. In Sect. 4, we evaluate the performance of our prototype. Finally, in Sect. 5, we conclude this paper with final remarks and an outline of future work.

2 Background and Related Work

In this section, we present the background on Network Functions Virtualization (NFV) and Virtualized Network Function (VNF). We also discuss the characteristics and limitations of existing VNF platforms.

2.1 Network Function Virtualization in a Nutshell

NFV aims to implement Network Functions (NFs) in software, so they can run on commodity hardware by employing common virtualization technologies [5]. Two of the most relevant standardization bodies are expending considerable energy in defining models, methodologies, and concepts in this area: the European Telecommunications Standards Institute (ETSI) and the Internet Engineering Task Force (IETF).

The NFV architectural framework [7] defined by ETSI is composed of three functional blocks: NFV Infrastructure (NFVI), NFV Management and Orchestration (NFV MANO), and Virtualized Network Functions (VNF). NFVI is the collection of physical resources (*e.g.*, processing, storage, and network) needed to execute VNFs. NFV MANO in turn encompasses: the Virtualized Infrastructure Manager (VIM), which controls the physical/software infrastructure; the VNF Manager (VNFM), which is responsible for VNF lifecycle operations (*e.g.*, instantiation, termination, scaling, and migrations); and the NFV Orchestrator (NFVO), which enables the management of network services. Finally, the VNF functional block itself represents the network functions that run on VNF platforms.

The IETF has been working on standardizing the Service Function Chain (SFC) [8], which consists of multiple VNFs working together in a composition that provides a network service. In addition to VNFs, a SFC also includes boundary nodes (*i.e.*, incoming and outgoing points of traffic) and steering specifications. One way to implement those SFCs is by using the architecture defined in the IETF's Request for Comments (RFC) number 7665 [8]. In that RFC, a Network Service Header (NSH) [13] is employed to enable packets to traverse a specific path of VNFs. Each VNF, in turn, can either be NSH-aware or not. NSH-aware VNFs make a request to a proxy element to remove the NSH before processing, after which the proxy element is again invoked to update/reinsert the NSH element. Alternatively, VNFs can process NSH themselves, without relying on external elements.

An NSH encapsulates L3 packets as they are processed by the SFC, and carries information about the Service Function Path (SFP), the current packet location in the SFP, and meta-data provided by the VNFs during packet processing. NSH is subdivided into three meta-headers: Base Header, Service Path Header, and Context Header. The Base Header provides general information about the protocol itself and the next meta-headers data. This meta-header (4 bytes), in turn, carries 5 fields (*i.e.*, version, O bit, TTL, length, and meta-data type), in addition to reserved space for future protocol use. The Service Path Header is also formed by 4 bytes and has 2 fields: Service Path Identifier (identifies the SFC's SFP) and the Service Index (provides the location of the packet in the SFP). Finally, the Context Header can be fixed (16 bytes) or variable to carry meta-data information as the SFC is processed.

VNF platforms are specially designed to host VNFs and their associated components. The ETSI lists the basic requirements for developing such platforms (*e.g.*, hardware independence, elasticity, reliability) [14]. The ETSI also specifies that a VNF platform must host VNFs independently of the underlying hardware. Furthermore, VNF platforms must provide the flexibility expected from the NFV paradigm to support the operations of deployment, scaling, and migration.

Several aspects of VNF platforms, however, are still unexplored in the literature. For example, the lack of standardization on existing VNFs platforms, mostly created without a well defined architecture and missing features such as NSH and SFC support. Existing platforms, detailed next, usually support only a single programming language, run on very specific hypervisors, do not describe a formal architecture, and do not fulfill all the above requirements defined by the ETSI.

2.2 VNF Platforms

Many research efforts concentrate on creating VNF platforms that are capable of executing NFs. However, these VNF platforms do not provide support for more recent NFV specifications, such as NSH. Still, two of the most important VNF platforms today are ClickOS and OpenNetVM, described next.

ClickOS [12] is an optimized platform for running NFs based on Mini-OS, *netmap*, and *Click Modular Router*. ClickOS uses paravirtualization techniques along with several modifications in both Xen and VALE to support fast packet I/O, being able to saturate 10GbE links with a single processing core. Alas, the ClickOS architecture is monolithic and inflexible and supports only a single packet acceleration tool for sending and receiving network traffic to an indivisible NF.

OpenNetVM [15] is a simplified platform that uses containers to execute VNFs on commodity servers along with the packet acceleration framework DPDK [9]. OpenNetVM meets the ETSI scalability requirements because of its lower overhead due to the use of containers, which are a lightweight solution in comparison with virtual machines. The OpenNetVM architecture consists of a packet acceleration tool interconnecting VNFs with Virtualized NICs (VNICs). NFs are created as a single component within a proprietary framework and are deployed on a container core. OpenNetVM also provides an internal router implemented using shared memory, which steers network traffic between multiple VNFs.

Both ClickOS and OpenNetVM unfortunately lack support for important VNF elements (*e.g.*, VNFCs and NSH). The ClickOS architecture is straightforward and does not have any native management method, providing only a minimalist environment to execute simple NFs. Despite OpenNetVM's internal traffic router, the platform is restricted to a single packet acceleration tool and a single packet processing framework, both deployed on a container core and managed by an external native agent. Furthermore, these platforms do not meet all of the requirements specified by the ETSI. ClickOS, for example, can only be executed on the Xen hypervisor and every control operation is performed locally through the XenStore. Also, OpenNetVM presents limitations related to NF instantiation (due to hardware device sharing), portability (migrations are only possible between compatible infrastructures), and security issues with containers (single kernel sharing) [11].

3 VNF Platform Architecture and Prototype

In this section, we introduce an architecture for VNF platforms which supports SFC chaining using NSH. The proposed architecture is designed to be flexible and technology agnostic. Ultimately, we expect this architecture to serve as a template for designing new systems and re-engineering existing ones.

3.1 Architecture Overview

Currently, there is no *de facto* standard for the design and development of VNF platforms, from neither industry nor academia. VNF platforms, however, must be developed to meet multiple strict requirements (*e.g.*, portability, performance, integration, management, and scalability), in order to fulfill the needs of modern networks. Furthermore, the NFV area is evolving, with new technologies being created continuously. Therefore, it is essential to design flexible solutions that support new NFV Enablers (*i.e.*, existing frameworks and technologies that contribute to the development and implementation of NFV) from an ever increasing number of players in the NFV market. VNF platforms must also be created with integration in mind. There are several systems (OSS/BSS, Hypervisors) and elements (NFVI, VNFM, EMS) that must work together with multiple VNFs in order to adequately provide virtualized network services [7].

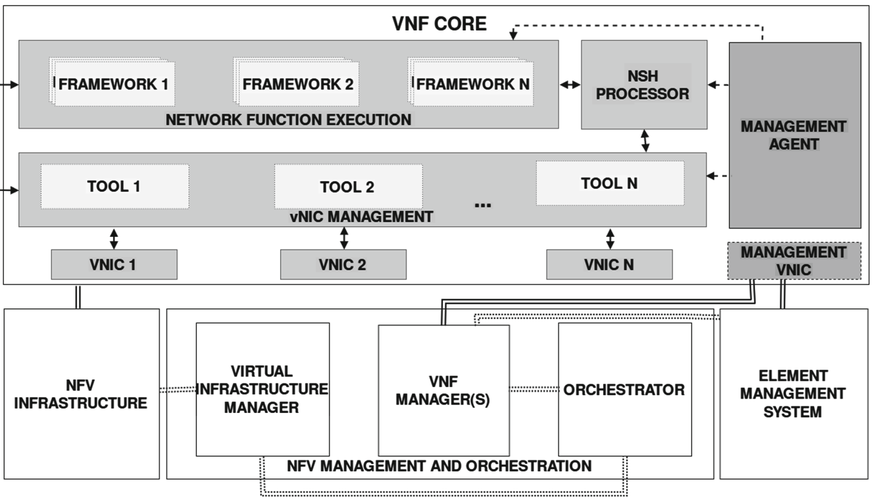

Fig. 1. VNF platform architecture

In this context, we propose a generic and flexible architecture for VNF platforms, (Fig. 1). The proposed platform consists of multiple modules, deployed on a host operating system (called here VNF Core): an module responsible for management access of Virtual Network Interfaces (vNIC), being the main point of input/output of packets from the architecture, a module consisting of a framework for development and execution of network functions, a management agent responsible for configuration of the host operating system and the other modules and, finally, a NSH Processor which provides an abstraction for the network functions regarding the existence of NSH packets.

Network Service Header (NSH) is a new service plane protocol, specified by IETF, inserted onto packets/frames to provide service function paths [13].

Despite its advantages, NSH is not always employed to steer traffic across multiple VNFs. In order to support the cases where NSH is employed or not, we introduced the NSH Processor, which is responsible for manipulating the NSH fields that may be modified when a packet is traversing a network path (*i.e.,* the Service Index - SI; and Context Header - CH). The proposed NSHP also provides the following operations: NSH removal, NSH reinsertion, CH retrieval, and CH update.

Packets are then received by the development frameworks used to implement network functions. Basically, these frameworks include applications (*e.g.*, Click Modular Router [10] and Vector Packet Processing [3]), programming languages (*e.g.*, C, C++, Python), libraries (*e.g.*, Scapy, libtins, libnet), or even single routines that support the construction and handling of network packets.

All the described modules are controlled by an management agent, which is responsible for monitoring and controling the execution of VNFs. Once a VNF is executing, the *retrieve* operations can be used to gather information about the VNF instance (*e.g.*, VNF ID, network interfaces), measuring performance indicators from the VNF Core (*e.g.*, CPU, memory, and network usage), and providing information from the extended agents deployed in the VNF platform.

3.2 NFV Enablers

As a proof of concept of the proposed architecture previously presented, we implemented a prototype that consists of a VNF platform that employs modern and well-accepted NFV enablers. In this context a NFV enabler is any technology used as a basis for the development of NFV ecosystems [5], such as hypervisors, packet acceleration frameworks, virtual routers, and operating systems. Specifically, we employed the following technologies in our solution:

- **Click Modular Router** – a well-known packet processing framework that provides an extensive list of network elements, native support for packet acceleration frameworks, and built-in control methods. The Click framework has been extensively investigated in academia and employed in several efforts to develop VNF platforms, such as ClickOS [12], Click-on-OSv [4], and the platform proposed by Bu *et al.* [2];
- **Intel Data Plane Development Kit (DPDK)** – a packet acceleration framework that provides high throughput with reduced resource consumption (by using PCI passthrough and zero-copy) and supports multiple network interface cards (including paravirtualized ones, such as `virtio` and `netfront`);
- **RESTful Web Services** – an architectural style for the provisioning of Web Services (WSs) with simplified access to resources. RESTful web services are lighter than traditional SOAP-based WSs and are used here as the basis for the management agent to export performance statistics regarding the VNF platform, and to receive management requests from an external VNFM or EMS.

Furthermore, we have chosen Debian as the operating system for the VNF Core. Although this operating system is not explicitly designed to support the NFV requirements (*e.g.*, performance), this decision enabled us to focus on the development of the internal modules without concerns about software compatibility, thus enabling the analysis of the architecture from the functional point-of-view. However, the development of NFV platforms for production environments should benefit from more recent solutions such as OSv, CoreOS, MiniOS, and Alpine.

3.3 Platform Prototype

For handling the vNICs, we initially chose DPDK. However, the platform also supports L2 Sockets to provide increased flexibility. These two options are available and can be selected according to the specific needs of the operator[1]. Although we have employed these two solutions, other solutions (*e.g.*, PF_RING, and netmap) could be used without significant implementation efforts.

The NSH Processor operates as a proxy during the platform execution and can be enabled by the network operator. When enabled, packets first pass through the NSHP in order to remove NSH before steering the packets to the network function (*i.e.*, CMR or Python-based NFs/VNFCs), and for NSH reinsertion before forwarding processed packets back to the vNIC (*i.e.*, DPDK or L2 Sockets). The NFs, when necessary, can access the NSH Context Header to retrieve or replace content using specific libraries we developed (both in Python and CMR)[2]. Notice that, during the NSHP reinsertion operation, the Service Index field is updated.

The Management Agent was developed as a RESTful WS and is able to control (through system calls and CMR's control socket) all the internal components of the platform. For monitoring, Glances[3] was used to recover system-wide statistics, while a custom REST API was developed to meet the management requirements as specified in [1], such as providing information regarding the VNF (*e.g.*, function description, system state, logs) and supporting the reconfiguration of running functions. These operations can be accessed either directly by network operators (through the integrated EMS) or by external EMS and VNF Managers (by using REST calls to the management interface).

To initiate the platform, the network operator must first provide (through the Management Agent) a VNF Package that contains the network function implementation, its descriptor, and settings to initialize the internal modules properly. These settings are used, for example, to enable/disable the NSH Processor. After the initial platform configuration, the NF lifecycle operations (*e.g.*, start, stop, status, and monitoring) become available to the network operator.

[1] The polling method of DPDK is inefficient regarding to CPU resources, while L2 Sockets are not able to achieve high throughput.

[2] By the time of writing, we opted to support only Context Headers of fixed-length [13].

[3] https://nicolargo.github.io/glances/.

The platform prototype supports three execution scenarios: NSH packets with NSH unaware NF, NSH packets with NSH aware NF, and packets without NSH.

4 Evaluation

To evaluate our proposed architecture, a case study of the prototype presented in Sect. 3.2 are presented. An Intel Core i7-4790K@3.60 Ghz server with 8 GB RAM DDR4 running Debian 8 was used. The prototype platform was configured to use L2 Sockets for input/output and NFs developed using both Python3 and/or CMR frameworks. All the experiments were repeated 30 times, considering a confidence level of 95%.

4.1 NSH for VNFs Intercommunication and SFC Steering

We used NSH to improve an existing NFV-based solution to mitigate DDoS attacks called DeMONS [6]. In DeMONS, six separate VNFs were employed to detect malicious traffic and steer it through separate channels with different bandwidth capacities. These VNFs are Manager, Priority Classifier, Firewall, Allocator, Traffic Policing, and Router. The Manager is responsible for orchestrating the environment execution, for example, by monitoring the network load to scale running VNFs. The Priority Classifier, the Firewall, and the Allocator are responsible for identifying and classifying benign traffic, and blocking malicious traffic. Finally, both the Traffic Policing and the Router are responsible for applying user-defined policies (*e.g.*, partial dropping and traffic shaping) on the suspicious flows.

In the original DeMONS, the first three VNFs (*i.e.*, Priority Classifier, Firewall, and Allocator) are connected through a static path and share the traffic reputation through the Manager (acting as a central point of communication). The Priority Classifier uses Intrusion Detection System (IDS) techniques to generate the traffic reputation, and classifies the incoming flows with values ranging from 0 to 1. Reputation 0 means a malicious flow, reputation 1 indicates a benign flow, and values between those limits indicate unclassified traffic. After the classification, the traffic is forwarded to the Firewall, which queries the Manager to verify the traffic reputation, blocks 0 marked flows, and forwards the rest to the Allocator. The Allocator, in turn, steers the traffic to a high priority tunnel (that guarantees QoS for high reputation flows) or to a low priority tunnel (that can be overloaded with low reputation flows). Figure 2-A presents the original DeMONS implementation design.

DeMONS was deployed using the IETF's SFC architecture (*i.e.*, Classifier and Service Function Forwarder) [8] and the VNF Platform prototype developed in this work. NSH was used to enable in-band control for sharing flow reputations and to orchestrate the traffic steering across all the employed NSH aware VNFs (*i.e.*, the SFC). The reputation table (originally at the Manager) is now maintained by the Priority Classifier and shared between the VNFs through the NSH Context Header. In case of benign traffic, the Service Index is decremented

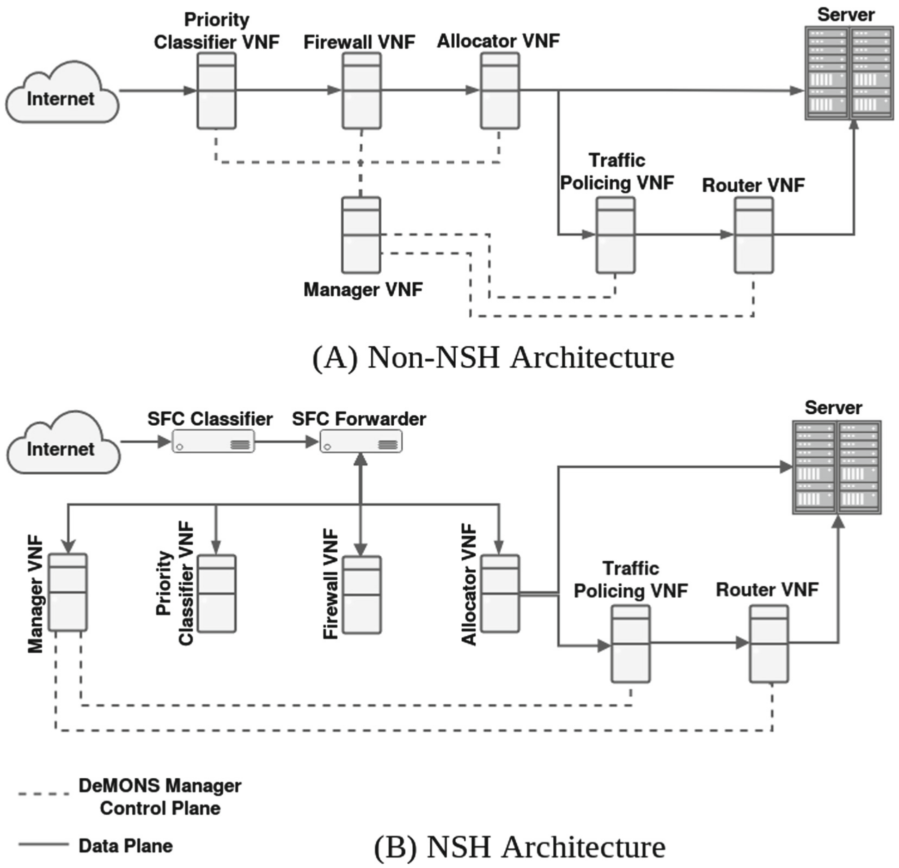

Fig. 2. DDoS mitigation NFV solution (DeMONS)

by two in order to forward the packets directly to the Allocator, thus skipping the Firewall. Finally, the Allocator gets the reputation value directly from the flow packet by using the NSH's Context Header. Figure 2-B presents the adapted DeMONS implementation design.

The use of NSH led to performance improvements in DeMONS due to redesign and the embedded mechanism to exchange control data. The first experiment was executed to evaluate the execution time overhead introduced by retrieving and processing the reputations in both original and modified DeMONS. Two versions of the Packet Filter VNFC were developed in order to operate in both scenarios (NSH and Non-NSH), and were instrumented to measure the elapsed processing time for each packet. Iperf was used to generate the network traffic (UDP packets of 1470 Bytes), with the results presented in Fig. 3.

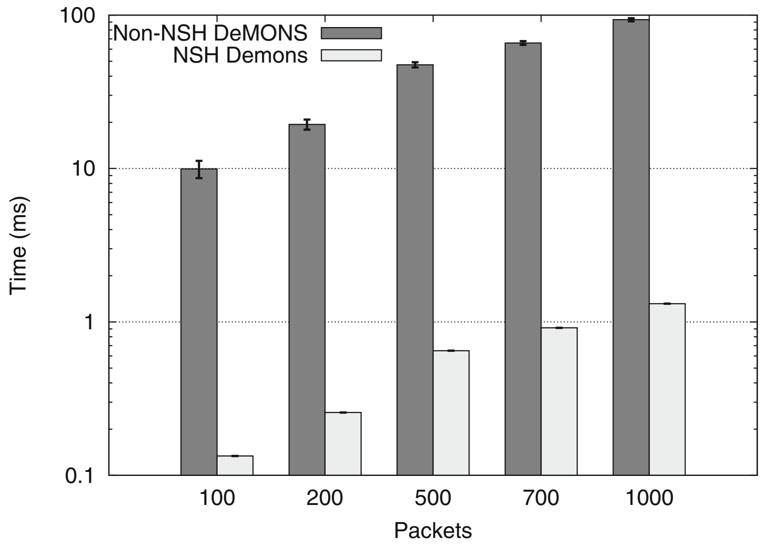

Fig. 3. Reputation retrieval processing time overhead

In the Non-NSH DeMONS, the reputation retrieval process consists in contacting the Manager (through a UDP Socket) to discover the reputations. This operation is performed for all the network packets traversing the VNFC. When NSH is employed, on the other hand, the reputation value is included in the NSH Context Header of each packet by the Priority Classifier. In this case, the Packet Filter just needs to open this header to retrieve the information. As suspected, the same operation (reputation retrieval) leads to significant differences in terms of processing time (in favor to NSH-based), which may affect other performance indicators (*e.g.*, throughput and packet loss). An important observation is that no major differences occurred for packets with different sizes.

The use of a Service Function Forwarder in the NSH DeMONS scenario also demonstrated interesting opportunities for improving the traffic steering overall. This SFC element (*e.g.*, an OpenFlow Switch or a P4-enabled device) steers the network traffic according to the Service Index value present in the NSH Service Path Header. In this way, it is possible to manipulate the VNF execution order by updating the NSH Service Index according to decisions taken during VNF processing. For our case study, the Firewall VNF only processes the malicious traffic in order to collect statistics (*e.g.*, number of discarded packets) and then discards the packets. In addition to not processing benign flows – when malicious traffic is nonexistent (*i.e.*, no attack occurring) – it is possible to temporarily disable the Firewall VNF, thus saving computational resources.

5 Conclusion and Future Work

Network Functions Virtualization (NFV) has been attracting great interest from both industry and academia. However, despite all the advancements in the field, there are still opportunities for research, development, and standardization. For example, there are no widely accepted definitions for the internal architecture of the platforms responsible for executing NFs, neither the processing of NSH inside those VNFs. This leads to a scenario where several platforms (*e.g.*, ClickOS and OpenNetVM) have been created without integration concerns in mind, the result is that none fulfills the complete set of NFV requirements.

This work proposed an architecture for VNF Platforms with NSH support. We specified the basic modules for building a platform, as well as identified existing NFV enablers that can be employed during such development. We also presented a platform prototype that employs the proposed architecture to support the execution of disparate network functions. Finally, we conducted a performance evalution to identify the advantages of employing NSH. The experiment shows the advantages of using NSH when designing SFCs. First, the NSH Context-Header enables the functions to communicate and change information in-band. Second, the Service Index field allows the creation of dynamic SFCs, without the need to *a priori* set the path of VNFs, thus enabling traffic steering to be executed by using common solutions (e.g., Open vSwitch, P4-enabled equipment). Future work includes the investigation on how the current VNF/SFC descriptors (*e.g., TOSCA*) have to be adapted to support NSH, and improve the prototype by supporting novel packet processing frameworks (*e.g.,* VPP) and other virtual network technologies (*e.g.,* netmap).

References

1. Bondan, L., dos Santos, C.R.P., Granville, L.Z.: Management requirements for ClickOS-based network function virtualization. In: 10th International Conference on Network and Service Management (CNSM) and Workshop, pp. 447–450 (2014). https://doi.org/10.1109/CNSM.2014.7014210
2. Bu, C., Wang, X., Huang, M., Li, K.: SDNFV-based dynamic network function deployment: model and mechanism. IEEE Commun. Lett. **22**(1), 93–96 (2018)
3. Cisco: Vector packet processing (2018). https://blogs.cisco.com/tag/vector-packet-processing. Accessed 13 Sept 2018
4. da Cruz Marcuzzo, L., Garcia, V.F., Cunha, V., Corujo, D., Barraca, J.P., Aguiar, R.L., Schaeffer-Filho, A.E., Granville, L.Z., dos Santos, C.R.: Click-on-OSv: a platform for running click-based middleboxes. In: 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pp. 885–886. IEEE (2017)
5. ETSI: Network functions virtualisation - an introduction, benefits, enablers, challenges & call for action (2012). https://portal.etsi.org/NFV/NFV_White_Paper.pdf. Accessed 13 Sept 2018
6. Garcia, V.F., de Freitas Gaiardo, G., da Cruz Marcuzzo, L., Nunes, R.C., dos Santos, C.R.P.: DeMONS: A DDoS mitigation NFV solution. In: 2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA), pp. 769–776. IEEE (2018)

7. GS E: Network functions virtualisation (NFV); architectural framework (2014). https://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf. Accessed 13 Sept 2018
8. Halpern, J.M., Pignataro, C.: Service Function Chaining (SFC) Architecture. RFC 7665 (2015). https://doi.org/10.17487/RFC7665. https://rfc-editor.org/rfc/rfc7665.txt
9. Intel: Data plane development kit (2014). http://dpdk.org. Accessed 13 Sept 2018
10. Kohler, E., Morris, R., Chen, B., Jannotti, J., Kaashoek, M.F.: The click modular router. ACM Trans. Comput. Syst. **18**(3), 263–297 (2000). https://doi.org/10.1145/354871.354874
11. Manco, F., Lupu, C., Schmidt, F., Mendes, J., Kuenzer, S., Sati, S., Yasukata, K., Raiciu, C., Huici, F.: My VM is lighter (and safer) than your container. In: Proceedings of the 26th Symposium on Operating Systems Principles, SOSP 2017, pp. 218–233. ACM (2017). https://doi.org/10.1145/3132747.3132763
12. Martins, J., Ahmed, M., Raiciu, C., Olteanu, V., Honda, M., Bifulco, R., Huici, F.: ClickOS and the art of network function virtualization. In: Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, NSDI 2014, CA, USA, pp. 459–473. USENIX Association, Berkeley (2014)
13. Quinn, P., Elzur, U., Pignataro, C.: Network Service Header (NSH). RFC 8300 (2018). https://doi.org/10.17487/RFC8300. https://rfc-editor.org/rfc/rfc8300.txt
14. SWA EG: Virtual network functions architecture (2014). https://www.etsi.org/deliver/etsi_gs/NFV-SWA/001_099/001/01.01.01_60/gs_NFV-SWA001v010101p.pdf. Accessed 13 Sept 2018
15. Zhang, W., Liu, G., Zhang, W., Shah, N., Lopreiato, P., Todeschi, G., Ramakrishnan, K., Wood, T.: OpenNetVM: a platform for high performance network service chains. In: Proceedings of the 2016 Workshop on Hot Topics in Middleboxes and Network Function Virtualization, HotMIddlebox 2016, NY, USA, pp. 26–31 (2016). ACM, New York. https://doi.org/10.1145/2940147.2940155