Journal of Internet Services and Applications

RESEARCH Open Access

Adaptive placement & chaining of virtual network functions with NFV-PEAR

Gustavo Miotto[1], Marcelo Caggiani Luizelli[2], Weverton Luis da Costa Cordeiro[1*] and Luciano Paschoal Gaspary[1]

Abstract
The design of flexible and efficient mechanisms for proper placement and chaining of virtual network functions (VNFs) is key for the success of Network Function Virtualization (NFV). Most state-of-the-art solutions, however, consider fixed (and immutable) flow processing and bandwidth requirements when placing VNFs in the Network Points of Presence (N-PoPs). This limitation becomes critical in NFV-enabled networks having highly dynamic flow behavior, and in which flow processing requirements and available N-PoP resources change constantly. To bridge this gap, we present NFV-PEAR, a framework for adaptive VNF placement and chaining. In NFV-PEAR, network operators may periodically (re)arrange previously determined placement and chaining of VNFs, with the goal of maintaining acceptable end-to-end flow performance despite fluctuations of flow processing costs and requirements. In parallel, NFV-PEAR seeks to minimize network changes (e.g., reallocation of VNFs or network flows). The results obtained from an analytical and experimental evaluation provide evidence that NFV-PEAR has potential to deliver more stable operation of network services, while significantly reducing the number of network changes required to ensure end-to-end flow performance.

Keywords: NFV, Placement, Chaining, Network functions

1 Introduction

Network Function Virtualization (NFV) is a relatively novel paradigm that aims at migrating functions like routing and caching, from proprietary appliances (middleboxes) to software-centric solutions running on virtual machines. Such migration provides several benefits, e.g. reduced total cost of ownership and maintenance, cheaper network function updates (instead of expensive middlebox hardware upgrades), as well as more flexible placement and chaining of network functions in the infrastructure [1].

NFV has experienced advances on various fronts, from the design and deployment of virtual network functions (VNFs) [2, 3] to their operation and management [4, 5]. In spite of progresses made, many research opportunities remain. One is related to the VNF placement and chaining problem. In summary, it involves determining where to place VNFs given a set of Network Points of Presence (N-PoPs), and how to steer network flows between them, as specified in Service Function Chains (SFCs). To materialize flow steering, Software Defined Networking (SDN) [6] can be considered a convenient ally, as it enables VNFs to be placed and chained in a highly flexible way. The complexity of this problem comes however from requirements and constraints that need to be satisfied upon placement and chaining, like computing power (at N-PoPs, where functions will be placed), bandwidth (between N-PoPs), and end-to-end delay. This problem, which was proven to be NP-hard [7], has been widely studied, with several optimization objectives proposed (e.g., minimize operational costs or network resource utilization [3, 8, 9]).

An important limitation of known approaches to VNF placement and chaining is that they consider, when computing how to best deploy a set of SFCs, VNF operating costs and resources available at N-PoPs as being fixed and immutable. In real-world environments, however, both the costs and available resources can change dynamically [10], depending on the network load. As a result, flow processing requirements, as specified in the SFCs, can be violated during peak hours. A traditional approach to

*Correspondence: weverton.cordeiro@inf.ufrgs.br
[1]Institute of Informatics — Federal University of Rio Grande do Sul, Porto Alegre, Brazil
Full list of author information is available at the end of the article

overcome this issue is analyzing the behavior of an individual VNF (firewall, for example) and deploying more VNFs in response to increasing loads. The individualized search for local solutions, as in the firewall example, may not lead to a global optimum regarding the balance between supply and demand in function placement and flow chaining. More importantly, it can lead to resource waste because of failure to explore idle flow processing capacity in VNFs/N-PoPs.

To fill in this gap, in a previous work we proposed NFV-PEAR, a framework for adaptive VNF orchestration [11]. Our goal was to enable (re)arrangement of previously assigned network functions and flow chaining, in parallel to the instantiation of new SFCs, in order to deal with the dynamic behavior of flows and fluctuations in resource availability in N-PoPs. To this end, we seek to (re)chain flows through VNFs with available bandwidth and computing power, as well as (re)organize VNFs into N-PoPs with more resources available. In this paper, we extend our previous work by providing: (*i*) a more detailed discussion on the formal model to ensure the best provision of SFCs in face of dynamic changes in demand and/or costs associated with networking equipment (virtual or not); (*ii*) an overview of the reference architecture and application programming interface for (re)design and deployment of SFCs, agnostic of virtualization and infrastructure technologies; (*iii*) a description of a proof-of-concept prototypical implementation of NFV-PEAR; and (*iv*) a more detailed evaluation on the efficacy and effectiveness of NFV-PEAR. From the results obtained via analytical and experimental evaluation, we observed that network resource consumption became evenly distributed among active VNFs, when compared to non-reconfigurable NFV environments. Furthermore, it became possible to reroute network flows with varying bandwidth/computing power demands, paving way for minimizing flow requirement violations.

The remainder of this paper is organized as follows. In Section 2 we provide empirical evidence on how VNF performance is strongly influenced by varying network load. In Section 3 we present an Integer Linear Programming (ILP) model for adaptive VNF provisioning. In Section 4 we describe NFV-PEAR, our solution for adaptive VNF provisioning and orchestration. In Section 5 we focus on implementation aspects of a proof-of-concept prototype, and in Section 6 we discuss evaluation scenarios and results achieved. In Section 7 we survey most prominent related work. Finally, in Section 8 we close the paper with final considerations and perspectives for future work.

2 Impact of network load on VNF performance

In the context of an adaptive framework such as NFV-PEAR, it is imperative to understand how VNFs are performing. In order to motivate and illustrate how performance indicators are affected as network functions are subjected to different traffic patterns, a series of experiments were carried out in a typical NFV environment. Next, we present a summary of our key findings, considering CPU, throughput, and packet loss metrics.

The experiments were performed in a controlled environment comprised of two servers, A and B, equipped with 1 Intel Xeon processor E5-2420 (1.9GHz, 12 cores and 15MB cache), 32GB of RAM (1333MHz), 1TB SAS hard drive, 1 Gbps network interface card (NIC), and Fedora GNU/Linux 21 (kernel v3.17). The NIC on server A was directly connected to the NIC in Server B. On server A, a KVM hypervisor and an Open vSwitch virtual switch were installed. On top of KVM we deployed a virtual machine with two logical Ethernet interfaces, 1 vCPU and 1GB of RAM. The virtual switch was connected to the physical NIC and to the virtual machine interfaces. On server B, two Docker containers were installed, each with a logical Ethernet interface, and an Open vSwitch connected to the containers and the physical NIC.

The experiment scenario was set up as follows. On server B, the containers worked as Iperf client and server, configured in UDP mode; on server A, the KVM virtual machine forwarded the packets between their interfaces. During the experiment, traffic originated by the Iperf client (running on B) went through the virtual machine (on A) and returned to the Iperf server (in B). This organization was chosen so that the cost of generating traffic would not interfere with the performance of the virtual machine, actual target of our measurement evaluation. In addition, two distinct experiments were performed, with the following configurations: (*i*) virtual machine with static routing table and (*ii*) virtual machine with routing by network function running on Click Router [12]. In each experiment, measured CPU usage includes cycles spent by the host operating system, virtual machine, and other processes involved in the experiment configuration. The results shown are an average of 30 runs.

Observe in Fig. 1 that packet loss start to occur as CPU usage approaches 100%, making it possible to predict when network function performance starts degrading. For example, with CPU usage as high as 95% for 400 Mbps throughput, packet loss occur at a rate of around 0.05%. The loss rate increases to around 0.1% with CPU usage at 100% for 600 Mbps, and to 10% for 700 Mbps. Observe also the overhead introduced by running IP routing as a network function on top of Click, which even increases exponentially for packet loss as CPU usage approaches 100%. With regard to memory usage, no statistically significant variations were observed.

It is important to mention that higher throughput could have been achieved with hardware acceleration technologies (like Intel DPDK and SR-IOV). Such optimizations would certainly push observable bottlenecks to points

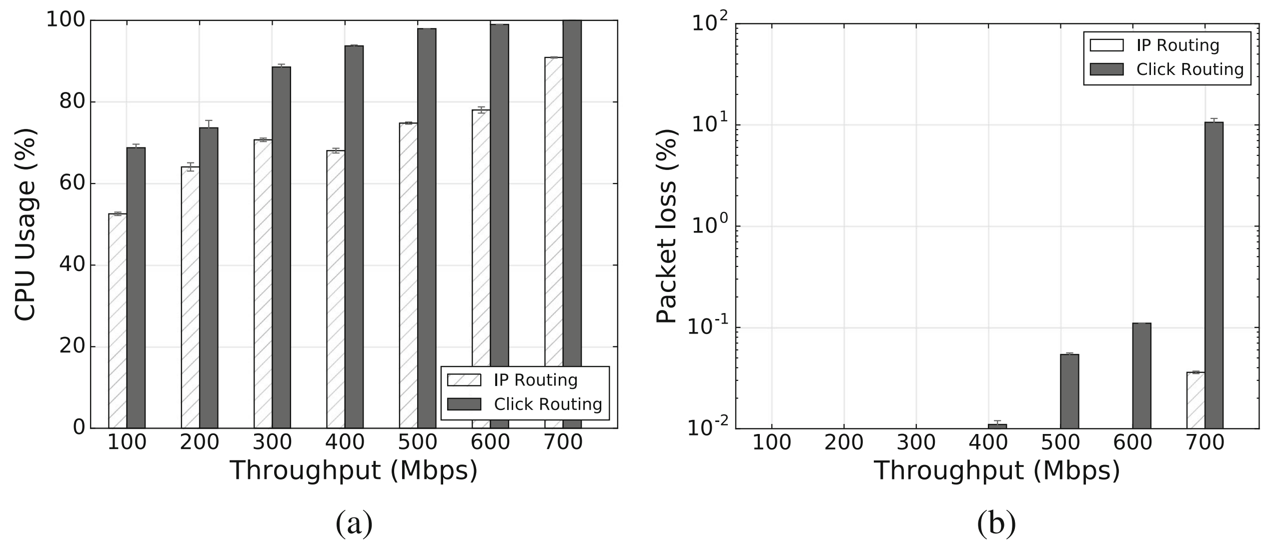

Fig. 1 Measurement metrics and their relationship with SFC performance. **a** CPU usage vs. throughput. **b** Packet loss vs. throughput

beyond those plotted in the graphs, but they would inevitably occur. In summary, the results and discussion above reinforce the importance of the adaptive mechanism being proposed in this work. More specifically, NFV-PEAR enables fine-tuning the provisioning of virtualized function chains to counteract VNF performance degradation or changes in network traffic profile.

3 A formal model for dynamic VNF provisioning

To deal with the dynamic behavior of network flows and to reorganize the allocation of VNFs without wasting physical resources or having performance degradation, it is necessary to revisit VNF allocation models and heuristics available in the literature. To this end, we use an adapted version of the model proposed by Luizelli et al. [7, 13], which formalizes the static placement and chaining of virtual functions using a set of constraints in a linear system.

Next we describe our proposed approach for adaptive VNF provisioning, starting with formal notation (Subsection 3.1), followed by the Integer Linear Programming model (Subsection 3.2). In our model, superscript letters ^{P} and ^{S} indicate symbols related to physical resources and SFCs, respectively. Similarly, ^{N} and ^{L} refer to N-PoPs/endpoints[1] and links that connect them. Finally, ^{H} is used to denote subgraphs of an SFC. For convenience, Table 1 presents the complete notation used in the formulation.

3.1 Model notation and description

Model input. The model proposed by Luizelli et al. [13] considers as input a set of SFCs $\mathcal{Q}$ and a physical infrastructure instance $p \in \mathcal{P}$, the latter being a triple $p = \left(N^P, L^P, E^P\right)$. N^P represents the set of nodes in the infrastructure (N-PoPs or routing devices), while pairs $(i,j) \in L^P$ are unidirectional physical links. Bidirectional links are represented by two links in opposite directions (i.e., (i,j) and (j,i)). The set of tuples $E^P = \left\{\langle i, r\rangle \mid i \in N^P \wedge r \in \mathbb{N}^*\right\}$ contains the location (represented as a unique numeric identifier) of each N-PoP. The proposed model captures the following constraints related to physical resources: computing power of N-PoPs (represented by C^P_i), bandwidth $\left(B^P_{i,j}\right)$, and link delay $D^P_{i,j}$. Note that our model captures packet loss indirectly, since that such losses occur due to exhausted computing power capacity at N-PoPs (as discussed in Section 2). Note however that packet loss may also occur due to factors not related to resource usage, like software/hardware failure, misconfiguration, etc.

SFCs $q \in \mathcal{Q}$ represent any forwarding topology. An SFC is represented by a triple $q = \left(N^S_q, L^S_q, E^S_q\right)$. The set N^S_q represents the virtual nodes (i.e., endpoints and VNFs), while L^S_q represents the virtual links that connect them. Note that each SFC q has at least two endpoints, which are given in advance by $E^S_q = \left\{\langle i, r\rangle \mid i \in N^S_q \wedge r \in \mathbb{N}^*\right\}$, where r is a numeric identifier for node $i \in N^S_q$. In addition, each SFC captures the following requirements related to virtual resources: processing required by a VNF i $\left(\text{represented by } C^S_{q,i}\right)$, the minimum bandwidth required for traffic between VNFs (or endpoints) i and j $\left(\text{represented by } B^S_{q,i,j}\right)$, and maximum latency tolerable between any pair of endpoints $\left(\text{represented by } D^S_q\right)$.

Table 1 Glossary of symbols and functions related to the optimization model

Symbol	Formal specification	Definition
Superscripts and subscripts		
P		Physical infrastructure entity
S		SFC entity
Sets and set objects		
$p \in \mathcal{P}$	$p = \left(N^P, L^P, E^P\right)$	Physical infrastructure instance, composed of nodes and links
$i \in N^P$	$N^P = \{i \mid i \text{ is a N-PoP}\}$	Network points of presence (N-PoPs) in the physical infrastructure
$(i,j) \in L^P$	$L^P = \left\{(i,j) \mid i,j \in N^P\right\}$	Unidirectional links connecting pairs of N-PoPs *i* and *j*
$\langle i,r \rangle \in E^P$	$E^P = \{\langle i,r \rangle \mid i \in N^P \wedge r \in \mathbb{N}^*\}$	Identifier *r* of the actual location of N-PoP *i*
$m \in \mathcal{F}$	$\mathcal{F} = \{m \mid m \text{ is a function type}\}$	Types of virtual network functions available
$j \in \mathcal{U}_m$	$\mathcal{U}_m = \{j \mid j \text{ is an instance of } m \in \mathcal{F}\}$	Instances of virtual network function *m* available
$\mathcal{Q}$		Set of Service function chaining (SFC) requests to be deployed
$q \in \mathcal{Q}$	$q = \left(N^S_q, L^S_q, E^S_q\right)$	A single SFC request, composed of VNFs and their chainings
$i \in N^S_q$	$N^S = \{i \mid i \text{ is a VNF instance or endpoint}\}$	SFC nodes (either a network function instance or an endpoint)
$(i,j) \in L^S_q$	$L^S_q = \left\{(i,j) \mid i,j \in N^S\right\}$	Unidirectional links connecting SFC nodes
$\langle i,r \rangle \in E^S_q$	$E^S_q = \{\langle i,r \rangle \mid i \in N^S \wedge r \in \mathbb{N}^*\}$	Required physical location *r* of SFC endpoint *i*
H^S_q		Distinct forwarding paths (subgraphs) contained in a given SFC *q*
$H^H_{q,i} \in H^S_q$	$H^H_{q,i} = \left(N^H_{q,i}, L^H_{q,i}\right)$	A possible subgraph (with two endpoints only) of SFC *q*
$N^H_{q,i}$	$N^H_{q,i} \subseteq N^S_q$	VNFs that compose the SFC subgraph $H^H_{q,i}$
$L^H_{q,i}$	$L^H_{q,i} \subseteq L^S_q$	Links that compose the SFC subgraph $H^H_{q,i}$
$y'_{i,m,j}$		Denotes whether there was a previous VNF placement
$\delta'_{i,q,j}$		Denotes whether there was a previous assignment of flow to VNF
$\lambda'_{i,j,q,k,l}$		Denotes whether there was a previous flow chaining
Parameters		
ϕ	$\phi \in \mathbb{R}_+, \phi \geq 0$	Percentage of capacity of VNFs that can be violated.
α, β, and γ		Weight factors of the ILP model.
$C^P_i \in \mathbb{R}_+$		Computing power capacity of N-PoP *i*
$B^P_{i,j} \in \mathbb{R}_+$		One-way link bandwidth between N-PoPs *i* and *j*
$D^P_{i,j} \in \mathbb{R}_+$		One-way link delay between N-PoPs *i* and *j*
$C^S_{q,i} \in \mathbb{R}_+$		Computing power required for network function *i* of SFC *q*
$B^S_{q,i,j} \in \mathbb{R}_+$		One-way link bandwidth required between nodes *i* and *j* of SFC *q*
$D^S_q \in \mathbb{R}_+$		Maximum tolerable end-to-end delay of SFC *q*
Functions		
f^{type}_m	$f^{type} : N^P \cup N^S \rightarrow \mathcal{F}$	Type of some given virtual network function (VNF)
$f^{cpu}_{m,j}$	$f^{cpu} : (\mathcal{F} \times \mathcal{U}_m) \rightarrow \mathbb{R}_+$	Computing power associated to instance *j* of VNF type *m*
f^{delay}_m	$f^{delay} : \mathcal{F} \rightarrow \mathbb{R}_+$	Processing delay associated to VNF type *m*
Variables		
$y_{i,m,j} \in Y$	$Y = \{y_{i,m,j}, \forall i \in N^P, m \in \mathcal{F}, j \in \mathcal{U}_m\}$	VNF placement
$\delta_{i,q,j} \in \Delta$	$\Delta = \left\{\delta_{i,q,j}, \forall i \in N^P, q \in \mathcal{Q}, j \in N^S_q\right\}$	Assignment of required network functions/endpoints
$\lambda_{i,j,q,k,l} \in \Lambda$	$\Lambda = \left\{\lambda_{i,j,q,k,l}, \forall (i,j) \in L^P, q \in \mathcal{Q}, (k,l) \in L^S_q\right\}$	Chaining allocation
$\overline{y}_{i,m,j} \in \overline{Y}$	$\overline{Y} = \{\overline{y}_{i,m,j}, \forall i \in N^P, m \in \mathcal{F}, j \in \mathcal{U}_m\}$	Denotes whether an VNF placement changes
$\overline{\delta}_{i,q,j} \in \overline{\Delta}$	$\overline{\Delta} = \left\{\overline{\delta}_{i,q,j}, \forall i \in N^P, q \in \mathcal{Q}, j \in N^S_q\right\}$	Denotes whether an assignment of flow to VNF changes
$\overline{\lambda}_{i,j,q,k,l} \in \overline{\Lambda}$	$\overline{\Lambda} = \left\{\overline{\lambda}_{i,j,q,k,l}, \forall (i,j) \in L^P, q \in \mathcal{Q}, (k,l) \in L^S_q\right\}$	Denotes whether a flow chaining changes

For simplicity, we assume that each SFC q has a set of virtual paths[2] represented by H_q. Each element $H_{q,i} \in H_q$ is a possible path in the subgraph q, with a source and a destination. Subsets $N^H_{q,i} \subseteq N^S_q$ and $L^H_{q,i} \subseteq L^S_q$ contain, respectively, the VNFs and the virtual links belonging to the path $H_{q,i}$.

The set $\mathcal{F}$ denotes the types of VNFs available (firewall, proxy, *etc.*). VNFs can be instantiated at most $\mathcal{U}_m$ times. We define $f^{type} : N^P \cup N^S \to \mathcal{F}$ for the type of a given VNF, which can be instantiated in an N-PoP or be part of a request. In addition, functions $f^{cpu} : (\mathcal{F} \times \mathcal{U}_m) \to \mathbb{R}_+$ and $f^{delay} : \mathcal{F} \to \mathbb{R}_+$ denote power and delays related to a VNF. We assume that the provisioned VNFs can have a higher demand than their pre-determined capacity (over commitment). The parameter ϕ $\{\phi| \in \mathbb{R}_+, \phi \geq 0\}$ defines the percentage of capacity of VNFs that can be violated.

Model output. The model solution is expressed by sets of binary variables, described next. Variables $Y = \left\{ y_{i,m,j}, \forall i \in N^P, m \in \mathcal{F}, j \in \mathcal{U}_m \right\}$ indicate a VNF placement. In other words, they indicate if an instance j of a network function m is mapped to N-PoP i. Similarly, variables $\overline{Y} = \left\{ \overline{y}_{i,m,j}, \forall i \in N^P, m \in \mathcal{F}, j \in \mathcal{U}_m \right\}$ indicate if the current placement of a VNF j has changed in relation to its previous placement, given by $y'_{i,m,j}$.

Variables $\Delta = \left\{ \delta_{i,q,j}, \forall i \in N^P, q \in \mathcal{Q}, j \in N^S_q \right\}$ represent the assignment of a requested VNF (or a flow) to a provisioned VNF. That is, it indicates whether node j (being a VNF or an endpoint), required by SFC q, is assigned to the i-th (N-PoP) node. Similarly, variables $\overline{\Delta} = \left\{ \overline{\delta}_{i,q,j}, \forall i \in N^P, q \in \mathcal{Q}, j \in N^S_q \right\}$ indicate that VNF (or flow) j of SFC q remains allocated to the same instance, in relation to an earlier assignment given by $\delta'_{i,q,j}$.

Finally, variables $\Lambda = \left\{ \lambda_{i,j,q,k,l}, \forall (i,j) \in L^P, q \in \mathcal{Q}, (k,l) \in L^S_q \right\}$ indicate a chaining provisioning in the physical infrastructure, *i.e.*, the virtual link (k,l) from SFC q is assigned to the physical link (i,j). Variables $\overline{\Lambda} = \left\{ \overline{\lambda}_{i,j,q,k,l}, \forall (i,j) \in L^P, q \in \mathcal{Q}, (k,l) \in L^S_q \right\}$ indicate, in turn, that the virtual link (k,l) of SFC q remains allocated to the physical link (i,j), in relation to an earlier assignment given by $\lambda'_{i,j,q,k,l}$.

3.2 Model formulation

The proposed model considers a multi-objective function, which simultaneously minimizes (i) resources consumed in the infrastructure (i.e., in N-PoPs, VNFs, and physical links), and (ii) (possible) changes in mappings due to fluctuation of allocated demand (e.g., provisioning of new VNFs, SFC reassignments, and VNF flow reassignments).

The first part of the objective function minimizes network resource consumption; it is materialized by reducing the number of allocated VNFs (described by y), and length of flow chainings (described by λ). The second part of the equation refers to the changes made in the infrastructure, and is defined by three components. The first one refers to the minimization of changes in the placement of already allocated VNFs (described by $\overline{y}$); the second one refers to minimization of modifications of existing chaining (described by $\overline{\lambda}$); and the third one captures changes related to flows (or SFCs) (re)assignment to VNFs (described by $\overline{\delta}$). Each component is weighted, respectively, by α, β, and γ, according to defined priorities.

Objective:

$$\textbf{Min.} \quad \left(\sum_{i \in N^P} \sum_{m \in \mathcal{F}} \sum_{j \in \mathcal{U}_m} y_{i,m,j} + \sum_{(i,j) \in L^P} \sum_{q \in \mathcal{Q}} \sum_{(k,l) \in L^S_q} \lambda_{i,j,q,k,l} \right) -$$

$$\left(\alpha \sum_{i \in N^P} \sum_{m \in \mathcal{F}} \sum_{j \in \mathcal{U}_m} \overline{y}_{i,m,j} + \beta \sum_{(i,j) \in L^P} \sum_{q \in \mathcal{Q}} \sum_{(k,l) \in L^S_q} \overline{\lambda}_{i,j,q,k,l} \right.$$

$$\left. + \gamma \sum_{i \in N^P} \sum_{q \in \mathcal{Q}} \sum_{k \in N^S_q} \overline{\delta}_{i,q,k} \right)$$

Subject to:

$$\sum_{m \in \mathcal{F}} \sum_{j \in \mathcal{U}_m} y_{i,m,j} \cdot f^{cpu}_{m,j} \leq C^P_i \qquad \left(\forall i \in N^P\right) \tag{1}$$

$$\sum_{q \in \mathcal{Q}} \sum_{j \in N^S_q : f^{type}_j = f^{type}_m} C^S_{q,j} \cdot \delta_{i,q,j} \leq \phi \cdot \sum_{j \in \mathcal{U}_m} y_{i,m,j} \cdot f^{cpu}_{m,j} \tag{2}$$

$$\left(\forall i \in N^P\right) (\forall m \in \mathcal{F})$$

$$\sum_{q \in \mathcal{Q}} \sum_{(k,l) \in L^S_q} B^S_{q,k,l} \cdot \lambda_{i,j,q,k,l} \leq B^P_{i,j} \qquad \left(\forall (i,j) \in L^P\right) \tag{3}$$

$$\sum_{i \in N^P} \delta_{i,q,j} = 1 \; (\forall q \in \mathcal{Q}) \left(\forall j \in N^S_q\right) \tag{4}$$

$$\delta_{i,q,k} \cdot l = \delta_{i,q,k} \cdot j \quad \left(\forall \langle i,j \rangle \in E^P\right) (\forall q \in \mathcal{Q}) \left(\forall \langle k,l \rangle \in E^S_q\right) \tag{5}$$

$$\delta_{i,q,k} \leq \sum_{m \in \mathcal{F}} \sum_{j \in \mathcal{U}_m : m = f(k)} y_{i,m,j} \left(\forall i \in N^P\right) (\forall q \in \mathcal{Q}) \left(\forall k \in N^S_q\right) \tag{6}$$

$$\sum_{j \in N^P} \lambda_{i,j,q,k,l} - \sum_{j \in N^P} \lambda_{j,i,q,k,l} = \delta_{i,q,k} - \delta_{i,q,l} l,$$

$$(\forall q \in \mathcal{Q}) \left(\forall i \in N^P\right) \left(\forall (k,l) \in L^S_q\right) \tag{7}$$

$$\sum_{(i,j)\in L^P} \sum_{(k,l)\in L^H_{q,t}} \lambda_{i,j,q,k,l} \cdot D^P_{i,j} + \sum_{i\in N^P} \sum_{k\in N^H_{q,t}} \delta_{i,q,j} \cdot f^{delay}_k \leq D^S_q (\forall q \in \mathcal{Q}) \left(\forall \left(N^H_{q,t}, L^H_{q,t}\right) \in H^S_q\right) \quad (8)$$

$$\bar{y}_{i,m,j} = y'_{i,m,j} \cdot y_{i,m,j} \quad \left(\forall i \in N^P\right) (\forall m \in \mathcal{F})(\forall j \in \mathcal{U}_m) \quad (9)$$

$$\bar{\delta}_{i,q,j} = \delta'_{i,q,j} \cdot \delta_{i,q,j} \quad \left(\forall i \in N^P\right) (\forall q \in \mathcal{Q}) \left(\forall j \in N^S_q\right) \quad (10)$$

$$\bar{\lambda}_{i,j,q,k,l} = \lambda'_{i,j,q,k,l} \cdot \lambda_{i,j,q,k,l} \left(\forall (i,j) \in L^P\right)(\forall q \in \mathcal{Q})\left(\forall (k,l) \in L^S_q\right) \quad (11)$$

The sets of constraints that make up the model are described below. The first three refer to resource limitations of the physical infrastructure. Constraint set (1) ensures that the sum of all instances of VNFs provisioned in a given N-PoP does not exceed the available computational capacity. Set (2) ensures that the demand required by the SFC flows does not exceed the provisioning capacity of the VNFs. Note that the provisioned capacity of the VNFs can be exceeded (during peak hours, for example) by a factor ϕ. Finally, set (3) ensures that the demands of the provisioned chains on a given physical link do not exceed the bandwidth available on the link.

Constraint sets (4)-(6) guarantee proper placement of virtual resources. Constraint set (4) ensures that each element of an SFC is mapped to the infrastructure. In turn, set (5) ensures that the SFCs' endpoints are mapped to certain devices of the infrastructure. Set (6) guarantees the availability of instances of VNFs in the N-PoPs in which the requests of the SFCs are mapped. That is, if a VNF requested by an SFC is mapped to a given N-PoP i, then (at least) one instance of the VNF is placed and running in i.

Constraints on the SFC chaining are described by the sets (7) and (8). Constraint set (7) ensures that there is a valid path in the physical infrastructure between all endpoints and SFC VNFs. In turn, set (8) ensures that the paths adopted to route the traffic respect the maximum delay limits between the endpoints. The first part of the equation refers to the delay associated to the physical links, while the second part refers to the delay incurred by packet processing in the VNFs themselves.

Finally, constraint sets (9) - (11) determine the similarity of SFCs placement and chaining in relation to a given known previous mapping (denoted by set P). Sets (9), (10) and (11) define, respectively, the similarity of variables related to VNF placement (variables y), to the assignment of SFCs to the placed VNFs (variables δ), and related to the adopted chaining (variables λ). Observe that the purpose of such equations is to identify cases in which allocation variables invert the assumed values from 1 to 0. These cases particularly identify when the allocations are modified.

4 Adaptive VNF placement and chaining with NFV-PEAR

After presenting the ILP model for adaptive placement and chaining of VNFs, in this section we introduce NFV-PEAR: an architecture for virtual network function deployment and orchestration[3]. NFV-PEAR relies on the proposed ILP model to allow the dynamic reallocation of network functions in response to oscillations in the demands of processing flows. Our architecture was designed in line with the main building blocks recommended by the ETSI MANO (Management and Orchestration) interface standard [14].

An overview of our architecture is illustrated in Fig. 2. It highlights the *optimization* and *deployment* layers, described in Sections 4.1 and 4.2, respectively. Figure 2 also highlights interactions with the SDN controller and NFV platform in use in the infrastructure, and the relationship between the architecture elements and the MANO interface building blocks. The Optimization Layer of the proposed architecture, for example, provides at least in part the expected services for the "orchestrator" building block of the MANO interface.

4.1 Optimization layer

The Optimization Layer aggregates the modules responsible for optimizing and planning the instantiation and chaining of SFCs in the infrastructure. Note that both deployed *and* to-be-deployed SFCs are processed by these modules, when (re)planning VNF allocation in the infrastructure.

The Optimizer module is responsible for computing the best possible allocation of VNFs in the network, considering the deployed and to-be-deployed SFCs mentioned above, as well as information about the current state of the infrastructure (endpoints, N-PoPs/VNFs and their resources available, links, etc.). To this end, Optimizer implements the ILP model discussed in the previous section. The output of this module — the solution for the ILP model in the given scenario — is forwarded to Planner.

The Planner module is responsible for determining algorithmically the best way to carry out, in practice, the necessary changes on VNF placement in the network and their corresponding chaining. The goal of Planner is to keep the infrastructure in a state close to optimal operation, with a minimum number of changes performed. Several strategies can be adopted to ensure smooth transition

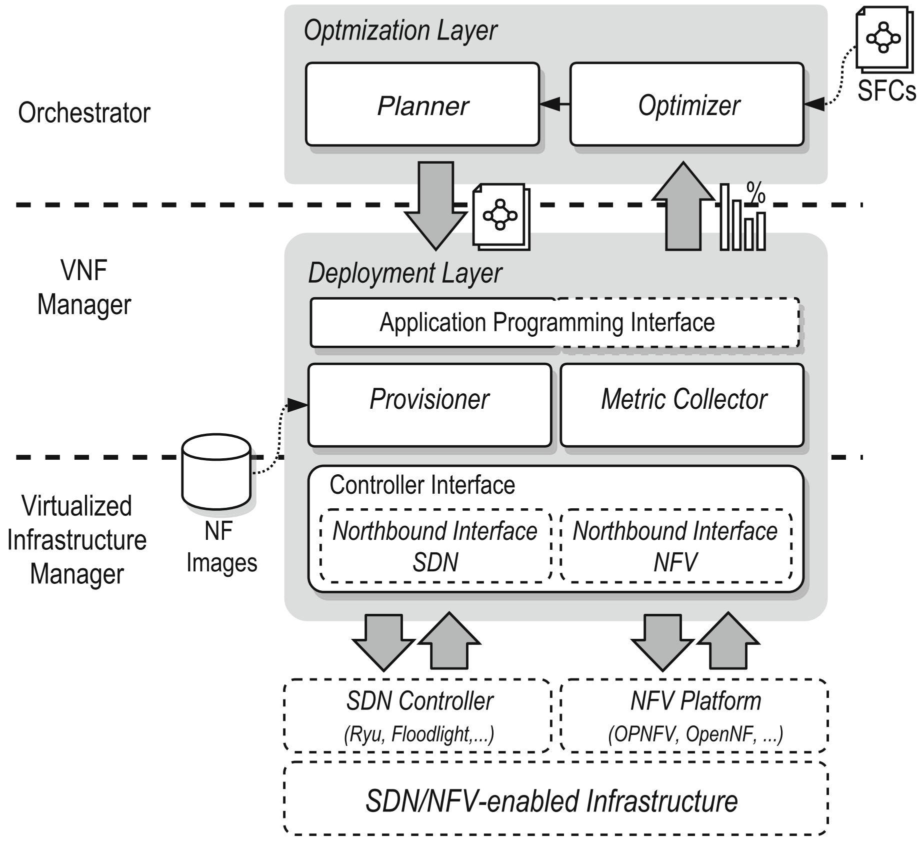

Fig. 2 Overview of the NFV-PEAR architecture

between states that the infrastructure must undergo for avoiding service disruption [15, 16].

4.2 Deployment layer

The Deployment Layer brings together the modules responsible for provisioning the SFCs in the physical network. The Provisioner module is responsible for VNF placement and chaining, according to the mapping of SFCs received from the Optimization Layer. The Metric Collector module monitors the VNFs deployed in the network and consolidates their operation statistics. It also gauges VNF operation states to identify reallocations required to deal with fluctuations in network traffic. The performance metrics we considered, and the methodology we used to gauge their importance, are described in Section 3. The consolidated VNF performance measures are passed on to the Optimization Layer.

Both modules communicate with the *Controller Interface* to perform the orchestration/monitoring activities of VNFs in the physical infrastructure. This interface is made up of two sub-modules, (*i*) *SDN northbound interface*, responsible for translating chain installation requests and state queries (for example, from switches) to the protocol used by the SDN controller, and (*ii*) *NFV northbound interface*, responsible for adapting requests relevant to the VNFs to the protocol used by the NFV controller of the infrastructure. A detailed definition of the Controller Interface is left for future work.

4.3 NFV-PEAR application programming interface

To facilitate the integration process with other solutions compatible with the MANO interface, the Deployment Layer modules expose a programming interface (API)[4] for orchestration and simplified deployment of VNFs. Its purpose is reducing the complexity and burden of SDN network programming (e.g., handling OpenFlow rules and installing them into switches), as well as VNF management. These tasks are essential in an NFV/SDN environment, as flow chaining is performed with OpenFlow, and VNFs are materialized using virtualization technologies like containers. In Table 2 we present and describe some methods available. Observe from the architectural view shown in Fig. 2 that our API currently exposes methods from the Provisioner module only. In a future release of

Table 2 A brief overview of the NFV-PEAR API

Class	Description

`class SFC(sfc_id, edges_list, dict):`
`def deploy_sfc()`
`def deploy_nf(nfunction, pop)`
`def enable_nf(nfunction)`
`def deploy_flow()`
`def enable_flow()`
	A class to materialize Service Function Chaining (SFC) documents. Each instance of this class must have an id, an array with flow steering specifications, and a dictionary that contains mapping information of network functions (NFs) into Network Points-of-Presence (N-PoPs). The class contains methods to deploy the SFC as a whole, and also to deploy and enable NFs individually, and deploy and enable flow steering between NFs (and between NFs and endpoints). The `deploy_sfc()` method deploys an SFC. Internally, it calls `deploy_nf()`, `deploy_flow()`, `enable_nf()`, and `enable_flow()` methods. The `deploy_nf()` method creates and returns an NF instance, receiving as parameter an NF image and an N-PoP instance. Finally, `deploy_flow()` deploys all flows of an SFC.

`class NfData(nfunction, npop, enabled):`
`def enable()`
`def disable()`
	A class to maintain NF operation data. Each NfData object points to an instance of NFunction and N-PoP. It also has a flag indicating if the NF is in operation, and methods to enable/disable its operation.

`class NFunction(nf_id, type):`	A class to represent NF instances. Each NFunction instance must have the identification number of the NF, and a string representing the NF type (ex: "Load Balancer"). The class constructor receives as input a network function id and type.

`class NPop(npop_id, location):`
`def add_deploy(deploymentFunction)`
`def is_deployed()`
	A class to represent N-PoP instances. The class constructor receives as input an N-PoP id and the location of the switch to which the N-PoP belongs.

NFV-PEAR we will extend the API to expose methods from the Metric Collector module.

In Fig. 3 we illustrate the usage of our API, written in Python. The code includes the definition of flow chains in the forward direction, i.e., carrying flows from host `h1` to `h3`, and also in the backward direction, i.e., from `h3` to `h1`. Each link belonging to a chain is a list of 3-tuple edges `(u,v,d)` with a source node `u`, a destination node `v`, and a dictionary `d`. The dictionary contains {`src_ip:x, dst_ip:y, npop: Bool`} (with a Boolean flag indicating if either source or destination is connected to an N-PoP. This list represents the paths `(source -> destination, destination -> source)` of an SFC.

The code snippet follows with the definition of an N-PoP ("npop31") and a VNF (firewall). The SFC is then created, considering those N-PoP and VNF definitions and their chaining (as specified in the "nfs_npop" dictionary). The last method, `deploy_sfc()` deploys the SFC into the infrastructure.

5 Prototypical implementation

Next we discuss our NFV-PEAR prototype. Our discussion is guided by the architectural view shown in Fig. 2, and focuses on the optimization (Section 5.1) and deployment (Section 5.2) layers. Afterwards, we describe the infrastructure used for evaluation (Section 5.3).

5.1 Optimization layer

As mentioned in Section 4, the Optimization Layer aggregates the Optimizer and Planner modules. The Optimizer (and the ILP model shown in Section 3) is implemented using CPLEX Optimization Studio version 12.4[5] and shell scripting, and receives as input (*i*) a text file containing information about the requested SFCs, expressed using CPLEX, and (*ii*) a text file containing infrastructure measurements. A shell script parses the SFC and measurement files, and invokes CPLEX to run the model. It then generates a JSON output (illustrated in Fig. 4), containing the computed flow placement and chaining.

The Planner is implemented using Python 2.7. It parses the Optimizer JSON output and deploys the requested changes in the network. As mentioned earlier, Planner must perform as few changes as possible to minimize network disruption. To this end, it uses the networkX library [17] to represent SFCs as graphs and compute the difference between the current network graph and the one suggested by Optimizer. Depending on the number of differences, the suggestion is either accepted (and the network changes, deployed) or discarded. This behavior can be customized; in our experiments, at least two changes must occur for accepting the suggestion. Planner uses the classes and methods from the Deployment Layer API discussed earlier to perform the changes.

5.2 Deployment layer

The modules in the Deployment Layer are also implemented using Python. This layer exposes the API described in Section 4.3, and receives method calls from Planner to deploy changes, these calls are relayed to the Provisioner. To perform changes, SFC information must be represented following the Python class definition

```
#!/usr/bin/python
from nfvsdnapi import SFC, NPop, NFunction

def load_sfcs():
  links = [
      # SFC links for flow handling in the forward direction
      ('h1', 'switch1', {'src_ip': 'h1', 'dst_ip': 'h3', 'npop': False}),
      ('switch1', 'switch2', {'src_ip': 'h1', 'dst_ip': 'h3', 'npop': False}),
      ('switch2', 'switch3', {'src_ip': 'h1', 'dst_ip': 'h3', 'npop': False}),
      ('switch3', 'npop31', {'src_ip': 'h1', 'dst_ip': 'h3', 'npop': True}),
      ('switch3', 'h3', {'src_ip': 'h1', 'dst_ip': 'h3', 'npop': True}),
      # SFC links for flow handling in the backward direction
      ('h3', 'switch3', {'src_ip': 'h3', 'dst_ip': 'h1', 'npop': False}),
      ('switch3', 'npop31', {'src_ip': 'h3', 'dst_ip': 'h1', 'npop': True}),
      ('switch3', 'switch2', {'src_ip': 'h3', 'dst_ip': 'h1', 'npop': True}),
      ('switch2', 'switch1', {'src_ip': 'h3', 'dst_ip': 'h1', 'npop': False}),
      ('switch1', 'h1', {'src_ip': 'h3', 'dst_ip': 'h1', 'npop': False}),
  ]

  npop31 = NPop('npop31') # Definition of an N-PoP
  fw = NFunction(0, "Firewall") # Definition of a network function
  nfs_npop = {fw: npop31} # NF to N-PoP dictionary
  sfc = SFC(0, links, nfs_npop) # SFC creation
  sfc.deploy_sfc() # Deployment of the SFC

if __name__ == '__main__':
  load_sfcs()
```

Fig. 3 Example of usage of NFV-PEAR API, considering a subset of the definitions shown in Table 2

shown in Table 2. That will enable the provision module to process and make the method calls to deploy it; these calls are done to the modules in the Controller Interface (described next).

The Metric Collector module uses SSH (secure shell) to obtain VNF-related metrics from the N-PoPs they are deployed. To this end, the Paramiko library [18] is used. The collected information is consolidated into a local NoSQL database, TinyDB [19]. There are two relevant settings for metric collection: (*i*) interval between collections (seconds), and (*ii*) VNF CPU threshold (percentage). The latter is related to a trigger for Optimizer, to be invoked when the CPU of some VNF reaches the established threshold.

In the Controller Interface, the SDN northbound interface corresponds to a Web Server Gateway Interface (WSGI) part of the Ryu controller, and enables external access to the methods defined in the SDN[6] control application. The communication with the SDN controller is done via HTTP GET calls to a REST (Representational State Transfer) web service; information about network flows is also passed on these calls. The NFV northbound interface corresponds to an RPC server (implemented using Spyne [20]), that exposes the NFV platform methods through Simple Object Access Protocol (SOAP). The communication with the NFV Platform is performed by means of a SOAP client, which accesses the methods related to NF instantiation and deployment.

5.3 Infrastructure

Our testbed SDN-NFV infrastructure includes the SDN controller, NFV platform, and devices used to materialize the deployment of network functions and flows. It is important to mention that our conceptual solution and its implementation are agnostic of our software choices for materializing our testbed.

We used Ryu [21] as SDN controller, providing as input a file containing the network topology, in a format similar

```
{ "sfc": [ { "id": 0, "_comment": "SFC identification number",
            "_comment": "Definition of nodes in the SFC. There are three nodes in this SFC:
                 ↪  two of them are endpoints, and one is a network function. The
                 ↪ attribute "location" refers to the switch to which the node is
                 ↪ associated to",
            "nodes": [ { "id": 0,
                         "type": "end-point",
                         "location": 0 },
                       { "id": 1,
                         "instance": 1,
                         "nfid": 0,
                         "type": "network-function",
                         "location": 1 },
                       { "id": 2,
                         "type": "end-point",
                         "location": 2 } ],

            "_comment": "Definition of links connecting the nodes. There are two links: one
                 ↪  connecting endpoint 0 and the network function (id 1), and another
                 ↪ connecting the network function and endpoint 2",
            "links": [ { "source": 0,
                         "target": 1,
                         "position": [ { "source": 0,
                                         "target": 1 } ] },
                       { "source": 1,
                         "target": 2,
                         "position": [ { "source": 1,
                                         "target": 2 } ] } ]
          } ] }
```

Fig. 4 CPLEX output of our model, illustrating key SFC features

to that obtained with the UNIX `ifconfig`. An example of a topology can be seen in Fig. 5. Links are described by means of one pair node-interface for origin and another one for destination (separated by colon), in which “node” can be a switch or end host. From the topology description, Ryu starts its execution and connects to all switches on the network. It also starts the REST server, waiting for the requests from upper layer modules.

Our NFV platform is materialized using a Python application that connects to N-PoPs for deploying and enabling network functions. Access to virtual machines is done through SSH (using Paramiko). Note that, in our prototype, *deploy* means upload the network function image to the N-PoP/VM, and *enable* means execute it in the N-PoP/VM. Our platform has been designed focusing on module isolation, thus enabling one to change the technology used to create VMs with few code changes.

For materializing network functions, we use Click Router [12]. In this case, VNFs correspond to VMs running click router, with configuration parameters described in a specific format file. Figure 6 shows an example of a click router configuration of a firewall[7]. One of the

```
# Links from switch s1 to host h1 and switch s2
link: s1-eth1:h1-eth0 s1-eth3:s2-eth3

# Links from switch s2 to N-PoP p21 and switches s1 and s3
link: s2-eth1:p21-eth0 s2-eth2:p21-eth1 s2-eth3:s1-eth3 s2-eth4:s3-eth4

# Links from switch s3 to switch s2 and host h3
link: s3-eth1:h3-eth0 s3-eth4:s2-eth4
```

Fig. 5 Example of SDN topology definition

```
/* receive packets from interface eth0 */
FromDevice(eth0, SNIFFER false, PROMISC true, BURST 8)
   -> pkt :: Classifier(12/0800, -) /* inspect IPv4 packets */
   -> ck :: CheckIPHeader(OFFSET 14) /* pointer to the begin of the IPv4 header */
   /* list of firewall rules, within IPFilter click element */
   -> IPFilter (allow icmp,
                allow tcp and (src or dst port 8000),
                allow tcp and (src or dst port 5001),
                allow udp and (src or dst port 5001),
                allow tcp and (src or dst port 5201),
                allow udp and (src or dst port 11111),
                allow tcp and (src or dst port 8080),
                drop all)
   /* define a thread-safe FIFO queue with buffer size of 10k packets */
   -> queue :: ThreadSafeQueue(10000)
   pkt[1] -> queue
   ck[1] -> queue
   /* output packets to interface eth1 */
   -> ToDevice(eth1, BURST 8);
```

Fig. 6 An example of network function (firewall) written in click. The firewall rules (written in tcpdump-like format) could be either embedded in the click code (as in the example above, for clarity and simplicity) or made available in a separate configuration file (for modularity)

advantages of using click is modularity. Note in Fig. 6 that the element that constitutes the firewall corresponds to IPFilter only; the other elements are auxiliary, used by almost all network functions. Thus, in order to change the behavior of a network function, few adjustments are needed; in most cases, replacing the main block suffices.

Monitoring of VMs/N-PoPs is done using sar, available in the GNU/Linux sysstat package. sar shows textual info about CPU, memory and I/O data, among others. That textual info is passed on to a python parser and then to the Metric Collector module. We chose this architectural deployment as is requires no changes to other modules in case one wishes to replace sar.

We also use Open vSwitch for materializing the SDN network. In addition to being open source, Open vSwitch supports OpenFlow and provides stable releases with a set of tools that makes it possible not only to create switches and links between them and end hosts. Finally, we use KVM for virtualization, i.e. for creating VMs that run network functions.

6 Evaluation

We carried out a systematic evaluation process to assess the efficacy and effectiveness of NFV-PEAR. The experiments were carried out in a machine with four Intel i5 2.6 GHz processors, 8 GB of RAM, running Ubuntu/Linux Server 11.10 x86_64 operating system.

6.1 Experiment workload and setup

We adopted a strategy similar to that employed in previous work [13] to carry out the experiments. The physical infrastructure was generated with Brite[8] using the Barabasi-Albert (BA-2) model [22]. That model has topological characteristics similar to infrastructures typical of ISPs (Internet Service Providers). The physical infrastructures considered contain a total of 50 N-PoPs, each with a computing power capacity of 100%. On average, each network has 300 links with uniform bandwidth capacity of 10 Gbps and average delay of 10 ms. The N-PoPs are placed at various distinct locations in the network.

Two types of VNF images were available for instantiation. For each type of VNF, the availability of small and large computational capacities (considering, respectively, 25% and 100% of the N-PoP computing power) was assumed. For our evaluation, 20 SFCs were submitted. The types of VNFs required by SFCs were randomly chosen. Each VNF required between 25% and 50% of the capacity of an image instantiated on an N-PoP (note that these percentages are different from the computational capacity of the NF images, mentioned earlier). The considered SFCs followed an in-line topology, with their endpoints in the physical infrastructure being randomly selected.

Our analysis focused mainly on the quality of the solutions generated by the Optimizer module. In order to assess the model ability to re-design the infrastructure with the minimum of disruptions, we artificially alternate some provisioned SFCs (by increasing flow rates) between normal consumption mode and overload (e.g., during peak hours). In the latter case, re-scheduling is necessary to maintain system performance and stability. The proposed model is compared with that of Luizelli et al. [13]. In that case, when re-planning is needed, all SFCs are resubmitted and provisioned in the infrastructure.

6.2 Number of modifications required in the infrastructure

Figure 7 presents the average number of modifications related to (*i*) repositioning of VNFs (Fig. 7a), (*ii*) reassignment of SFCs to VNFs (Fig. 7b) and (*iii*) rearrangement of SFCs (Fig. 7c), given some variation in traffic demand. The proportion of under dimensioned SFCs (i.e., those with higher than provisioned demands) ranges from 10 to 80% of the total number of SFCs allocated in the infrastructure. In addition, traffic flow demand exceeds the provisioned capacities from 10 to 80% (each scenario is depicted using a distinct curve). For these experiments, the values of α, β, γ, and ϕ (from the ILP model) are considered to be 1. Observe that possible values to these parameters are not bounded by any constraint, since that infrastructure characteristics (like network size and load) might affect the effectiveness of any setting for them. We left for future research an in-depth analysis of the interrelationship among these parameters.

It is observed that the number of changes needed (axis y) to re-adjust the network to the new demand is proportional 1) to the percentage of SFCs with increased demand and 2) to the demand values exceeded (x-axis). In addition, it is observed that the number of changes related to the repositioning of VNFs is substantially lower compared to that observed for reassignment of flows and re-engineering of SFCs. This indicates the feasibility of our model in real environments, since the time required to instantiate (or migrate) a VNF is substantially higher (in the order of milliseconds to seconds) than that of reprogramming a routing device (order of milliseconds), for example. Also, comparing to the baseline, one may observe that our model reduces by 25% the number of changes related to VNF placement and by up to twice the number of changes related to the chaining and reassignments of SFCs to VNFs.

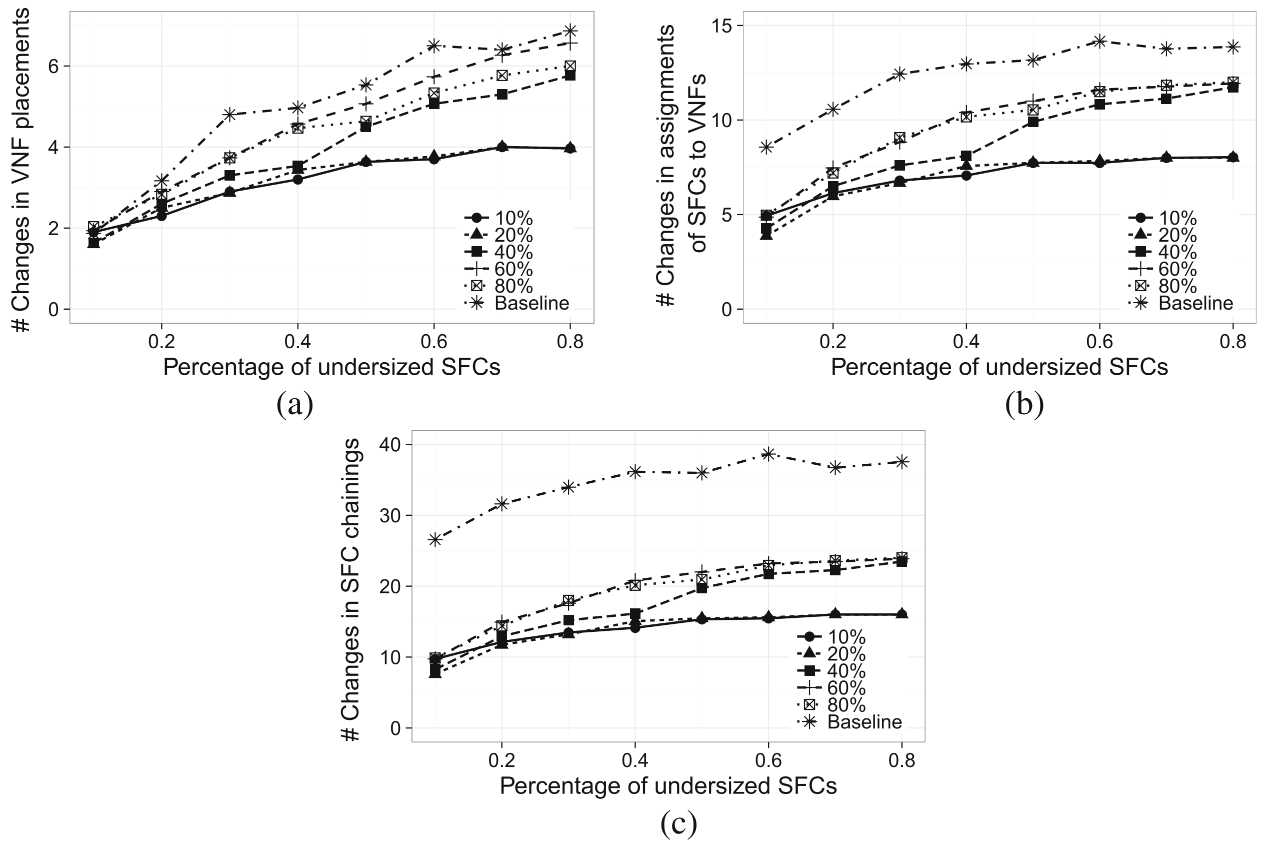

Fig. 7 Analysis of SFC reassignments with traffic flow demand considering a baseline scenario, and exceeding the provisioned bandwidth capacities in 10%, 20%, 40%, 60%, and 80%. (**a**) repositioning of VNFs, (**b**) reassignment of SFCs to VNFs, (**c**) rearrangement of SFCs

6.3 Impact of the over-commitment factor of VNFs on SFC replanning

For this evaluation, around 80% of SFCs initially operate under normal (off-peak) condition, and flow processing demand is then increased by 40%. The over commitment factor ϕ parameter varied between 0 and 40%. Note that higher values for over commitment factor increase the chance that some VNF will exhibit performance degradation. Figure 8 illustrates the number of changes required to re-adjust SFC planning to the new network traffic demand in this scenario. Note that the higher the over commitment factor, the lower the number of changes needed in the infrastructure. For example, with 10% over commitment, there is a 30% reduction in the total number of changes (compared to the scenario with 0% over commitment).

We also evaluated the time needed to re-deploy SFCs. In order to estimate the time necessary to reconfigure the whole infrastructure, we took into account realistic estimates for VNF boot-up time. For VNF boot-up time we considered two values/cases: (*i*) 50 ms, for VNFs implemented as containers; and (*ii*) 1000 ms for VNFs implemented as virtual machines [23, 24]. To account for placement and chaining changes, we also considered that those are related to SDN rule insertion in a forwarding device. For simplicity, we considered that the time required to insert (or modify) a single SDN rule is 10 ms [25].

Our estimates for time required for SFC re-deployment are shown in Fig. 9. As over commitment increases, the overall number of infrastructure changes decreases. Consequently, the time needed to rearrange SFCs is reduced. In the scenario in which VNFs run in virtual machines, re-deployment time is reduced from 7 s (0% over commitment) to 3 s (when 30% over commitment is allowed). Similarly, re-deployment time is reduced from 1 s to 300 ms with VNF running in containers. Although small in scale, these improvements can become substantial in large scale deployments with several ongoing SFCs requests.

6.4 Efficiency in replanning SFC chaining

Figure 10 shows the average time needed to find an optimal solution to the SFC replanning problem. In most experiments, less than 4 s are required to solve the proposed model. Observe that our model is solved faster in comparison to the case in which all SFCs are re-deployed (baseline).

Observe also that with more SFCs in normal operation, more time is required to find the optimal solution. Although the VNF placement and chaining problem is NP-hard, these results suggest that finding exact solutions is feasible in small- and medium-scale scenarios. For larger scale scenarios, additional research is needed to assess computing time bounds.

6.5 Case studies

Next we discuss example case studies designed to obtain a deeper insight on the potentialities of NFV-PEAR. In these cases our goal is to analyze resource elasticity and traffic engineering capabilities in the context of network function management. We thus focus on a case in which

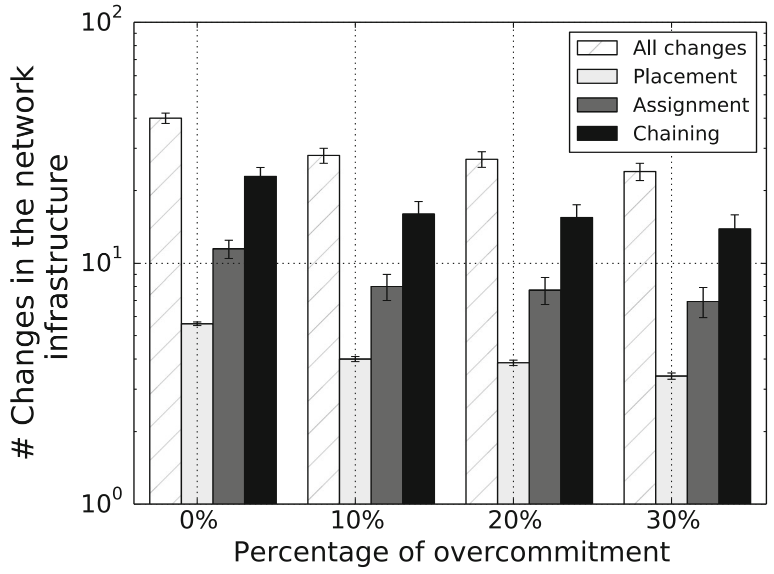

Fig. 8 Impact of over commitment on network replanning

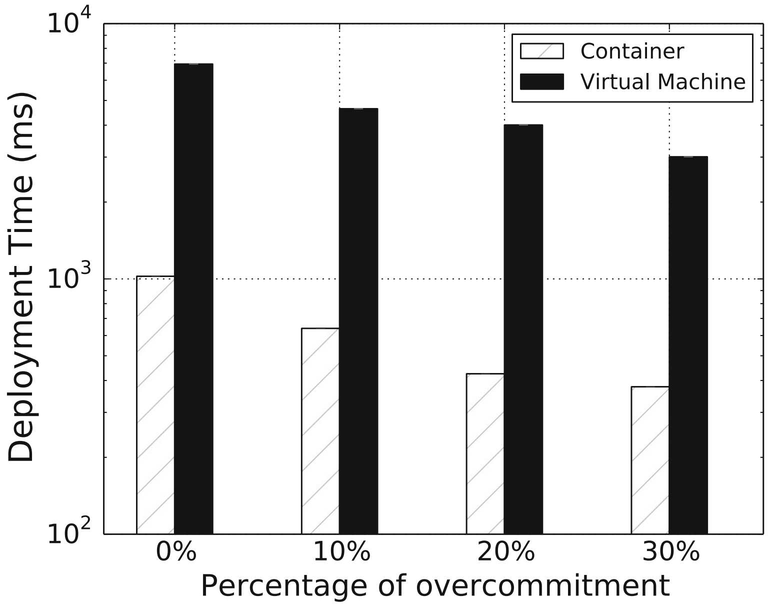

Fig. 9 Impact of varying over commitment factors on deployment time

expansion of network function instances is needed (scale up) as a result of increased demand, and another in which network function migration is required.

6.5.1 Scaling up network functions

In this case study we illustrate (and measure) the impact of scaling up network functions on flow performance. Scale up is a technique used to overcome network function overloading, by deploying another instance of the same function and balancing flows between them. By doing so, network operators can avoid flow degradation while fostering economic resource allocation, as computing power is allocated dynamically in response to fluctuations in demand. We therefore base this scenario on the study from Section 2, which points to occurrence of packet loss events as CPU usage approaches 100%. To investigate NFV-PEAR response to packet losses, we considered the environment shown in Fig. 11; it consists of 1 edge router, 2 virtual switches, and 1 N-PoP. This topology was instantiated on a same server, according to the infrastructure described in detail in Section 2.

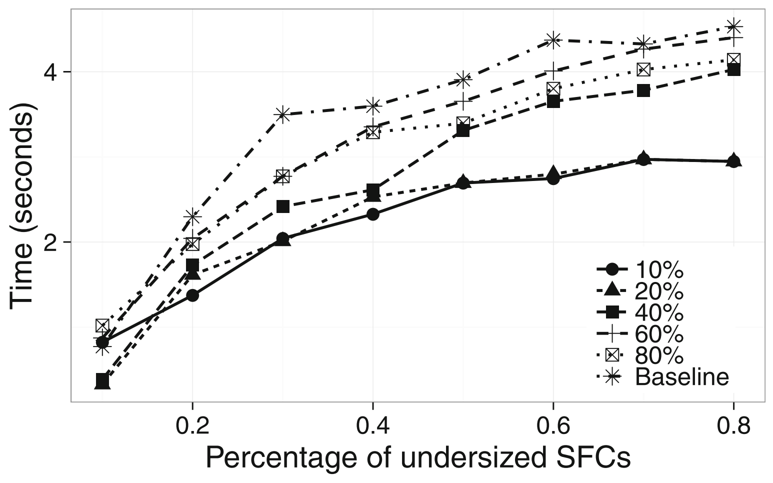

Fig. 10 Average time to find an optimal SFC rechaining

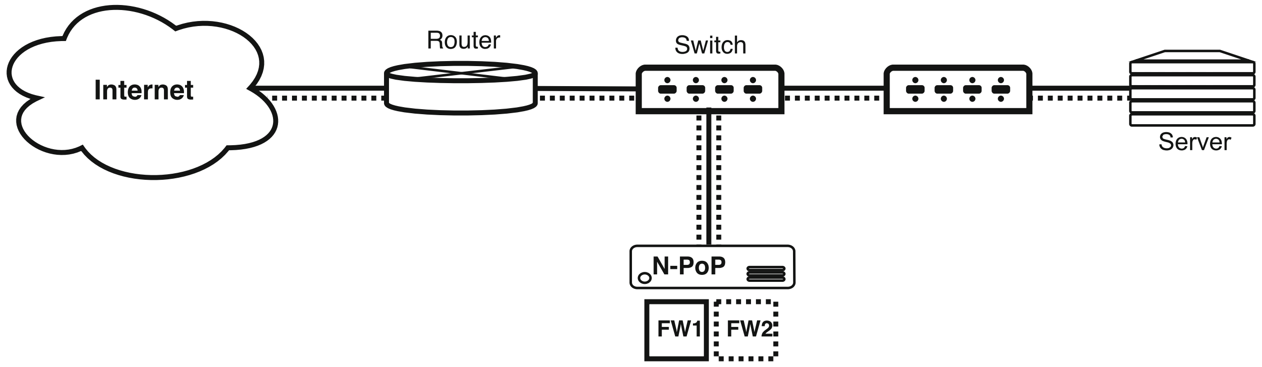

Fig. 11 Infrastructure considered in the NF scale-up case study

Our experiment consisted of a download request coming from the Internet to an internal server, and subsequent data transfer over 5 min. At instants $t = 100$ and $t = 200$ s, one additional transfer was initiated (two in total), which thus increased CPU usage at the N-PoP hosting FW1 (a firewall). As previously described, NFV-PEAR constantly gather network metrics (through the Metric Collector module) that are then used as input to the Optmizer. In this example, CPU metrics are used as input to Equation sets (1) and (2). Figure 12 shows collected results, based on the topology illustrated in Fig. 11 when model parameter ϕ is set to 0.8. Between 0s and 200s only FW1 is running; the network operated without packet loss in that period and, therefore, NFV-PEAR does not trigger the Optmizer module. At instant $t = 200$ network usage increased (because of the additional flow initiated), which caused an increase in CPU usage beyond the predefined threshold. At this point, Equation set (2) is violated and thus the Optmizer module is initiated by NFV-PEAR. Figure 12a shows this behavior from two perspectives. In the first one, the dotted line indicates the CPU usage of FW1 without scale up, showing that CPU consumption would exceed 80%. In the second perspective, the solid black line corresponds to the stabilized CPU usage at FW1, along with the red solid line, which corresponds to the CPU usage at FW2. In the second case, flow processing capacity was scaled up. Through the metrics collected, NFV-PEAR suggested that a new function instance — FW2, dotted in Fig. 11 — had to be instantiated to avoid flow degradation. In the case FW2 was not running, there was a 10s boot delay and thus some packet loss.

6.5.2 *Migrating network functions*

Our goal with this case study was to assess the effectiveness of NFV-PEAR in migrating network functions

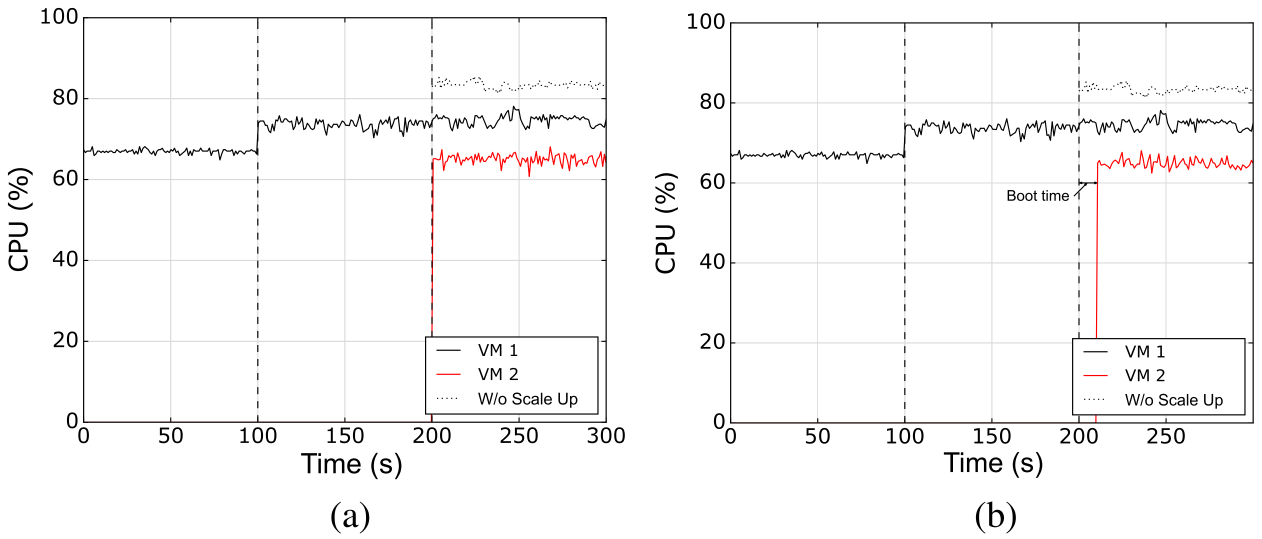

Fig. 12 CPU usage over time with function scale up. **a** Active VMs. **b** VM 1 active - VM 2 off

between N-PoPs. Note that function migration, i.e. transfering the instance of a network function to another N-PoP, can be very useful to improve the function performance (by running it on a more powerful N-PoP) or to reroute a flow (and associated network functions) to avoid a congested path. To evaluate NFV-PEAR with regard to function migration, we considered the environment shown in Fig. 13. It consists of 4 Open vSwitches, 2 N-PoPs and 3 end hosts. This infrastructure was instantiated in the same environment described in Section 2.

The infrastructure setting in the first 200 s is illustrated in Fig. 13a; the first 100 s without data transfer, and the next 100s with a flow from H1 to H3, passing through the FW (firewall) function deployed at N-PoP 1. Figure 13b shows the network after the first 200 s: the flow from H1 to H3 ceased, and another flow between H2 and H3 began, and continued until $t = 300$ s; for this reason, the FW instance was migrated to N-PoP 2.

CPU usage from the case study is shown in Fig. 14. In the first 100 s, CPU usage at N-PoP 1 (Fig. 14a) is close to 0%. At $t = 100$ s, as the data transfer between H1 and H3 initiates, CPU usage of N-PoP 1 approaches 80%. In the meantime (between $t = 0$ and $t = 200$) N-PoP 2 is basically idle (Fig. 14b), as it is not hosting any function. From $t = 200$ s, the flow between H1 and H3 ceased, and a flow between H2 and H3 began, as discussed earlier. At this point, NFV-PEAR reorganizes the network so as the flow between H2 and H3 is steered through N-PoP 2. In doing so, the path length between H2 and H3 (captured by the model in Equations set (7) and (8)), is reduced by 20% in comparison to the previous path through N-PoP 1. Then, the Planner module suggests to migrate FW to N-PoP 2 and assigns (and routes) flows H2 and H3 to the instantiated FW (Equations set (1), (2), and (3)). As a consequence of the FW migration, CPU usage at N-PoP 2 reaches almost 80%.

Note that we opted for a simplified approach for network function migration in this case study. For this reason, performance impact related to function migration is not considered, as different approaches for function migration have diverse impacts on performance. For example, complete function migration (including function image, data, and context) may cause unexpected network congestion, in contrast to an approach in which function image and data is shared beforehand among all N-PoPs (at the cost of extra storage required), and only context info is migrated.

7 Related work

NFV research can be organized considering several perspectives. In the context of SFC deployment planning (i.e., VNF placement and chaining), several investigations merit attention, in special the ones from Bari et al. [3] and Luizelli et al. [7, 13]. Bari et al. [3] describe the orchestration problem of VNFs, which consists in determining the number of VNFs and their locations in the network so that operating costs are optimal. The authors formulate the problem through the linear system (Integer Linear Programming), and use CPLEX and dynamic programming to optimize allocations in smaller scale NFV environments. More recently, Luizelli et al. [7] addressed the problem for large-scale environments, by proposing an optimization algorithm that combines mathematical programming and search meta-heuristics.

In the field of orchestration (post placement & chaining), one of the most notorious efforts is OPNFV [4]. This open source platform aims to foster interoperability between NFV enabling technologies (e.g., Open vSwitch, KVM and Xen) with the other layers of the architecture proposed by ETSI [14] (for orchestration and monitoring). In parallel, several other platforms have been proposed to overcome specific gaps in the orchestration of VNFs, for example ClickOS [23], Slick [26], OpenNetVM [5],

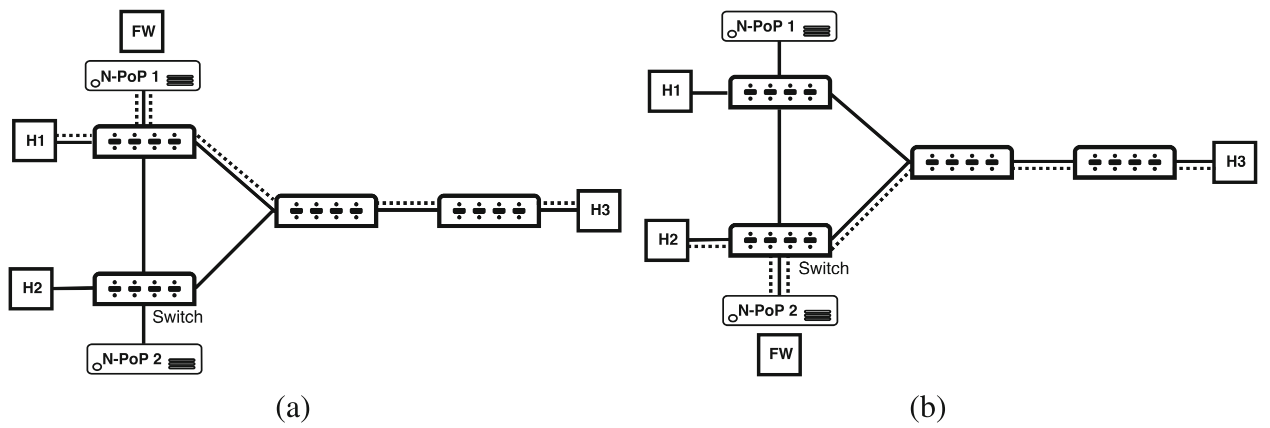

Fig. 13 Network setting considered in the NF migration case study. **a** From $t = 0$s to $t = 200$s. **b** From $t = 200$s to $t = 300$s

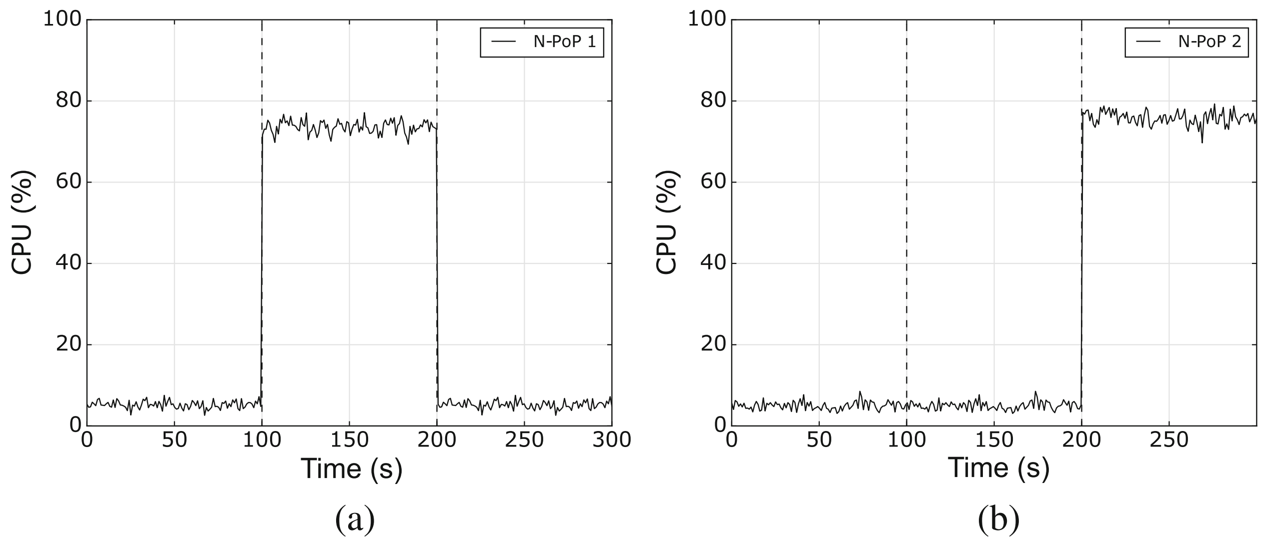

Fig. 14 CPU usage over time with function migration. **a** CPU usage over time for N-PoP 1. **b** CPU usage over time for N-PoP 2

and VirthPhy [27]. Clickos, based on the Xen hypervisor and using virtual functions written in Click [12], aims at reducing packet copy overhead between interfaces, allowing to achieve near-link throughput. OpenNetVM, in turn, introduces a virtual routing layer to integrate the lightweight Docker virtualization engine into the Intel DPDK packet acceleration library. Slick provides a framework for programming network functions as a single high-level control program. Finally, VirtPhy presents a programmable platform for small data centers in which both the functions and the network elements that interconnect them are virtualized.

There are also investigations addressing specific aspetcs of VNF orchestration, such as reliability and QoS performance during flow steering [28], adaptive path provisioning in dynamic service chaining in response to congestion events [29], and dynamic adaptation of VNF orchestration with high availability for 5G applications [30]. Another recent trend in NFV is offloading part of the components that form a VNF to run directly from forwarding devices [31], in an approach similar to OpenBox [16]. Although offloading may significantly save computing resources in this context, such benefit could possibly come at the expense of more complex orchestration procedures.

In the area of VNF performance monitoring, one of the main initiatives is NFV-VITAL [32]. In that paper, the authors propose a framework to characterize the performance of VNFs running in cloud environments. From this characterization, it is possible (*i*) to estimate the best allocation of computational resources to execute VNFs, and (*ii*) to determine the impact of different virtualization and hardware configurations on the performance of VNFs. Another initiative is NFVPerf [33], a tool for bottleneck detection in NFV environments. By analyzing the data flows that transit between VNFs, it makes it possible to calculate average throughput and delays, and thus possible to detect performance degradations.

Despite the observed advances, existing solutions do not address localized fluctuation and bottleneck scenarios that occur due to variations in the volume of flows in transit in the network. An ad hoc strategy to deal with these fluctuations is to re-execute VNF allocation algorithms, and to rearrange them according to the results obtained. Although effective, this strategy is computationally more expensive (by re-executing globally the optimization algorithms), and does not allow to react efficiently to dynamic flow behavior. The work of Rankothge et al. [34, 35] is the closest approach to an effective solution to this problem. In that work, the authors use genetic algorithms to introduce network functions with scalable processing capabilities. However, those network functions are considered in isolation, therefore without taking into account possible global optimizations, such as steering flows with similar requirements for higher capacity VNFs.

Considering the limitations discussed above, NFV-PEAR presents itself as a solution to re-adjust the network against demand variations, through the identification of bottlenecks in the processing of flows, reorganization of the placement and chaining of network functions locally/globally, and aiming at the minimization of disruption in the processing of transit flows.

8 Final considerations

In this work, NFV-PEAR — a framework for adaptive orchestration of network functions in NFV environments — was proposed. The contributions of this paper unfold in (*i*) a formal model to ensure the best provision of SFCs against dynamic changes in demand and/or costs associated with network equipment, (*ii*) a reference architecture for (re)planning and deploying SFCs, agnostic to virtualization and infrastructure technologies, and (*iii*) a preliminary analysis on a subset of metrics to represent performance indicators for VNF operation.

We provided a formal model for adaptive (re)planning of virtual network functions, along with an analytical evaluation. The results showed that our model contributes significantly to a reduction in the number of changes in the physical infrastructure (up to 25% in the repositioning of VNFs and over 200% in the re-routing of network functions). From two distinct case studies, we also assessed the feasibility of NFV-PEAR in bringing resource elasticity and traffic engineering capabilities to the NFV realm.

As prospect for future work, we intend to extend our evaluation to identify correlations in the valuation of the parameters of the model (especially α, β, and γ) in the quality of the solutions obtained. Finally, we aim at developing and integrating methods of traffic demand prediction into our ILP model.

Endnotes

[1] An endpoint represents the source or destination of a traffic flow.

[2] By *virtual path* we mean a path from a source to a destination endpoint in an SFC. To illustrate, suppose an SFC containing three endpoints A, B, and C, and one function for load balancing; endpoint A is linked to the load balancer, which in turn is linked to endpoints B and C. That SFC has two virtual paths: one from A to the load balancer then B, and another one from A to the load balancer then C.

[3] NFV-PEAR code is available for download at https://github.com/nfvpear/.

[4] See https://github.com/nfvpear/nfvpear/blob/master/nfvsdnapi.py for details.

[5] https://www.ibm.com/products/ilog-cplex-optimization-studio

[6] For further details see https://osrg.github.io/ryu-book/en/html/rest_api.html

[7] Further details about the objects used in the click implementation can be found at https://github.com/kohler/click/wiki/Elements

[8] http://www.cs.bu.edu/brite/

Acknowledgements
Not applicable.

Funding
This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) - Finance Code 001.

Availability of data and materials
Please contact authors for data requests.

Authors' contributions
GM collaborated to analyze the impact of network load on VNF performance, participated in the design of the conceptual solution, participated on the proposal of the NFV-PEAR API, coded a proof-of-concept implementation, and ran evaluation experiments. ML collaborated to analyze the impact of network load on VNF performance, participated in the design of the conceptual solution, participated in the design of the improved version of the optimization model, participated on the proposal of the NFV-PEAR API, and ran evaluation experiments and analyzed results obtained from experimental evaluations. WC participated in the design of the conceptual solution, participated in the design of the improved version of the optimization model, participated on the proposal of the NFV-PEAR API, and analyzed results obtained from experimental evaluations. LG participated in the design of the conceptual solution, and analyzed results obtained from experimental evaluations. All authors wrote, read, and approved the final manuscript.

Authors' information
Gustavo Miotto has a M.Sc. degree in Computer Science from Federal University of Rio Grande do Sul (UFRGS). His research interests are focused on Network Function Virtualization and Software Defined Networks.
Marcelo Caggiani Luizelli is an Assistant Professor at Federal University of Pampa (UNIPAMPA), Brazil. He holds a Ph.D. degree in Computer Science from UFRGS (2017). Recently, Marcelo was a visiting research at CS Department of Technion University and NOKIA Bell Labs (Israel) under the supervision of Prof. Danny Raz. His current research interests are Computer Networks, Algorithms, and Optimization, focusing on Network Function Virtualization, Software Defined Networks and Programmable Data Planes. More information about him can be found at http://porteiras.s.unipampa.edu.br/marceloluizelli/.
Weverton Luis da Costa Cordeiro is an Assistant Professor at INF/UFRGS (since 2017), and CNPq-Brazil Research Fellow (PQ-2). He holds a PhD degree in Computer Science (UFRGS, 2014). His research is broadly focused on software defined networking, network function virtualization, programmable forwarding planes, and network security. He recently began working with fault tolerance in large-scale experiments, and recommendation systems for e-commerce platforms. He also has an interest on algorithm analysis and design, and software engineering. Prof. Cordeiro has been publishing his work in high-impact journals and conferences, like IEEE TNSM, Elsevier ComNet, Springer JNSM, IEEE/IFIP NOMS & IM, and USENIX. He has a history of participation in several research communities, characterized by: (i) PC and/or OC member service for ACM SIGCOMM, Brazilian SBRC, IFIP/IEEE CNSM and IFIP/IEEE NOMS & IM, and (ii) major awards and distinctions, including Microsoft Research Latin American Ph.D. Fellowship (2011), and invitation for Dagstuhl Seminar (2014). He works as Editor-in-Chief of Brazilian Electronic Journal of Scientific Initiation in Computing. He is also Guest Editor of a Special Issue on networking security for Wiley IJNM. He coordinated and/or participated in projects with funding/support from CNPq (Brazil), Microsoft Research (USA), National Science Foundation (USA), and RNP (Brazil). He is actively involved with organization and student training for the ACM International Collegiate Programming Contest (ICPC). He is member of the Brazilian Computer Society (SBC). Web page: http://www.inf.ufrgs.br/~wlccordeiro/.
Luciano Paschoal Gaspary holds a Ph.D. in Computer Science (UFRGS, 2002) and is Deputy Dean and Associate Professor at the Institute of Informatics, UFRGS. From 2008 to 2014, he worked as Director of the National Laboratory on Computer Networks (LARC) and, from 2009 to 2013, was Managing Director of the Brazilian Computer Society (SBC). He is a CNPq Research Fellow (1D) and has been involved in various research areas, mainly computer networks, network/service management and computer system security. He is author of more than 120 full papers published in leading peer-reviewed publications. In 2016, he has been appointed as Associate Managing Editor for the Springer's Journal of Network and Systems Management and Publications Committee member of the IEEE SDN initiative. More information can be found at http://www.inf.ufrgs.br/~paschoal/.

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Author details
[1]Institute of Informatics — Federal University of Rio Grande do Sul, Porto Alegre, Brazil. [2]Federal University of Pampa, Bagé, Brazil.

Received: 2 July 2018 Accepted: 4 January 2019
Published online: 04 February 2019

References

1. Han B, Gopalakrishnan V, Ji L, Lee S. Network function virtualization: Challenges and opportunities for innovations. IEEE Commun Mag. 2015;53(2):90–97.
2. Cohen R, Lewin-Eytan L, Naor JS, Raz D. Near optimal placement of virtual network functions. In: IEEE Conference on Computer Communications. INFOCOM '15; 2015. p. 1346–54.
3. Bari MF, Chowdhury SR, Ahmed R, Boutaba R. On Orchestrating Virtual Network Functions. In: 11th International Conference on Network and Service Management. CNSM '15; 2015. p. 50–56.
4. Open Networking Lab. Open Platform for NFV (OPNFV). 2018. Available at https://www.opnfv.org/. Accessed 29 May 2018.
5. Zhang W, Liu G, Zhang W, Shah N, Lopreiato P, Todeschi G, et al. OpenNetVM: A Platform for High Performance Network Service Chains. In: ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization. HotMiddlebox '16; 2016.
6. McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, et al. OpenFlow: Enabling Innovation in Campus Networks. SIGCOMM Comput Commun Rev. 2008;38(2):69–74.
7. Luizelli MC, Cordeiro W, Buriol LS, Gaspary LP. A fix-and-optimize approach for efficient and large scale virtual network function placement and chaining. Comput Commun. 2017;102:67–77.
8. Kuo TW, Liou BH, Lin JC, Tsai MJ. Deploying Chains of Virtual Network Functions: On the Relation Between Link and Server Usage. In: IEEE International Conference on Computer Communications. INFOCOM '16. San Francisco: Springer; 2016.
9. Luizelli MC, Saar Y, Raz D, Optimizing NFV. Chain Deployment Through Minimizing the Cost of Virtual Switching. Piscataway: IEEE Press; 2018, pp. 1–9.
10. Luizelli MC, Raz D, Sa'ar Y, Yallouz J. The actual cost of software switching for NFV chaining. In: IFIP/IEEE Symposium on Integrated Network and Service Management. IM '17; 2017.
11. Miotto G, Luizelli MC, da Costa Cordeiro WL, Gaspary LP. NFV-PEAR: Posicionamento e Encadeamento Adaptativo de Funcoes Virtuais de Rede. In: Brazilian Symposium on Computer Networks and Distributed Systems. SBRC '17; 2017. p. 1–14.
12. Kohler E, Morris R, Chen B, Jannotti J, Kaashoek MF. The Click modular router. ACM Trans Comput Syst. 2000;18(3):263–97.
13. Luizelli MC, Bays LR, Buriol LS, Barcellos MP, Gaspary LP. Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions. In: IFIP/IEEE Int'l Symposium on Integrated Network Management. IM '15; 2015.
14. ETSI. Network Functions Virtualisation (NFV). 2018. Available at http://www.etsi.org/technologies-clusters/technologies/nfv. Accessed 29 May 2018.
15. Rajagopalan S, Williams D, Jamjoom H, Warfield A. Split/Merge: System Support for Elastic Execution in Virtual Middleboxes. In: USENIX Symposium on Networked Systems Design and Implementation. NSDI '13. USENIX. New York; 2013. p. 227–240.
16. Bremler-Barr A, Harchol Y, Hay D. OpenBox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions. New York: ACM; 2016, pp. 511–24.
17. NetworkX. NetworkX - Software for complex networks. 2018. Available at https://networkx.github.io/. Accessed 2 Feb 2018.
18. Paramiko. Welcome to Paramiko. 2018. Available at http://www.paramiko.org. Accessed 29 May 2018.
19. TinyDB. Welcome to TinyDB. 2018. Available at http://tinydb.readthedocs.io. Accessed 3 Feb 2018.
20. Arskom Ltd. spyne - RPC that doesn't break your back. 2018. vailable at http://spyne.io/. Accessed 3 Feb 2018.
21. Ryu. Ryu SDN Framework. 2018. Available at http://osrg.github.io/ryu/. Accessed 4 Jan 2018.
22. Albert R. Barabási AL. Topology of Evolving Networks: Local Events and Universality. Phys Rev Lett. 2000;85:5234–7.
23. Martins J, Ahmed M, Raiciu C, Olteanu V, Honda M, Bifulco R, et al. ClickOS and the Art of Network Function Virtualization. Seattle: USENIX Association; 2014, pp. 459–73.
24. Cziva R, Jouet S, White KJS, Pezaros DP. Container-based network function virtualization for software-defined networks. In: IEEE Symposium on Computers and Communication. ISCC '15; 2015. p. 415–420.
25. He K, Khalid J, Gember-Jacobson A, Das S, Prakash C, Akella A, et al. Measuring Control Plane Latency in SDN-enabled Switches. In: ACM SIGCOMM Symposium on Software Defined Networking Research. SOSR '15. New York: ACM; 2015. p. 25:1–25:6.
26. Anwer B, Benson T, Feamster N, Levin D. Programming Slick Network Functions. In: ACM SIGCOMM Symposium on Software Defined Networking Research. SOSR '15. New York: ACM; 2015. p. 14:1–14:13.
27. Dominicini CK, Vassoler GL, Ribeiro MR, Martinello M. VirtPhy: A Fully Programmable Infrastructure for Efficient NFV in Small Data Centers. In: IEEE Conference on Network Function Virtualization and Software Defined Network. NFV-SDN '16; 2016.
28. Gharbaoui M, Fichera S, Castoldi P, Martini B. Network orchestrator for QoS-enabled service function chaining in reliable NFV/SDN infrastructure. In: IEEE Conference on Network Softwarization. NetSoft '17; 2017. p. 1–5.
29. Mohammed AA, Gharbaoui M, Martini B, Paganelli F, Castoldi P. SDN controller for network-aware adaptive orchestration in dynamic service chaining. In: IEEE NetSoft Conference and Workshops. NetSoft '16; 2016. p. 126–130.
30. Martini B, Gharbaoui M, Fichera S, Castoldi P. Network orchestration in reliable 5G/NFV/SDN infrastructures. In: Int'l Conference on Transparent Optical Networks. ICTON '17; 2017. p. 1–5.
31. Cordeiro W, Marques JA, Gaspary LP. Data Plane Programmability Beyond OpenFlow: Opportunities and Challenges for Network and Service Operations and Management. J Netw Syst Manag. 2017;1:47–53.
32. Cao L, Sharma P, Fahmy S, Saxena V. NFV-VITAL: A framework for characterizing the performance of virtual network functions. In: IEEE Conference on Network Function Virtualization and Software Defined Network. NFV-SDN '15; 2015. p. 93–99.
33. Naik P, Shaw DK, Vutukuru M. NFVPerf: Online Performance Monitoring and Bottleneck Detection for NFV. In: IEEE Conference on Network Function Virtualization and Software Defined Network. NFV-SDN '16; 2016.
34. Rankothge W, Le F, Russo A, Lobo J. Experimental Results on the use of Genetic Algorithms for Scaling Virtualized Network Functions. In: Network Function Virtualization and Software Defined Network. NFV-SDN '15; 2015.
35. Rankothge W, Le F, Russo A, Lobo J. Optimizing Resource Allocation for Virtualized Network Functions in a Cloud Center Using Genetic Algorithms. IEEE Trans Netw Serv Manag. 2017;14(2):343–56.