Towards a Flexible Architecture for Virtualized Network Function Platforms

Extended Abstract

Vinícius Fülber Garcia
Elias Procópio Duarte Jr.
vfgarcia@inf.ufpr.br
elias@inf.ufpr.br
Federal University of Paraná

Leonardo da Cruz Marcuzzo
Carlos Raniery P. dos Santos
lmarcuzzo@inf.ufsm.br
csantos@inf.ufsm.br
Federal University of Santa Maria

ABSTRACT

Within the context of NFV, the VNF architecture block has not yet been tackled by the research community. Furthermore, a key component of VNFs, VNF Component (VNFC), is not well defined, with definitions ranging from a small piece of code with a specific network purpose to an entire virtual machine running this component. In this paper, we present a flexible architecture for VNFs, where key components can be swapped by other best suited to the operator needs, in order to create tailored VNFs for each infrastructure.

1 INTRODUCTION

Computer networks typically rely on dedicated hardware (middleboxes) to perform common network functions, due to their packet processing performance and reliability. However, middleboxes present many limitations regarding the high CAPital and OPerational EXpenditures (CAPEX and OPEX), the lack of management and maintenance flexibility. Network Function Virtualization is a paradigm which aims to decouple those network functions from their underlying proprietary hardware. To do that, virtualization technologies, based in off-the-shelf equipment, are employed in order to mitigate the middleboxes issues.

One of the main blocks of the NFV architecture is the Virtualized Network Function (VNF), defined by the ETSI as the implementation of a Network Function (NF) which can be deployed on an NFV Infrastructure [10]. Since there is not a standardized architecture for developing VNFs, different approaches are being adopted by the community. For example, ClickOS [8] deploys a VNF as a single Virtual Machine (VM) running a unique NF, while OpenNetVM [14] deploys a VNF as a set of containers, each one with a particular NF and lifecycle management, connected through an umbrella NF Manager. These approaches structural differences, however, violate one of the key NFV paradigm benefits, where VNFs should be interoperable between themselves and between the NFV Infrastructure [9].

Another important definition from ETSI is that of VNF Components (VNFC) [10]. Components are internal operational elements of a single VNF that must be mapped to an individual virtual instance (*e.g.*, process virtualization, virtual machines or containers). While this definition is appropriate for the development of interoperable internal building blocks of a VNF, the VNFCs have often been overlooked by the research community. Currently, VNFs are released without any documentation of their internal modules and connections, effectively resulting in proprietary virtualized functions.

In this paper, we present an overview of a generic and flexible VNF platform architecture. Besides foreseeing the adoption of Network Service Header (NSH) [11] and Element Management System (EMS) [7], the architecture also enables the definition and execution of multiple VNFCs connected by an internal manager. These characteristics facilitate the development of modular VNF platforms, where internal modules can be easily swapped in order to create a tailored VNF to specific deployment needs. It is a work in progress, but the authors believe this architecture can ease the development of new NFV enablers, fostering the evolution of the NFV paradigm.

2 ARCHITECTURE

The proposed VNF platforms architecture, depicted in Figure 1, consists of six main modules deployed on a host operating system (called here VNF Core): *(i)* Virtual Network Subsystem, *(ii)* Internal Router, *(iii)* NSH Processor, *(iv)* Packet Processing Subsystem, *(v)* Management Agent, and *(vi)* Extended Agents. Each module performs specific operations within the VNF and can process network packets from both the data and control planes (depicted by solid lines) and the management plane (depicted by dashed lines). External interfaces to the VNF Core are also defined in the architecture to enable the use of virtualized resources (available in the NFVI) and to support both management and orchestration operations (by using EMS and VNFM systems).

Each module of the architecture is designed to be loosely coupled and with well-defined access interfaces. In this way,

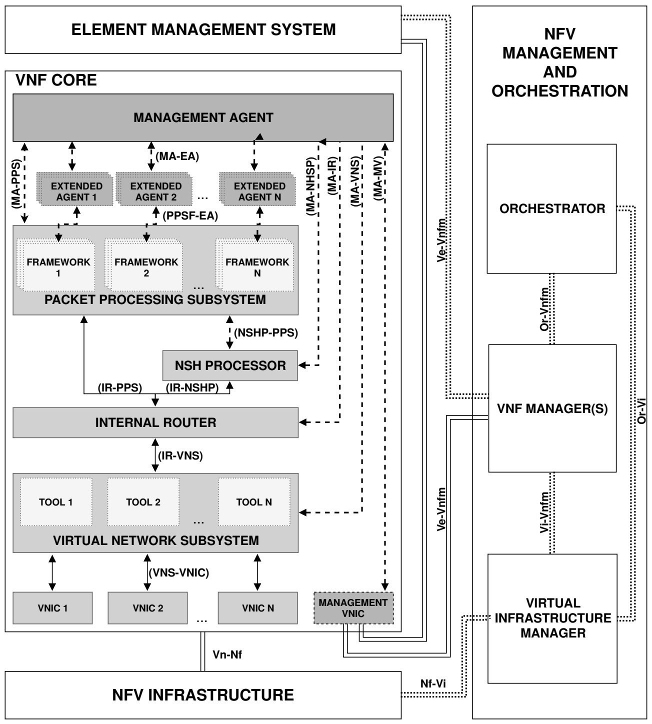

Figure 1: VNF Platforms Architecture

the modules can be redesigned or replaced as new software becomes available. Modules are described below:

- **VNF Core**: it consists of a virtualized instance (*e.g.*, Virtual Machine and Container) controlled by a hypervisor or container manager. The VNF Core provides the necessary computational resources and software environment, such as memory, virtualized CPU cores, virtualized network interfaces, programming languages, and frameworks, for the other modules execution.
- **Virtual Network Subsystem** (VNS): this module is responsible for accessing the Virtual Network Interface Controllers (VNICs) – provided by the hypervisor – for sending/receiving network packets. The VNS can be deployed using L2 Sockets, to packet acceleration tools such as Intel DPDK [5], NetMap [12], PacketShader [2], PF_RING/DNA [4], and OpenOnload [13].
- **Internal Router** (IR): once the network packets are captured by the VNS, they are forwarded internally to the VNF Core by the Internal Router. Once the Internal Router is initialized, it uses communication channels (*e.g.*, using shared memory, pipes, sockets) to forward the network packets between the VNS, the Packet Processing Subsystem, and the NSH Processor.
- **NSH Processor** (NSHP): despite the advantages of using NSH to steer traffic across multiple VNF [11], it is optinal. In order to support both cases, we define the NSH Processor, which provides an abstraction for the network functions regarding the existence of NSH packets. Specifically, when the Internal Router detects a NSH packet, it forwards the packet to be processed by the NSHP.
- **Packet Processing Subsystem** (PPS): this module corresponds to the development frameworks used to implement network functions. These frameworks include applications *e.g.*, Click Modular Router [6] and Vector Packet Processing [1]), programming languages (*e.g.*, C, Python), libraries (*e.g.*, Scapy, libtins), or even single routines that support the construction and handling of network packets.
- **Management Agent** (MA): the primary goal of a MA is to monitor and control the execution of VNFs. Furthermore, it is also responsible for coordinating the execution of all internal modules of the VNF platform. MA provides five main operations: request, retrieve, start, stop, and monitor. The request operation receives a VNF Package (VNFP) [3] from the network operator and deploys the specified VNF instance. Once a VNF is executing, retrieve operations can be used to gather information about the VNF instance (*e.g.*, VNF ID, network interfaces). The start and stop operations are responsible for the VNF lifecycle management. Finally, the monitoring operation is responsible for measuring performance indicators from the VNF Core (*e.g.*, CPU, memory, and network usage) and providing information retrieved from the extended agents deployed in the VNF platform.
- **Extended Agent** (EA): this module is controlled by the Management Agent and is used to monitor/control each network function or component. It is supposed to be developed by the creator of the VNF/VNFC, as it acts on the individual management data of those implementations (*e.g.*, number of discarded packets by a firewall). This module must provide at least one standard operation which we call "list". This operation is used by MA to discover all the management data that can be accessed by network operators.

3 CONCLUSION

In this paper, a brief overview of a modular VNF platform architecture was presented. Although this is still a work in progress, a prototype platform, fully compliant with the presented architecture and based on technologies such as DPDK, Click Modular Router and Python 3, is currently being developed. As for the next steps, the developed prototype will be deployed on a test bed reflecting a common network scenario, in order to gather performance measurements and identify possible bottlenecks and optimizations in the architecture.

REFERENCES

[1] Cisco. 2018. Vector Packet Processing. (2018). https://blogs.cisco.com/tag/vector-packet-processing Acessed in 2019-03-11.

[2] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. 2010. PacketShader: a GPU-Accelerated Software Router. *Computer Communication Review* 40 (2010), 195–206.

[3] ETSI IFA. 2018. *VNF Descriptor and Packaging Specification.* Technical Report. European Telecommunications Standards Institute.

[4] Intel. 2013. PF_RING: High-speed packet capture, filtering and analysis. (2013). https://www.ntop.org/products/packet-capture/pf_ring Acessed in 2019-03-11.

[5] Intel. 2014. Data Plane Development Kit. (2014). http://dpdk.org Acessed in 2019-03-11.

[6] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. 2000. The Click Modular Router. *ACM Trans. Comput. Syst.* 18, 3 (2000), 263–297.

[7] ETSI MANSG. 2014. *Network Function Virtualisation (NFV): Management and Orchestration.* Technical Report. European Telecommunications Standards Institute.

[8] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. 2014. ClickOS and the art of network function virtualization. In *11th USENIX Conference on Networked Systems Design and Implementation*. USENIX Association, Seattle, WA, USA, 459–473.

[9] ETSI NFVISG. 2013. *Network Functions Virtualisation (NFV): Virtualisation Requirements.* Technical Report. European Telecommunications Standards Institute.

[10] ETSI NFVISG. 2014. *Network Function Virtualisation (NFV): Terminology for Main concepts in NFV.* Technical Report. European Telecommunications Standards Institute.

[11] Paul Quinn, Uri Elzur, and Carlos Pignataro. 2018. *Network Service Header (NSH) - RFC 8300.* Technical Report. Internet Engineering Task Force.

[12] Luigi Rizzo. 2012. netmap: A Novel Framework for Fast Packet I/O. In *2012 USENIX Annual Technical Conference (USENIX ATC 12)*. USENIX Association, Boston, MA, USA, 101–112.

[13] Solarflare. 2008. OpenLoad. (2008). https://www.openonload.org/ Acessed in 2019-03-11.

[14] Wei Zhang, Guyue Liu, Wenhui Zhang, Neel Shah, Phillip Lopreiato, Gregoire Todeschi, K.K. Ramakrishnan, and Timothy Wood. 2016. OpenNetVM: A Platform for High Performance Network Service Chains. In *2016 Workshop on Hot Topics in Middleboxes and Network Function Virtualization*. Association for Computing Machinery, New York, NY, USA, 26–31.