

Modeling Native Software Components as Virtual Network Functions

*Original*

Modeling Native Software Components as Virtual Network Functions / Baldi, Mario; Bonafiglia, Roberto; Risso, FULVIO GIOVANNI OTTAVIO; Sapio, Amedeo. - STAMPA. - (2016), pp. 605-606. ( Proceedings of the 2016 ACM Conference on Special Interest Group on Data Communication (SIGCOMM 2016) Florianopolis (BRA) August 2016) [10.1145/2934872.2959069].

*Availability:*

This version is available at: 11583/2652788 since: 2016-10-11T22:48:47Z

*Publisher:*

ACM

*Published*

DOI:10.1145/2934872.2959069

*Terms of use:*

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

*Publisher copyright*

(Article begins on next page)

# Modeling Native Software Components as Virtual Network Functions

Mario Baldi, Roberto Bonafiglia, Fulvio Rizzo, Amedeo Sapia  
Department of Control and Computer Engineering, Politecnico di Torino, Italy  
{mario.baldi, roberto.bonafiglia, fulvio.rizzo, amedeo.sapia}@polito.it

## ABSTRACT

Virtual Network Functions (VNFs) are often realized using virtual machines (VMs) because they provide an isolated environment compatible with classical cloud computing technologies. However, VMs are demanding in terms of required resources (CPU and memory) and therefore not suitable for low-cost devices like residential gateways. Such equipment often runs a Linux-based operating system that includes by default a (large) number of common network functions, which can provide some of the services otherwise offered by simple VNFs, but with reduced overhead. In this paper those native software components are made available through a Network Function Virtualization (NFV) platform, thus making their use transparent from the VNF developer point of view.

## CCS Concepts

•**Networks** → *Cloud computing; Middle boxes / network appliances;*

## Keywords

Network Functions Virtualization; Virtual Network Functions; Service Orchestration

## 1. INTRODUCTION

While Network Service Providers (NSPs) could benefit from a flexible infrastructure that can rapidly and efficiently provide dedicated, on-demand network services, so far they have not been able to leverage it due to the complexity of deploying middleboxes in the network. Network Functions Virtualization (NFV) has the potential to overcome this by exploiting virtualization tech-

niques, typical of cloud computing, to instantiate Virtualized Network Functions (VNFs) in compute nodes with unprecedented agility. However, current NFV implementations are designed for (centralized) data centers, while NSPs leverage a distributed infrastructure consisting of heterogeneous devices. More specifically, it would be particularly beneficial to offer NFV capability in Customer Premise Equipment (CPE), which is particularly challenging because it is usually based on low-cost hardware. On the other hand, CPE operating systems are often some Linux flavor, which embeds a large number of software-based “native” network functions, such as firewall and NAT (e.g., `iptables`), virtual switch (e.g., `linuxbridge`), and more.

This paper proposes a solution to integrate such functions in an existing NFV infrastructure so that while resource-hungry VNFs are run in the NSP data center, simpler ones are run in the CPE, possibly as **Native Network Functions** (NNFs). Our solution facilitates the provision of services that require Network Functions (NFs) close to the end user (e.g., IPsec terminator), enabling the possibility to execute both VNFs and NNFs, which combines the benefits of flexibility and low execution overhead. In order to support NNF integration, the compute controller in an NFV server is extended with an additional plugin that manages all the NNFs available on the node, along with standard VNFs.

## 2. ARCHITECTURE

Our proposal is based on the compute node presented in [1], which follows closely the NFV architecture. As shown in Figure 1, a local orchestrator receives a Network Functions Forwarding Graph (NF-FG) and instantiates the required VNFs. For each NF-FG a new software switch, called Logical Switch Instance (LSI), is created in order to steer traffic among the corresponding VNFs in the right order, while a base LSI is in charge of classifying the traffic received by the node and delivering it to the proper NF-FG-specific LSI. VNFs are instantiated and managed by a compute manager through ad-hoc drivers matching the specific VNF support technology (e.g., VM, Docker, DPDK process), while each LSI is managed by its own OpenFlow controller that dynamically inserts the proper rules in flow table(s). All

