Impact of Processing-Resource Sharing on the Placement of Chained Virtual Network Functions

Marco Savi, Massimo Tornatore, Giacomo Verticale

***Abstract*—Network Function Virtualization (NFV) provides higher flexibility for network operators and reduces the complexity in network service deployment. Using NFV, Virtual Network Functions (VNF) can be located in various network nodes and chained together in a Service Function Chain (SFC) to provide a specific service. Consolidating multiple VNFs in a smaller number of locations would allow decreasing capital expenditures. However, excessive consolidation of VNFs might cause additional latency penalties due to processing-resource sharing, and this is undesirable, as SFCs are bounded by service-specific latency requirements. In this paper, we identify two different types of penalties (referred as "costs") related to the processing-resource sharing among multiple VNFs: the *context switching costs* and the *upscaling costs*. Context switching costs arise when multiple CPU processes (e.g., supporting different VNFs) share the same CPU and thus repeated loading/saving of their context is required. Upscaling costs are incurred by VNFs requiring multi-core implementations, since they suffer a penalty due to the load-balancing needs among CPU cores. These costs affect how the chained VNFs are placed in the network to meet the performance requirement of the SFCs. We evaluate their impact while considering SFCs with different bandwidth and latency requirements in a scenario of VNF consolidation.**

***Index Terms*—Network Function Virtualization, Service Function Chaining, Processing-Resource Sharing, Context Switching**

I. Introduction

In the last years, *Network Function Virtualization* (NFV) has emerged as a promising technique to help network operators reduce capital and energy costs. NFV is based on the concept of *network function*, which is an abstract building block representing a piece of software designed to process the network traffic and accomplish a specific task. Examples of network functions are firewalls, network address translators, traffic monitors, or even more complex entities such as 4G/5G service or packet gateways. So far, network functions have been implemented using dedicated hardware referred to as *middleboxes*. Such middleboxes are able to handle heavy traffic loads, but have an expensive and slow provisioning cycle. Additionally, they cannot be easily re-purposed and must be dimensioned at peak loads, leading to waste of resources when the traffic is low, e.g., in off-peak hours. The NFV paradigm consists in moving from a hardware to a software implementation of network functions in a virtualized environment. This way, multiple and heterogeneous *virtual network functions* (VNFs) can be hosted by the same generic commercial-off-the shelf (COTS) hardware. NFV adds flexibility to the network since it allows network operators to efficiently consolidate the VNFs and makes it possible on-the-fly provisioning. Another value added by NFV is the simplicity in the deployment of heterogeneous network services. NFV exploits the concept of *service function chaining* [2], according to which a service (e.g., web browsing, VoIP, etc.) can be provided by one or more service function chains (SFCs), i.e., a concatenation of appropriate VNFs that must be crossed by the traffic associated to that specific service. The main weakness of NFV is the hard-to-predict performance due to resource sharing of hardware among different functions, especially concerning the *processing* [3].

In this paper, we evaluate the impact of processing-resource sharing on the placement of VNFs and on the embedding of SFCs in a VNF consolidation scenario, i.e., when we want to minimize the amount of COTS hardware deployed in the network. We identify two sources of inefficiency and performance degradation. The first, which we will refer to as *context switching costs*, stems from the need of sharing Central Processing Unit (CPU) resources among different VNFs and results in additional latency, since packets *(i)* must wait for the correct VNF to be scheduled and *(ii)* some CPU time is wasted due to the need of saving/loading the state of the VNFs at each scheduling period. The second, which we will refer to as *upscaling costs*, represents the additional latency and processing cost *(i)* of balancing network traffic among multiple CPU cores in multi-core architectures and *(ii)* of keeping the shared state synchronized among the NFV instances running over different cores.

Such performance degradation affects how the VNFs must be placed in the network to guarantee the requirements for different types of SFCs. Specifically, we propose a novel detailed node model that takes into account the aforementioned processing-resource sharing costs. Compared to existing models where resource-sharing penalties are not considered (e.g. [4]), our model enables a more accurate distribution and scaling of VNFs, preventing excessive VNF consolidation when it might jeopardize SFC performance. To the best of our knowledge, this paper is the first study evaluating the impact of such processing-resource sharing costs on service function chaining in an NFV scenario.

The remainder of the paper is organized as follows. Section II discusses related work concerning both processing-resource sharing and VNF placement. In Section III we introduce our system model by modeling the physical network, the VNFs, the SFCs and the processing-resource sharing costs. Section IV introduces the problem of VNF consolidation. We formulate an Integer Linear Programming (ILP) model and

Marco Savi is with Fondazione Bruno Kessler, CREATE-NET Research Center, 38123, Trento, Italy (email: m.savi@fbk.eu).

Massimo Tornatore and Giacomo Verticale are with the Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, 20133, Milan, Italy.

A preliminary version of this paper appeared in [1].

propose a heuristic algorithm to solve the defined problem in a scalable way. In Section V we show illustrative numerical results over a realistic network scenario. We first compare results obtained by solving the ILP model and by running our heuristic algorithm, then move our focus to the embedding of a more diverse set of SFCs and to the comparison with the state of the art. Finally, Section VI draws the conclusion of our work.

II. Related work

Our work is related both to processing-resource sharing in multi-core architectures and to VNF placement/SFC embedding. In the next two subsections we recall the related work with respect to such topics.

A. Processing-resource sharing

Several studies in literature have investigated the challenges arising from processing-resource sharing. Refs. [5][6][7] were among the first works investigating processing-resource sharing challenges due to the adoption of multi-core architectures. Refs. [5][6] survey the architectural upgrades needed to efficiently scale processing performance by adopting multi-core technologies. Ref. [7] argues that, even if the adoption of multi-core systems is the dominant trend, network devices hardly fully exploit multiple cores. Among the challenges related to multi-core systems, *load balancing* is one of the most complex. For example, Ref. [8] investigates how load balancing, by adding a new layer in the system architecture that can become a bottleneck, must be carefully designed and could lead to performance penalties, while Ref. [9] defines a novel adaptive traffic distribution among the CPU cores on a per-packet basis trying to mitigate such issue. As we will better describe later, we call *upscaling costs* the performance degradation due to load balancing.

Due to processing-resource sharing, another issue arises related to a well-known operation performed by processors, called *context switching*. Context switching has been thoroughly investigated in literature and it is related to the need of saving/loading the context (i.e., the state) of a CPU process to enable the execution of multiple CPU processes on a single CPU. Ref. [10] defines a methodology to quantify the costs related to context switching in terms of latency, while Ref. [11] investigates context switching costs due to cache interference among multiple CPU processes. Additionally, Ref. [12] focuses on analogous issues but related to accelerated services provided by Graphics Processing Units (GPUs). Refs. [13][14][15] have instead investigated the impact of context switching on NFV. Specifically, Ref. [13] defines some strategies to reduce the context switching costs, while Refs. [14][15] design and implement algorithms for efficient sharing of processing resources among VNFs, which are considered as specific types of CPU processes. Finally, Ref. [16] presents a latency-aware NFV scheme where software middleboxes can be dynamically scheduled according to the changing traffic and resources, which affect latency in the network due to processing-resource sharing. In our paper we also deal with upscaling and context switching costs, but our goal is evaluating their impact on VNF placement and SFC embedding.

B. VNF placement and SFC embedding

We formalize an optimization problem for VNF placement and SFC embedding that can be seen as an extension of some well-known Virtual Network Embedding (VNE) problems, as the ones shown in Refs. [17][18][19]. In our problem, the SFCs are the virtual networks and the chained VNFs are virtual nodes, which are connected by virtual links and must be crossed in sequential order. Such SFCs must be embedded in a physical network, where each virtual link can be mapped to a physical path [17], multiple virtual nodes can be mapped to the same physical node [18], and virtual nodes must be consolidated [19]. In our SFC embedding problem it must be also guaranteed that a virtual node can be shared among multiple virtual networks. From another point of view, the SFCs can be seen as *walks* on the physical graph. Ref. [20] is the first work investigating the optimal embedding of SFCs in the network following a VNE approach. The authors formulate a Mixed Integer Quadratically Constrained Problem to evaluate the optimal placement of VNFs. In our paper, we formulate a similar problem, but we extend the analysis to cover also processing-resource sharing aspects.

Some other studies have dealt with the placement of VNFs in the network. Refs. [21][22][23][24][25][26][27][28][29][30] all formulate an ILP model to solve the problem of optimal VNF placement and/or SFC embedding, considering different objective functions. Ref. [21] minimizes the used servers, Refs. [22][24][28] the maximum utilization of links, Ref. [25] the OPEX and resource utilization, Ref. [26] the network load, Ref. [27] the VNF mapping cost, Ref. [23] the overall consumed bandwidth and Ref. [29] the traffic delivery cost. Instead, Ref. [30] maximizes the throughput. Except for Refs. [21][22][23], all the other works also present a heuristic algorithm to improve the scalability of the model. Concerning about heuristic algorithms for VNF placement, Ref. [31] proposes four genetic algorithms for network load balancing, while Ref. [32] aims at minimizing the SFC embedding cost. In our work the objective is different, as we want to maximally consolidate the VNFs as Ref. [21], by taking into account processing-resource sharing, which adds more practical flavor to the solution of the problem.

While solving an ILP allows to obtain a solution for a static placement of VNFs, some other papers deal with the definition and implementation of online algorithms for an on-the-fly and dynamic deployment. Specifically, Refs. [33][34] implement dynamic VNF chaining by means of OpenFlow [33] and OpenStack [34]. Refs. [35][36][37] define some algorithms for online VNF scheduling and placement: while Refs. [35][36] take into account service function chaining aspects, Ref. [37] does not. Refs. [38][39][40][41] focus on the definition of algorithms for online embedding of SFCs while minimizing the resources consumed by the infrastructure to embed each SFC request. Unlike the other works, Ref. [41] also leverages an online learning method for service demand prediction and proactive VNF provisioning. Finally, Ref. [30] designs an online algorithm with proven competitive ratio with respect to the objective of maximizing throughput. Even though in our paper we do not design and implement any online algorithm

for SFC embedding, we define a heuristic algorithm for VNF consolidation that embeds the SFC one-by-one and that can be used as a basic engine for any online algorithm.

III. System Model

Our model of NFV-enabled network comprises representations of: an Internet Service Provider (ISP) network with physical links and nodes, a set of edge-to-edge[1] services implemented as a chain of VNFs, a set of atomic VNFs that can be deployed in an NFV-capable physical node. Our model of NFV nodes is in line with current trends towards the re-architecture of core nodes as *micro-datacenter* able to host several VNFs [42]: such NFV nodes have a limited amount of processing power, which is split over multiple computational cores. To achieve parallelism, any VNFs must adopt load balancing mechanisms, resulting in the employment of CPU time and in additional latency to traverse the node. Furthermore, multiple VNFs can be involved in computational power contention, resulting in the employment of CPU time for VNF coordination and in additional latency for the packets that must wait the scheduling of their reference VNF.

A. Physical network and SFC/VNF modeling

1) Physical network: We model the physical network as a connected directed graph $\mathcal{G} = (V, E)$. All the network nodes $v \in V$ have basic packet forwarding capabilities. The links $(v, v') \in E$ have capacity $\beta_{v,v'}$ and latency $\lambda_{v,v'}$. A subset of the network nodes v can be also equipped with hardware capable of executing VNFs. The model is agnostic with respect to the physical location of such nodes, which can be cabinets, central offices, core exchanges, etc. We generally refer to this nodes as *NFV nodes* (see Fig. 1). Every NFV node is connected to an ideal zero-latency ($\lambda_{v,v} = 0$) and infinite-bandwidth ($\beta_{v,v} = \infty$) self-loop $(v, v) \in E$. Its use will be discussed later in the paper. An NFV node is also equipped with a *multi-core CPU*. We measure its processing capacity γ_v, expressed in terms of number of CPU cores that the multi-core CPU supports. If a physical node has only forwarding capabilities (i.e., it is a forwarding-only node, such as a legacy router or a switch), it follows that $\gamma_v = 0$. In the rest of the paper we will assume a one-to-one correspondence between an NFV node and a multi-core CPU, and we will interchangeably use the terms *multi-core CPU* and *NFV node*.

2) Service function chains: When a network operator provisions a *service* between two end-points, it deploys one or more SFCs c in the network. For the sake of simplicity, we consider a one-to-one correspondence between a service and an SFC: thus, the carrier's goal is to embed a set of SFCs C.

A single SFC c can be modeled by a walk $\mathcal{C}^c = (X^c \cup U^c, G^c)$, where X^c is the set of start/end points u, U^c is the set of VNF requests u and G^c is the set of virtual links (u, u') chaining consecutive VNF requests/start/end points u and u'.

[1]In this paper we deal with the embedding of aggregated SFCs between the edges of the core network (i.e., *ISP network*), serving multiple users belonging to the same access/aggregation networks. For this reason, we will use the terms *end-to-end* and *edge-to-edge* interchangeably.

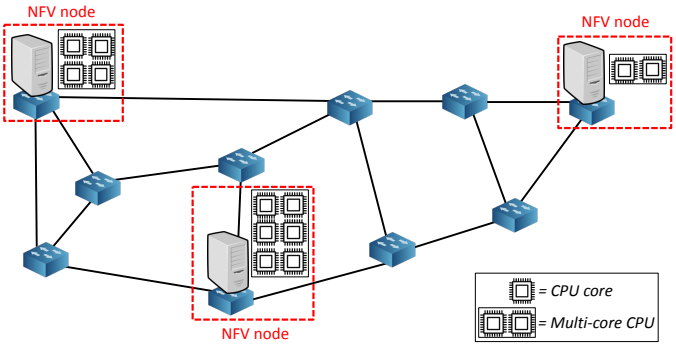

Fig. 1. Physical topology where some nodes are equipped with generic multi-core COTS hardware (i.e., *NFV nodes*)

From a topological perspective, both VNF requests and start/end points are virtual nodes $u \in X^c \cup U^c$. Note that in our model, as done in [20], we decouple the concepts of *VNF* $f \in F$ and of *VNF request* $u \in U^c$. For each SFC c, we have a chain of VNF requests u; each VNF request u is mapped to a specific VNF f through the mapping parameter $\tau_u^c = f \in F$. The input parameter τ_u^c allows us to relate each VNF request u to the specific VNF f it requests (see Fig. 2). Moreover, what we place in the NFV nodes are *VNF instances* of different VNFs f. In this way, multiple VNF requests u for different SFCs c can be mapped to the same VNF instance of a VNF f. Similar considerations can be done for the start/end points $u \in X^c$. Such points are fixed in the network, and we introduce the input mapping parameter $\eta_u^c = v \in V$ to make explicit the mapping between the start/end point u for the SFC c to a specific physical node v (see Fig. 2).

In our model, every SFC c can serve an aggregated *number of users* N^{user}. Such aggregated SFCs are deployed when multiple users require the same service between the same pair of start/end point. Every SFC is then associated to a set of performance constraints:

- The *aggregated requested bandwidth* $\delta_{u,u'}^c$, i.e., the bandwidth that must be guaranteed between two VNF requests/start/end points u and u' to support the service offered by the SFC c for all the users. It follows that $\delta_{u,u'}^c = N^{\text{user}} \delta_{u,u'}^{c,\text{user}}$, where $\delta_{u,u'}^{c,\text{user}}$ is the *requested bandwidth per user* for each virtual link (u, u'). Every virtual link (u, u') can be associated to a different bandwidth requirement since the chained VNFs can lead to a change in the traffic throughput.
- The *maximum tolerated latency* φ^c, i.e., the maximum end-to-end delay that can be introduced by the network without affecting the service between the start/end points of the SFC c.

3) Virtual network functions: A VNF can be seen as a black-box performing some operations on the network traffic. Every VNF f hosted by an NFV node v (i.e., every VNF instance) must be assigned a dedicated amount of processing $c_{f,v}$ per time unit in order to perform all the necessary operations on the input traffic. The processing $c_{f,v}$ is expressed as a number or fraction of the CPU cores γ_v of the NFV node v assigned to the instance of VNF f. For example, $c_{f,v} = 2.5$ means that the instance of VNF f fully consumes, on the NFV node v, the processing resources of two CPU cores

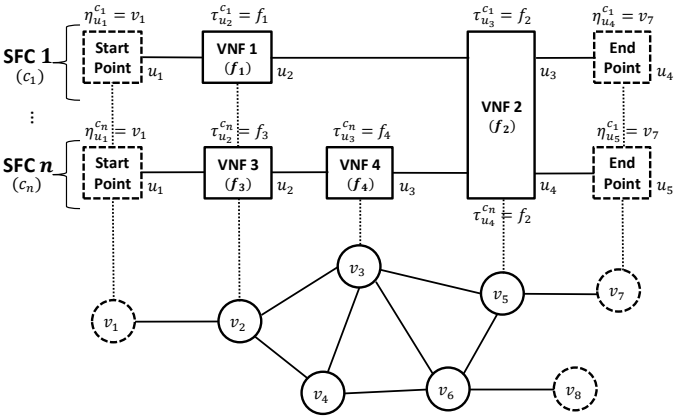

Fig. 2. Example of SFCs that must be embedded in the physical network, where different VNFs can share the processing resources of the same NFV node and multiple SFCs can share the same VNF

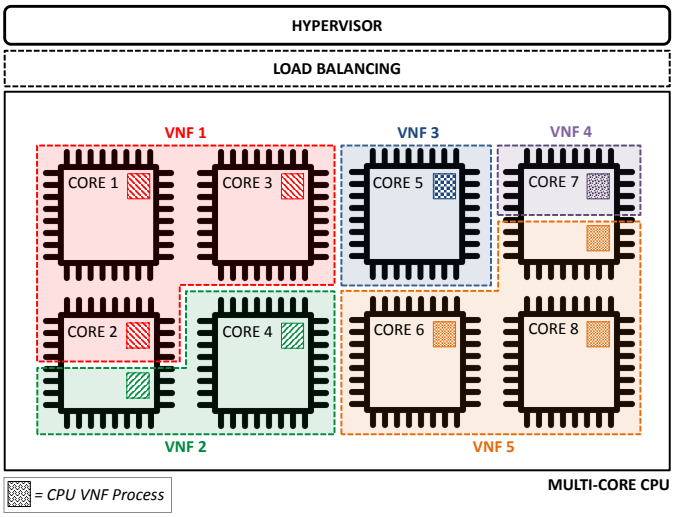

Fig. 3. Pictorial view of CPU sharing among multiple VNFs, responsible for the *context switching costs*

plus half the resources of a third CPU core. We say that a VNF instance has a larger *size* when it is assigned a larger processing capability $c_{f,v}$. We then define π_f as the *processing per VNF request*, also expressed as fraction of CPU cores. π_f indicates the processing resources that must be dedicated to each VNF request u mapped to an instance of VNF f. The ratio $c_{f,v}/\pi_f$ is thus the theoretical maximum number of VNF requests that can share the instance of VNF f hosted by the NFV node v. π_f depends on the number of users N^{user} per each SFC c: in general, we have that $\pi_f = N^{\text{user}}\pi_f^{\text{user}}$, where π_f^{user} is the *processing per user* for the VNF f. Note that different VNFs are characterized by a π_f^{user} that can largely vary from one to each other, depending on the complexity of the performed operations.

B. Modeling of processing-resource sharing costs

As seen in Section II, when multiple VNFs share the computational resources available at the same NFV node, some performance degradation due to processing-resource sharing is expected. As already introduced, we have identified two types of costs: *context switching costs* and *upscaling costs*.

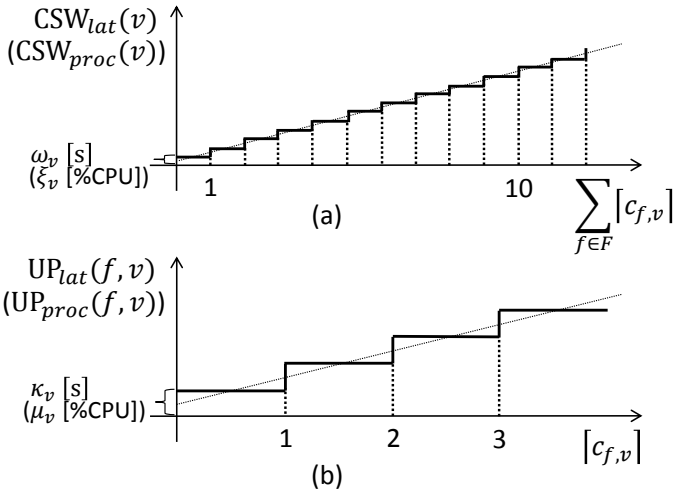

Fig. 4. Modelling of context switching (a) and upscaling (b) costs

1) Context switching costs: In an NFV node v the CPU cores are shared among instances of different VNFs f. The processing resources of a single core can be used by a VNF through a dedicated *process*. Fig. 3 shows an example where the processing resources of an 8-core CPU are shared among six different VNFs. Every time a VNF requires processing resources by multiple cores, a different process used by that VNF must be executed on each core. Having multiple processes sharing multiple cores leads to performance penalties which we will refer to as *context switching* costs. In fact, the CPU needs some time and some dedicated processing capacity to perform the operation of context switching. The degradation effects due to context switching are then an increase in the *latency* introduced by the NFV node (i.e., latency costs) and a reduction of the *actual processing capacity* that can be used to host the VNFs (i.e., processing costs). According to [15], we model the context switching costs as linearly increasing with respect to the overall number of processes sharing the NFV node v, as shown in Fig. 4(a).

We can express such costs in the following way:

$$\text{CSW}_{\text{lat}}(v) = \sum_{f \in F} \lceil c_{f,v} \rceil \omega_v \tag{1}$$

$$\text{CSW}_{\text{proc}}(v) = \sum_{f \in F} \lceil c_{f,v} \rceil \xi_v \tag{2}$$

where $\sum_{f \in F} \lceil c_{f,v} \rceil$ indicates the overall number of processes involved in the context switching operation for the NFV node v, ω_v is the context switching latency parameter (measured in time units) and ξ_v is the context switching processing parameter (measured in number or fractions of CPU cores). Such parameters can vary for different NFV nodes depending on the adopted multi-core CPU technology. If follows that $\text{CSW}_{\text{lat}}(v)$ is a time quantity (e.g. expressed in ms) and $\text{CSW}_{\text{proc}}(v)$ is a fraction of the CPU cores.

2) Upscaling costs: As stated above, an instance of a VNF f placed on an NFV node v can require more than the processing resources provided by a single CPU core. This happens when the VNF must process a high quantity of traffic (e.g., when a high number of users shares such VNF). In this case, the network traffic handled by that VNF must be balanced among multiple CPU cores. Figure 5 shows how

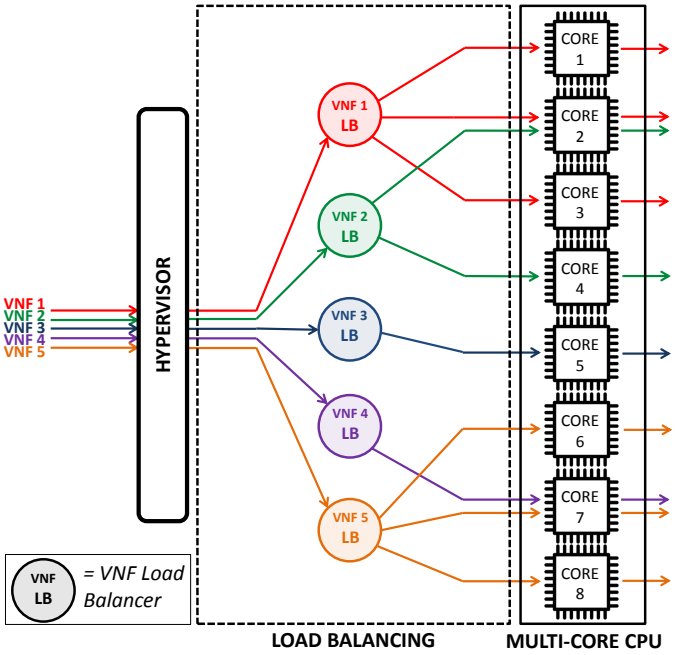

Fig. 5. Pictorial view of the load balancing layer, responsible for the *upscaling costs*

network traffic must be balanced according to the example shown in Fig. 3. The new layer of *load balancing* shown in Fig. 5 is responsible for the *upscaling costs*. In fact, every VNF that is hosted by the NFV node needs a dedicated *load balancer* that takes the decision on how the traffic is sorted among the CPU cores. The load balancer can be itself seen as an auxiliary VNF performing the specific task of balancing traffic among the CPU cores: it thus needs some time to take the balancing decision and some dedicated processing capacity to perform such operation. As for context switching, the overall degradation effects due to load balancing are an increase in the *latency* introduced by the NFV node for the considered VNF (i.e, latency costs) and a reduction of the *actual processing capacity* that can be used to host the VNFs (i.e., processing costs). We model the upscaling costs, both concerning latency and processing, as linearly increasing with respect to the number of CPU cores among which the traffic is balanced, as shown in Fig. 4(b). We can express such costs as:

$$\mathrm{UP}_{\mathrm{lat}}(f,v) = \lceil c_{f,v} \rceil \kappa_v \quad (3)$$
$$\mathrm{UP}_{\mathrm{proc}}(f,v) = \lceil c_{f,v} \rceil \mu_v \quad (4)$$

where $\lceil c_{f,v} \rceil$ indicates the number of cores involved in the load balancing for the instance of VNF f placed in NFV node v. κ_v is the upscaling latency parameter and μ_v is the upscaling processing parameter. They can differ for heterogeneous NFV nodes, depending on how the load balancing layer is implemented. If follows that $\mathrm{UP}_{\mathrm{lat}}(f,v)$ is a time quantity and $\mathrm{UP}_{\mathrm{proc}}(f,v)$ is a fraction of CPU cores.

Note that, in this paper, we focus on how physical processing resources are shared among different VNFs in an ISP network with micro-datacenters, where no horizontal VNF *scale out* is expected (contrarily to what happens in large datacenters [43]). How virtual processing resources (i.e., vCPUs) are mapped to physical resources (i.e., CPU cores) is a task performed by a *Hypervisor*, depicted in Fig. 3 and Fig. 5. Different hypervisors lead to a different mapping among physical and virtual resources. The evaluation of the performance of different hypervisors is outside the scope of this paper, in which we assume that an optimal mapping among physical and virtual resources always occurs.

TABLE I
SUMMARY OF GRAPHS AND SETS CONSIDERED IN THE MODEL

Graph/Set	Description
$\mathcal{G} = (V, E)$	Physical network graph, where V is the set of physical nodes v and E is the set of physical links (v, v') connecting the nodes v and v'
C	Set of the service function chains c that must be embedded in the physical network $\mathcal{G}$
$\mathcal{C}^c = (X^c \cup U^c, G^c)$	Virtual graph for the SFC c, where X^c is the set of fixed start/end point u, U^c is the set of VNF requests u, G^c is the set of virtual links (u, u') connecting the VNF request (or start point) u and the VNF request (or end point) u'
F	Set of VNFs f that can be requested and placed in the network

TABLE II
SUMMARY OF PARAMETERS CONSIDERED IN THE MODEL

Parameter	Domain	Description
τ_u^c	$c \in C$ $u \in U^c$	VNF requested by the VNF request u in the SFC c ($\tau_u^c \in F$)
η_u^c	$c \in C$ $u \in X^c$	Physical node where the start/end point u for the SFC c is mapped ($\eta_u^c \in V$)
γ_v	$v \in V$	Number of the CPU cores (i.e., processing capacity) hosted by the node v
$\beta_{v,v'}$	$(v, v') \in E$	Bandwidth capacity of the physical link (v, v')
$\lambda_{v,v'}$	$(v, v') \in E$	Latency of the physical link (v, v')
ω_v	$v \in V$	Context-switching latency of the node v
ξ_v	$v \in V$	Context-switching processing of the node v
κ_v	$v \in V$	Upscaling latency of the node v
μ_v	$v \in V$	Upscaling processing of the node v
N^{user}		Number of aggregated users per SFC
$\delta_{u,u'}^{c,\mathrm{user}}$	$c \in C$ $(u, u') \in G^c$	Requested bandwidth per user on the virtual link (u, u') of the SFC c
$\delta_{u,u'}^{c}$	$c \in C$ $(u, u') \in G^c$	$= N^{\mathrm{user}} \delta_{u,u'}^{c,\mathrm{user}}$: Requested bandwidth on the virtual link (u, u') per SFC c
φ^c	$c \in C$	Maximum tolerated latency by the SFC c
π_f^{user}	$f \in F$	Fraction of the CPU core (i.e., processing requirement) per user for the VNF f
π_f	$f \in F$	$= N^{\mathrm{user}} \pi_f^{\mathrm{user}}$: Fraction of the CPU processing required by each VNF request u for the VNF f

IV. Solving the Problem of VNF Consolidation

A summary of the sets and parameters defined in Section III and used in this section is reported in Tables I and II. In the following we focus on the problem of *VNF consolidation*: given a physical network topology and some SFCs, we want to decide position and size, in terms of dedicated CPU cores, of the chained VNFs while minimizing the number of *active* NFV nodes (i.e., the nodes hosting at least one VNF) in the network. This optimization problem can be useful for network operators to plan the best placement of the COTS hardware, since the number of active NFV nodes in the network is a measure of the *cost for NFV implementation*: it follows that the cost is minimized when the VNFs are maximally consolidated. We adopt two different approaches for solving the problem of

TABLE III
DECISION VARIABLES FOR THE ILP MODEL

Variable	Domain	Description
$m^c_{u,v} \in \{0,1\}$	$c \in C$ $u \in U^c$ $v \in V$	Binary variable such that $m^c_{u,v} = 1$ iff the VNF request u for the SFC c is mapped to the node v, otherwise $m^c_{u,v} = 0$
$c_{f,v} \in [0, \gamma_v]$	$f \in F$ $v \in V$	Real variable indicating the fraction of the CPU cores in the node v used by the VNF f
$i_{f,v} \in \{0,1\}$	$f \in F$ $v \in V$	Binary variable such that $i_{f,v} = 1$ iff the VNF f is hosted by the node v, otherwise $i_{f,v} = 0$
$e^c_{v,v',x,y,u,u'} \in \{0,1\}$	$c \in C$ $(v,v') \in E$ $x \in V$ $y \in V$ $(u,u') \in G^c$	Binary variable such that $e^c_{v,v',x,y,u,u'} = 1$ iff the physical link (v,v') belongs to the path between the nodes x and y, where the VNF requests u and u' for the SFC c are mapped to, otherwise $e^c_{v,v',x,y,u,u'} = 0$
$a_v \in \{0,1\}$	$v \in V$	Binary variable such that $a_v = 1$ iff the node v is active, otherwise $a_v = 0$

VNF consolidation. First, we define an *ILP model* that, once solved, allows to obtain an optimal solution for it. However, as specified in Section II, we deal with a virtual network embedding problem and it is well-known that virtual network embedding problems are NP-hard [19]. For this reason, we then design a heuristic algorithm called *Heuristic Cost-aware Algorithm (HCA)* that allows to obtain a suboptimal solution in a much shorter time and could also be used for a real-time placement of SFCs.

A. ILP model description

In Table III we have reported the decision variables for our ILP model. The model extends the formulation proposed in [20] to consider, in the SFC embedding evaluation, the NFV node processing capacity and the VNF processing requirements (eq. 9-11). It also additionally includes upscaling and context switching costs in terms of added latency (eq. 25-26) and reduced node processing capacity (eq. 13-14), as modeled in Section III-B. In the following, we describe the objective function and the constraints.

1) Objective function:

$$\min \sum_{v \in V} a_v \tag{5}$$

The objective function simply minimizes the number of active NFV nodes, as already proposed in [20].

The constraints are then grouped in three categories: *request placement*, *routing* and *performance* constraints. The request placement constraints (eq. 6-14) ensure a correct mapping of VNFs to the NFV nodes as well as a correct mapping between VNF requests and VNFs. The routing constraints (eq. 15-23) guarantee a correct mapping of virtual links to physical paths. Finally, the performance constraints (eq. 24-28) are related to performance requirements that must be guaranteed for both physical network and SFCs.

2) Request placement constraints:

$$m^c_{u,\eta^c_u} = 1 \quad c \in C, u \in X^c \tag{6}$$

$$m^c_{u,v} = 0 \quad c \in C, u \in X^c, v \in V : v \neq \eta^c_u \tag{7}$$

$$\sum_{v \in V} m^c_{u,v} = 1 \quad c \in C, u \in U^c \tag{8}$$

$$\sum_{\substack{c \in C \\ u \in U^c : \tau^c_u = f}} m^c_{u,v} \leq \frac{c_{f,v}}{\pi_f} \quad f \in F, v \in V \tag{9}$$

$$c_{f,v} \leq \mathcal{M}_\gamma \; i_{f,v} \quad f \in F, v \in V \tag{10}$$

$$i_{f,v} - c_{f,v} < 1 \quad f \in F, v \in V \tag{11}$$

$$i_{f,v} \leq \sum_{\substack{c \in C \\ u \in U^c : \tau^c_u = f}} m^c_{u,v} \quad f \in F, v \in V \tag{12}$$

$$\begin{aligned} \psi_v &= \mathrm{CSW}_{\mathrm{proc}}(v) + \sum_{f \in F} \mathrm{UP}_{\mathrm{proc}}(f,v) \\ &= \sum_{f \in F} \lceil c_{f,v} \rceil \xi_v + \sum_{f \in F} \lceil c_{f,v} \rceil \mu_v \quad v \in V \end{aligned} \tag{13}$$

$$\sum_{f \in F} c_{f,v} \leq \gamma_v - \psi_v \quad v \in V \tag{14}$$

Eq. 6 guarantees that the fixed start/end point u of a SFC c is mapped to the node v specified by the parameter η^c_u and eq. 7 that it is not mapped to any other node. Note that eq. 6 and eq. 7 fix the value for some variables, because start/end points of SFCs are a-priori known.

Every VNF request u for each SFC c must be mapped to exactly one node v (eq. 8), in such a way that the overall number of VNF requests u mapped to the VNF instance f hosted by the node v cannot overcome $c_{f,v}/\pi_f$ (eq. 9). Remind that $\pi_f = N^{\mathrm{user}} \pi^{\mathrm{user}}_f$, i.e., the higher the number of users per SFC, the higher the processing requirement per VNF. Eq. 10 and eq. 11 ensure that $i_{f,v} = 0$ if $c_{f,v} = 0$ and that $i_{f,v} = 1$ if $c_{f,v} > 0$. $i_{f,v}$ is a flag variable specifying whether VNF f is mapped to node v. $\mathcal{M}_\gamma$ is a big-M parameter, greater than the maximum value taken by $c_{f,v}$, i.e., $\mathcal{M}_\gamma > \max_{v \in V}\{\gamma_v\}$. Note that eq. 11 includes a strict inequality and must be linearized[2]. Eq. 12 guarantees that an instance of VNF f is placed on a node v only if there is at least one request u mapped to it. Then, in eq. 13 we compute ψ_v, i.e., the overall context switching and upscaling processing costs per node v, as defined in Section III. Thus, the overall CPU processing assigned to instances of different VNFs f cannot overcome the actual processing capacity $\gamma_v - \psi_v$ of node v (eq. 14). Note that the ceiling function $\lceil c_{f,v} \rceil$ that appears in eq. 13 must be linearized.

3) Routing constraints:

$$\begin{aligned} &e^c_{v,v',x,y,u,u'} \leq m^c_{u,x} \; m^c_{u',y} \\ &c \in C, (v,v') \in E, x \in V, y \in V, (u,u') \in G^c \end{aligned} \tag{15}$$

$$\sum_{(x,v) \in E, \; x,y \in V} e^c_{x,v,x,y,u,u'} \; m^c_{u,x} \; m^c_{u',y} = 1 \quad c \in C, (u,u') \in G^c \tag{16}$$

$$\sum_{(v,y) \in E, \; x,y \in V} e^c_{v,y,x,y,u,u'} \; m^c_{u,x} \; m^c_{u',y} = 1 \quad c \in C, (u,u') \in G^c \tag{17}$$

[2]All the linearization techniques used in this work follow standard approaches for linearization of constraints such as the ones shown in [44].

$$\sum_{(v,x)\in E,\ v\in V} e^c_{v,x,x,y,u,u'} = 0$$
$$c \in C, x \in V, y \in V, x \neq y, (u,u') \in G^c \quad (18)$$

$$\sum_{(y,v)\in E,\ v\in V} e^c_{y,v,x,y,u,u'} = 0$$
$$c \in C, x \in V, y \in V, x \neq y, (u,u') \in G^c \quad (19)$$

$$\sum_{(v,w)\in E,\ v\in V} e^c_{v,w,x,y,u,u'} = \sum_{(w,v')\in E,\ v'\in V} e^c_{w,v',x,y,u,u'}$$
$$c \in C, w \in V, x \in V, y \in V, x \neq w, y \neq w, (u,u') \in G^c \quad (20)$$

$$\sum_{(v,w)\in E,\ v\in V} e^c_{v,w,x,y,u,u'} \leq 1$$
$$c \in C, w \in V, x \in V, y \in V, x \neq w, y \neq w, (u,u') \in G^c \quad (21)$$

$$e^c_{v,v,x,y,u,u'} = 0$$
$$c \in C, v \in V, x \in V, y \in V, x \neq y, (u,u') \in G^c \quad (22)$$

$$e^c_{v,v',x,x,u,u'} = 0$$
$$c \in C, x \in V, (u,u') \in G^c, (v,v') \in E, v \neq v' \quad (23)$$

Eq. 15 guarantees that the physical link (v, v') can belong to a path between two nodes x and y for a virtual link (u, u') of the SFC c only if the two consecutive VNF requests or start/end points u and u' are mapped to nodes x and y, respectively. The product $m^c_{u,x}\ m^c_{u',y}$ of binary variables must be linearized. Eq. 16 and eq. 17 are respectively the *source* and *destination* constraints. In fact, eq. 16 assures that the virtual link (u, u') between two consecutive VNF requests or start/end points u and u' originates from one of the links connected to the node x, where the VNF request or start point u is mapped to (eq. 16), and it ends in one of the links of the node y, where the VNF request or end point u' is mapped to (eq. 17). The products $e^c_{x,v,x,y,u,u'}\ m^c_{u,x}\ m^c_{u',y}$ and $e^c_{v,y,x,y,u,u'}\ m^c_{u,x}\ m^c_{u',y}$ must be linearized. In addition to source and destination constraints, the model also has to guarantee that no spurious links are selected. To do so, we have introduced eq. 18 and eq. 19. While mapping the virtual link (u, u') for the SFC c on a physical path between the nodes x and y where the VNF requests or start/end points u and u' are mapped to, no incoming physical link for the node x (eq. 18) and no outgoing link for the node y (eq. 19) must be considered. Eq. 20 and eq. 21 are then the *transit* constraints. While considering a generic node w (neither source node x nor destination node y of a virtual link (u, u')), if one of its incoming links belongs to the path between nodes x and y, then also one of its outgoing links must belong to the path (eq. 20). Without eq. 21 multiple incoming/outgoing links could be considered, but in this paper we deal with unsplittable flows. Finally, eq. 22 and eq. 23 ensure a correct usage of self-loops, as introduced in Section III. A self-loop for an NFV node v is used when two consecutive VNF requests or start/end points u and u' are mapped to the same node x, and cannot be used otherwise (eq. 22). Moreover, no physical link (v, v') other than the self-loop is used when VNF requests or start/end points u and u' are mapped to the same node x (eq. 23).

4) Performance constraints:

$$\sum_{\substack{c\in C,\ x,y\in V\\ (u,u')\in G^c}} e^c_{v,v',x,y,u,u'}\ \delta^c_{u,u'} \leq \beta_{v,v'} \quad (v,v') \in E \quad (24)$$

$$\sigma^c = \sum_{\substack{f\in F,\ v\in V\\ u\in U^c:\tau^c_u=f}} m^c_{u,v}\ (\mathrm{CSW}_{\mathrm{lat}}(v) + \mathrm{UP}_{\mathrm{lat}}(f,v))$$
$$= \sum_{\substack{f\in F,\ v\in V\\ u\in U^c:\tau^c_u=f}} m^c_{u,v}\ (\sum_{f\in F} \lceil c_{f,v}\rceil \omega_v + \lceil c_{f,v}\rceil \kappa_v) \quad c \in C \quad (25)$$

$$\sum_{\substack{(v,v')\in E,\ x,y\in V\\ (u,u')\in G^c}} e^c_{v,v',x,y,u,u'}\ l_{v,v'} + \sigma^c \leq \varphi^c \quad c \in C \quad (26)$$

$$\sum_{f\in F} i_{f,v} \leq \mathcal{M}_F\ a_v \quad v \in V \quad (27)$$

$$a_v \leq \sum_{f\in F} i_{f,v} \quad v \in V \quad (28)$$

Eq. 24 is the *bandwidth constraint*. It assures that the overall bandwidth $\delta^c_{u,u'}$ requested by the virtual link (u, u') of every SFC c and mapped to the physical link (v, v') cannot exceed the capacity of the link $\beta_{v,v'}$. In eq. 25 we compute σ^c, i.e., the overall context switching and upscaling latency costs per SFC c as defined in Section III. Note that σ^c refers to the latency introduced by *all* the NFV nodes crossed by the SFC c. Eq. 26 is then the *latency constraint*. It assures that the latency introduced by the network between start and end points of a SFC c cannot overcome the maximum tolerated latency φ^c. It takes into account both latency introduced by the propagation over physical links and by the NFV nodes due to upscaling and context switching (i.e., σ^c). Note that eq. 25 requires the linearization of the product between the binary variable $m^c_{u,v}$ and the real variable $\sum_{f\in F}\lceil c_{f,v}\rceil\omega_v + \lceil c_{f,v}\rceil\kappa_v$, as well as of the ceiling function $\lceil c_{f,v}\rceil$. Finally, eq. 27 and eq. 28 assure that a node is marked as *active* (i.e., $a_v = 1$) only if at least one instance of any VNF f is hosted by it. The big-M parameter $\mathcal{M}_F$ must be chosen such that $\mathcal{M}_F > |F|$.

B. Heuristic Cost-aware Algorithm description

Our Heuristic Cost-aware Algorithm (HCA) sequentially embeds SFCs $c \in C$. The algorithm starts by assuming that the physical links always have enough bandwidth capacity to accommodate the bandwidth requested by each SFC. This means that the link capacity is never a bottleneck, meaning that the network is overprovisioned as usually occurs in the core network segment. The pseudocode of HCA is shown in Alg. 1. The main idea is to build, using a greedy procedure, an embedding solution for each SFC c while trying to re-use already-placed VNF instances or already-active NFV nodes first (*Phase 1*). Only if the latency requirement φ^c for the SFC c is not met after *Phase 1*, a *Phase 2* is started to improve the solution.

Before starting the embedding, SFCs are sorted in an increasing order according to φ^c (line 1). This way, the most latency-sensitive SFCs are placed first. Then, the next SFC to embed is selected (line 3) and *Phase 1* starts. As first step of *Phase 1*, the next VNF request to chain for the SFC is

Algorithm 1 Heuristic Cost-aware Algorithm (HCA)

```
1: Sort SFCs by increasing latency
2: repeat
3:     Pick next SFC
       \* Start of Phase 1 *\
4:     repeat
5:         Pick next VNF request in SFC
6:         if ∃ instances of VNF in the network then
7:             Sort VNF instances by increasing
               distance from the last placed VNF instance
               or start point (current node)
8:             Try to scale up VNF instances until success
               or all VNF instances have been tried
9:             if success then
10:                continue
11:            end if
12:        end if
           \* Failure *\
13:        Sort NFV nodes by increasing residual capacity
14:        Try placing new VNF instance on an NFV node
           or all NFV nodes have been tried
15:        if failed then
16:            return(infeasible)
17:        end if
18:    until all VNF requests are chained
       \* Start of Phase 2 *\
19:    Check end-to-end latency of embedded SFC against
       latency requirement
20:    if success then
21:        continue
22:    end if
       \* Failure *\
23:    Release resources allocated in Phase 1
24:    Place chained VNF instances along the end-to-end
       latency shortest path
25:    Check end-to-end latency of embedded SFC against
       latency requirement
26:    if failed then
27:        return(infeasible)
28:    end if
29: until all SFCs are embedded
30: return(success)
```

selected (line 5) and an already-placed VNF instance for that VNF request is searched in the network (line 6). If more than one VNF instances are available, they are sorted by increasing distance (line 7), i.e., the one mapped to the closest NFV node v from the *current node*, according to the latency shortest paths (SPs), is selected first. The *current node* can be either the start point, if the embedding has just started, or the NFV node where the last-selected VNF request has been mapped to. HCA then tries to scale up the processing resources $c_{f,v}$ of the selected VNF instance on the NFV node v by a value π_f, i.e., $c_{f,v} \rightarrow c_{f,v} + \pi_f$ (line 8), in order to be able to process the additional aggregated traffic passing through the VNF request. This operation leads to an increase in both context switching and upscaling costs for the node v since they both depend on $c_{f,v}$. If the increase of context switching/upscaling latency costs compromises the end-to-end latency of already-embedded SFCs having one or more VNF requests mapped to v (i.e., for some already-placed SFC it happens that the updated end-to-end latency overcomes φ^c), the scale up fails and the next VNF instance, on another NFV node v, is checked (line 8). Otherwise, if successful, the scale up is executed, the VNF request is mapped to the VNF instance on v and the latency SP between the *current node* and the selected NFV node v is used to steer the SFC aggregated traffic between such physical nodes. The scale up can also fail if it triggers an increase in the context switching/upscaling processing costs such that not enough residual processing capacity is available after the scale up itself. If the scale up fails for all the already-placed VNF instances (i.e., the check of line 9 fails), the algorithm sorts the remaining NFV nodes by increasing residual capacity (line 13) and tries to place a new VNF instance of size π_f on an NFV node (line 14). This way, the most used NFV node v is selected first. This operation leads to an increase in the context switching/upscaling costs, and also in this case the algorithm checks that *(i)* the end-to-end latency for the already-embedded SFCs with one or more VNF requests mapped to v is not compromised and *(ii)* the residual processing capacity of the NFV node v, after updating the context switching/upscaling processing costs, is enough to host the new VNF instance. If such operation succeeds, the VNF request is mapped to the new VNF instance. The latency SP in the physical network between the *current node* and the selected NFV node v is then used to steer SFC aggregated traffic between such physical nodes. Note that the operation described in line 14 can imply the activation of an NFV node, if not enough residual processing capacity is available on already-active NFV nodes. If during *Phase 1* the placement of a VNF request fails, it means that the related SFC cannot be embedded for lack of processing resources and the overall embedding process is aborted (lines 15-16). Conversely, if all VNF requests are greedily mapped to NFV nodes (line 18), the NFV node where the last VNF request has been mapped is connected to the end node through the latency SP on the physical network. *Phase 1* is then completed and an embedding solution for the SFC is found.

The greedy SFC embedding solution built according to the steps shown in *Phase 1* tries to first use NFV nodes that are already active: this feature fits well with the objective of maximum consolidation. However, such solution can lead to a high end-to-end latency for the selected SFC, since the placement is unaware of the relative position of SFC start/end points with respect to NFV nodes where the chained VNF requests are placed. In fact, *Phase 1* only assures that traffic among VNFs is locally steered according to latency SPs among those NFV nodes where VNFs are mapped, but there is no guarantee that the solution is effective in terms of end-to-end latency.

For this reason, a *Phase 2* needs to be executed. *Phase 2* starts with the evaluation of the end-to-end latency of the solution built in *Phase 1* (line 19). If such latency is less or equal to φ^c, the embedding for the selected SFC is successful (lines 20-21): this happens typically for SFCs with a loose

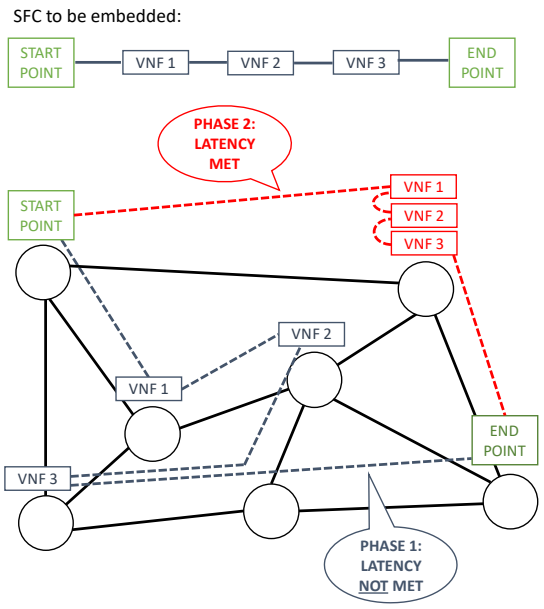

Fig. 6. Pictorial example of HCA *Phase 1* and *Phase 2* execution

latency requirement. Otherwise, the solution is discarded: this means that the processing resources allocated in *Phase 1* are released (line 23) and involved VNF instances are scaled down ($c_{f,v} \rightarrow c_{f,v} - \pi_f$). A new embedding solution is then built based on the latency SP between the start and end point of the selected SFC (line 24). This helps to minimize the end-to-end latency introduced by the links. In order to maximally consolidate the VNF instances, we activate an inactive NFV node on such SP and we place a new VNF instance f of size π_f for each chained VNF request on it, by also updating the context switching and upscaling costs for the NFV node. For the placement of those multiple VNFs, the algorithm chooses the NFV node with maximum processing capacity γ_v, to facilitate re-usability of the node for future embeddings. Finally, the end-to-end latency is checked again (line 25). If the latency requirement φ^c still cannot be met or no NFV node can be activated, there is no feasible solution (lines 26-27), otherwise the SFC embedding is completed and a new SFC is selected, until all SFCs have been successfully embedded (lines 29-30) or the operation fails during the embedding of other SFCs. Note that it is not always guaranteed that between start/end points of a SFC an NFV node exists on the latency SP. To overcome this issue, in *Phase 2* the algorithm computes the latency k shortest paths (k-SP), where k is chosen to ensure that at least one path crossing at least one NFV node is explored.

Figure 6 shows a pictorial example where a SFC concatenating three different VNF requests is embedded in the physical network according to *Phase 1* steps, and then the solution is improved by *Phase 2*. It should also now be clear why as first step HCA sorts SFCs in an increasing order according to their latency requirement: the idea is to first activate all the NFV nodes thar are needed to meet the latency requirement for latency-sensitive SFCs, and then to reuse such NFV nodes to embed SFCs requiring looser latency, which do not need to have a latency-optimized end-to-end path.

TABLE IV
PROCESSING REQUIREMENT (PER USER) FOR THE VNFS

Virtual Network Function	π^{user}
Network Address Translator (NAT)	0.00092
Firewall (FW)	0.0009
Traffic Monitor (TM)	0.0133
WAN Optimization Controller (WOC)	0.0054
Intrusion Detection Prevention System (IDPS)	0.0107
Video Optimization Controller (VOC)	0.0054

TABLE V
PERFORMANCE REQUIREMENTS FOR THE SFCS

SFC	Chained VNFs	Latency φ	Bandwidth δ^{user}
Web Service	NAT-FW-TM-WOC-IDPS	500 ms	100 kb/s
VoIP	NAT-FW-TM-FW-NAT	100 ms	64 kb/s
Video Streaming	NAT-FW-TM-VOC-IDPS	100 ms	4 Mb/s
Cloud Gaming	NAT-FW-VOC-WOC-IDPS	60 ms	4 Mb/s

V. COMPUTATIONAL RESULTS

In this section we apply our algorithms to an example network to study how they can be used for VNF consolidation. We also show how the two processing-resource sharing cost parameters introduced in the paper impact on the cost for NFV implementation in different scenarios.

We first describe the computational settings, then we compare results by solving the ILP model and running HCA on some small-scale instances. Finally, we deepen our investigation by running our HCA on much larger-scale instances, and we compare our strategy with a state-of-the-art approach.

A. Computational settings

We consider the ISP physical network topology shown in Fig. 1 with ten physical nodes ($|V| = 10$) and fifteen physical links ($|E| = 15$). This network topology is taken from the US Internet2 network [45], considering only the nodes with advanced layer 3 services. The latency introduced by the physical links due to propagation and forwarding is in the order of milliseconds (the shortest link has $\lambda_{v,v'} = 3$ ms, while the longest link has $\lambda_{v,v'} = 13.5$ ms). We also assume that the bandwidth on the physical links cannot be a bottleneck ($\beta_{v,v'} = \infty \ \forall (v,v') \in E$) and that all the physical nodes are NFV nodes and equipped with a multi-core CPU ($\gamma_v = 16 \ \forall v \in V$). Moreover, all the NFV nodes are always characterized by the same context switching and upscaling parameters, both in terms of latency (ω_v and κ_v) and processing (ξ_v and μ_v). We assume that latency and processing parameters (both concerning context switching and upscaling) are linearly dependent, according to another parameter h. This means that it always holds $\omega_v = h\xi_v$ and $\kappa_v = h\mu_v$. In our computational tests, unless otherwise specified, we set $h = 0.01$.

We consider six different VNFs, reported in Table IV. Each VNF has a different processing requirement per user π^{user}. The processing requirement for VNFs has been obtained according to middleboxes datasheets (see e.g. [46]) as ratio between the number of adopted CPU cores and of flows supported by the middlebox. Even though this is just a possible and rough estimation for the processing requirement (see [47] for a different strategy based on CPU cycles/s), it allows to

TABLE VI
PERFORMANCE COMPARISON BETWEEN ILP AND HCA

	$\|C\|=3$, $N^{\text{user}}=300$, $N^{\text{iter}}=100$		$\|C\|=6$, $N^{\text{user}}=150$, $N^{\text{iter}}=100$		$\|C\|=8$, $N^{\text{user}}=450$, $N^{\text{iter}}=10$	
Latency Costs [15]	HCA	ILP	HCA	ILP	HCA	ILP
Average number of active NFV nodes						
$\omega=0$ ms, $\kappa=0$ ms	2.91 ± 0.057	2.91 ± 0.057	2.95 ± 0.043	2.95 ± 0.043	6.00 ± 0.000	6.00 ± 0.000
$\omega=0$ ms, $\kappa=1.75$ ms	$2.95 \pm 0,052$	2.93 ± 0.058	2.99 ± 0.034	2.97 ± 0.034	6.11 ± 0.260	6.00 ± 0.000
$\omega=0.4$ ms, $\kappa=0$ ms	3.09 ± 0.090	3.07 ± 0.086	3.00 ± 0.028	2.99 ± 0.020	7.86 ± 0.640	6.45 ± 0.410
Results are reported along with 95% confidence intervals						
% Infeasible computational instances						
$\omega=0$ ms, $\kappa=0$ ms	0	0	0	0	0	0
$\omega=0$ ms, $\kappa=1.75$ ms	0	0	0	0	10	10
$\omega=0.4$ ms, $\kappa=0$ ms	0	0	0	0	30	10
Execution time per computational instance						
$\omega=0$ ms, $\kappa=0$ ms	32.81 ms	2.91 min	49.34 ms	4.32 min	2.01 s	1.80 h
$\omega=0$ ms, $\kappa=1.75$ ms	33.27 ms	3.23 min	49.90 ms	4.87 min	2.76 s	13.70 h
$\omega=0.4$ ms, $\kappa=0$ ms	33.13 ms	3.55 min	49.69 ms	5.19 min	2.35 s	16.85 h

TABLE VII
TUNABLE PARAMETERS IN EVALUATIONS

Tunable param.	Description
h	Dependency between latency/processing penalties
ω	Context switching latency
κ	Upscaling latency
N^{iter}	Number of computational instances
$\|C\|$	Number of SFCs to be embedded
N^{user}	Number of users per SFC

understand which are the most processing-hungry VNFs: for example, according to our estimation, a Traffic Monitor is about 15 times more processing-hungry than a Firewall. The six VNFs can be chained in different ways to provide four heterogeneous SFCs, reported in Table V. Such SFCs represent four different services, i.e., Web Service (WS), VoIP, Video Streaming (VS) and Cloud Gaming (CG). Table V shows also the performance requirements in terms of bandwidth δ and maximum tolerated latency φ for each SFC, assuming that every virtual link (u, u') of a SFC requires the same bandwidth. Performance requirements for WS, VoIP and VS are well known, while performance requirements for CG, which is a more recent service gaining more and more interest by the research community due to the technical challenges it poses [48], are taken by [49].

We show our results in different scenarios: a *mixed scenario* and some *homogeneous scenarios*. In the former we run a number N^{iter} of computational instances while uniformly randomizing the choice of SFC type to embed, among the four different types of SFCs, and of start/end points, among all the 10 physical nodes of the network. In the latter we only uniformly randomize the choice of start/end points, while we assume that only one type of SFC is embedded in the network.

Note that Table VII reports a summary of the tunable parameters in our evaluations to help the reader better understand the following results.

B. ILP and HCA performance comparison

We solved the ILP model using ILOG CPLEX solver, while we implemented HCA in Matlab. All the computational tests were run on a workstation equipped with 8×2 GHz CPU cores and with 32 GB of RAM. To compare results obtained by ILP and HCA we have focused on three different processing-resource sharing cost settings:

- A *No costs* setting ($\omega = 0$ ms, $\kappa = 0$ ms);
- An *Only upscaling costs* setting ($\omega = 0$ ms, $\kappa = 1.75$ ms);
- An *Only context switching costs* setting ($\omega = 0.4$ ms, $\kappa = 0$ ms).

We chose aforementioned values for ω and κ since they lead to NFV node latencies of few milliseconds, as in [14].

In this evaluation we consider a *mixed scenario*. For any of the cost settings, we have simulated an increasing number of SFCs, i.e., $|C| = 3$, 6, and 8. For $|C| = 3$ and $|C| = 6$ the overall number of users in the network is the same ($|C|N^{\text{user}} = 900$ in both cases), while for $|C| = 8$ a higher number of users ($|C|N^{\text{user}} = 3600$) is considered, leading thus to a more loaded network scenario. We consider a number of randomized instances $N^{\text{iter}} = 100$ for $|C| = 3$ and $|C| = 6$, while a number of randomized instances of $N^{\text{iter}} = 10$ for $|C| = 8$. Results are shown in Table VI. We report the *average number of active NFV nodes* along with the 95% confidence interval. We also report the *percentage of infeasible computational instances* and the *execution time per computational instance*. We can see how for $|C| = 3$ and $|C| = 6$, HCA closely matches results obtained by solving the ILP for all the cost settings. There are not infeasible instances and HCA allows to obtain a near-optimal solution in a computational time in the order of milliseconds per instance, while some minutes are needed to solve the ILP model. In case of $|C| = 8$ and more loaded network the results are slightly different. First of all, for both HCA and ILP some instances are infeasible. This can happen because *(i)* the latency requirement for some of the SFCs cannot be met or *(ii)* there is not enough processing in the NFV nodes to place all the VNFs. Especially, for the *Only context switching costs* there is a higher infeasibility percentage for HCA (30%) than for the ILP (10%). This means that part of the ILP feasible solutions cannot be explored by HCA. Additionally, in such case, HCA activates on average about 1.4 NFV nodes more than the optimal solution. However, execution times per computational instance are in the order of seconds for HCA, while the ILP needs several hours to be solved. Especially, the needed time to solve it for the *Only context switching costs*

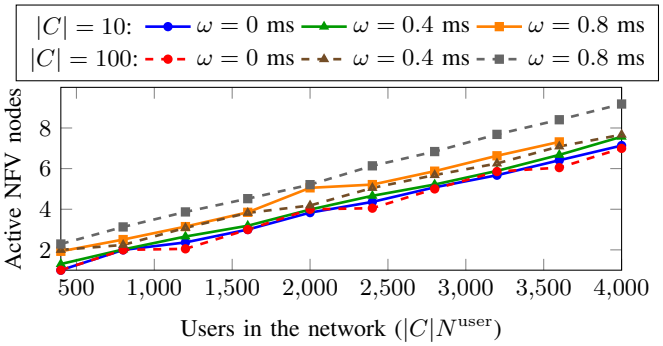

Fig. 7. Number of active NFV nodes as a function of the overall number of users in the network, while considering the impact of different *latency context switching costs* ω and different numbers of SFCs $|C|$ in the *mixed scenario* ($\kappa = 0$, $N^{\text{iter}} = 1000$)

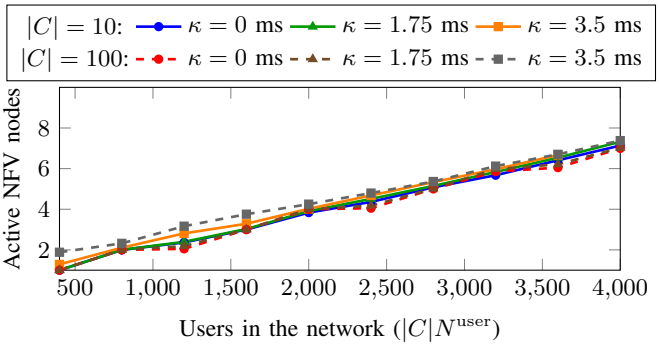

Fig. 8. Number of active NFV nodes as a function of the overall number of users in the network, while considering the impact of different *latency upscaling costs* κ and different numbers of SFCs $|C|$ in the *mixed scenario* ($\omega = 0$, $N^{\text{iter}} = 1000$)

(16.85 h) and *Only upscaling costs* (13.70 h) settings is about 10 times higher than the *No costs* setting solving time (1.80 h). This happens because in the *Only context switching costs* and *Only upscaling costs* settings the solver must compute $\psi_v \geq 0$ (eq. 13) and $\sigma^c \geq 0$ (eq. 25). Those terms include the ceiling function $\lceil c_{f,v} \rceil$, which is nonlinear and must be linearized. This is not true for the *No costs* setting, since $\omega = \kappa = 0$ implies $\psi_v = \sigma^c = 0$ and thus $\lceil c_{f,v} \rceil$ must not be computed by the solver, leading to a simpler ILP model to solve.

C. Mixed scenario

After having verified the effectiveness of HCA, we ran it for larger-scale instances to deepen the study. In this section we show the results obtained for the mixed scenario. Figure 7 shows the impact of different context switching costs, by varying ω, on the number of active NFV nodes. This is evaluated as a function of the overall number of users in the network $|C|N^{\text{user}}$. Upscaling costs are considered negligible ($\kappa = 0$). We consider $N^{\text{iter}} = 1000$ and the embedding of $|C| = 10$ and $|C| = 100$ SFCs. The overall number of users in the network is equally split among the number of SFCs $|C|$ (i.e., if the users in the network are 1000 and $|C| = 100$ then $N^{\text{user}} = 10$, while if $|C| = 10$ then $N^{\text{user}} = 100$). This way, we can evaluate the impact of the number of SFCs to embed on VNF consolidation without changes in the overall network load. Figure 8 shows instead the impact of different upscaling costs, by varying κ, when context switching costs are negligible ($\omega = 0$). We adopt the same settings as for Fig. 7. In both cases we plot only values for which the percentage of infeasible instances is less than 20%. The curves for both Fig. 7 and Fig. 8 show a non-decreasing trend. In fact, increasing the number of users in the network implies an increase in the processing requirements π for any placed VNF instance. Thus, more NFV nodes need to be activated to accommodate bigger VNFs. By comparing Fig. 7 and Fig. 8, we can then notice two different trends. In Fig. 7, by increasing the value of ω, the relative difference between $|C| = 10$ and $|C| = 100$ curves strongly increases. This is not true for Fig. 8, where such relative difference does not significantly vary. Moreover, in Fig. 8 all the curves show a weak dependence on the upscaling costs and on the number of SFCs, meaning that in general upscaling costs have less impact than context switching costs on the number of active NFV nodes.

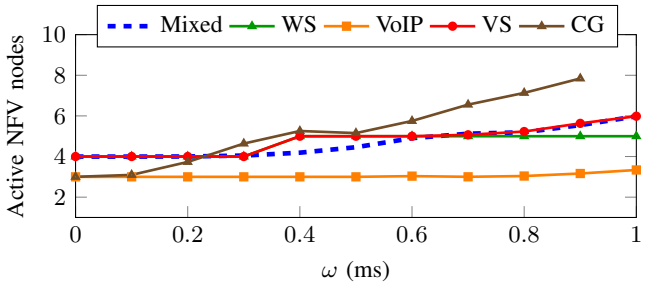

Fig. 9. Number of active NFV nodes as a function of the *latency context switching costs* parameter ω, considering four *homogeneous scenarios* according to the SFCs of Tab. V ($|C| = 100$, $N^{\text{user}} = 20$, $\kappa = 0$, $N^{\text{iter}} = 1000$)

D. Homogeneous scenarios

Here we consider results obtained in four different homogeneous scenarios. Figure 9 shows the number of active NFV nodes in the network as a function of the context switching costs parameter ω for the SFCs of Table V. As we can see, SFCs with different requirements and chaining different VNFs have diverse impact on the cost for NFV implementation, measured by the number of active NFV nodes. The difference is mainly due to the distinct impact of both context switching and processing requirement of VNFs chained by SFCs. Looking at Tab. IV and Tab. V, it is easy to see how WS and VS SFCs concatenate VNFs that, on average, have more processing requirement per user than VNFs chained by VoIP and Cloud Gaming SFCs. This explains why, for small values of ω, the number of active NFV nodes is the same for WS/VS and for VoIP/CG homogeneous scenarios. Increasing the context switching parameter ω, in general, leads to an increase in the number of active NFV nodes because both processing and latency context switching costs increase, making it harder to meet VNF processing and SFC latency requirements while activating a small number of nodes. However, the impact of context switching for VoIP homogeneous scenario starts to be noticeable only for high values of ω, since the latency requirement is not very strict (100 ms) and the processing requirement for its chained VNFs is low. Additionally, the impact of context switching costs is very similar for WS and

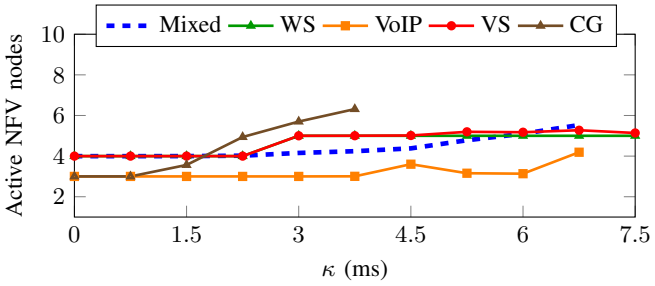

Fig. 10. Number of active NFV nodes as a function of the *latency upscaling costs* parameter κ, considering four *homogeneous scenarios* according to the SFCs of Tab. V ($|C| = 100$, $N^{\text{user}} = 20$, $\omega = 0$, $N^{\text{iter}} = 1000$)

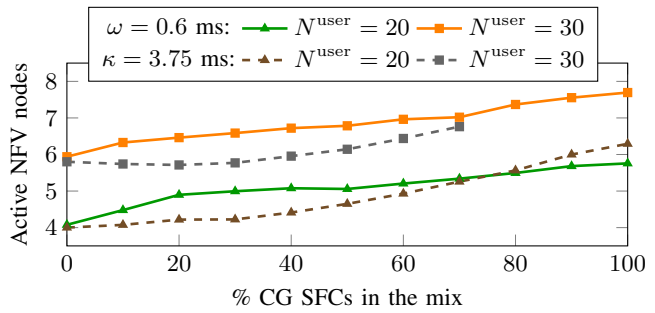

Fig. 11. Number of active NFV nodes as a function of the percentage of Cloud Gaming SFCs in the mix considering context switching latency costs $\omega = 0.6$ ms, upscaling latency costs $\kappa = 3.75$ ms and different number of aggregated users N^{user} per SFC ($|C| = 100$, $N^{\text{iter}} = 1000$)

VS despite for high values of ω. This happens because the average processing requirement for the VNFs chained by WS and VS SFCs, as recalled earlier, is very similar. However, for $\omega > 0.7$ ms, the curves start diverging, being the number of active nodes for VS higher than for WS. In fact, VS SFCs have a stricter latency requirement than WS SFCs and, starting from $\omega > 0.7$ ms, latency introduced by NFV nodes due to context switching becomes significant. This implies that more nodes need to be activated to meet the VF SFCs latency requirement. Finally, CG SFCs have a very strict latency requirement (60 ms). For this reason, context switching costs have a very strong impact on the cost for NFV implementation. A high number of NFV nodes must be activated because, in order to guarantee the latency requirement, all VNFs must be placed on a end-to-end physical path between start/end points of each SFC close to the latency shortest path. It is also important to note how the curve of mixed scenario leads more or less to average values with respect to the curves representing homogeneous scenarios. For values of $\omega > 0.4$ ms, latency introduced by NFV nodes due to context switching starts to affect the placement of VNFs, which must ensure that latency requirement for latency-sensitive SFCs (i.e., CG) in the mix is met. Similar considerations can be made for the upscaling costs, whose results are shown in Fig. 10.

Since the CG homogeneous scenario has the greatest impact on the cost for NFV implementation, it is interesting to investigate how different percentages of CG SFCs in the mix influence the number of NFV active nodes. Figure 11 shows the number of active NFV nodes as a function of the percentage of CG SFCs in the mix. Shown results focus on a

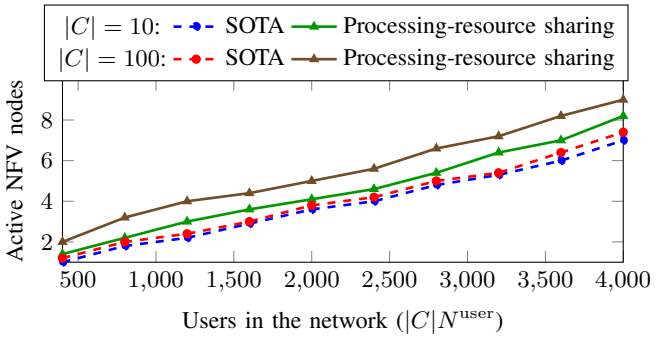

Fig. 12. Number of active NFV nodes as a function of the of the overall number of users in the network, while considering SOTA and our proposed *node model* for different numbers of SFCs $|C|$ in the *mixed scenario* ($h = 0$, $N^{\text{iter}} = 1000$)

total number of SFCs $|C| = 100$, consider different values of N^{user} ($N^{\text{user}} = 20$ and $N^{\text{user}} = 30$) and examine both context switching ($\omega = 0.6$ ms) and upscaling ($\kappa = 3.75$ ms). Results show that, concerning context switching costs, even a small number of CG SFCs in the mix has a strong impact on the number of active NFV nodes. For example, in case of $\omega = 0.6$ ms and $N^{\text{user}} = 20$, having 20% of CG SFCs in the mix is enough to significantly increase the average number of active NFV nodes with respect to the case where no CG SFCs must be embedded (0%). This does not happen with upscaling costs. In this case, for $\kappa = 3.75$ ms and $N^{\text{user}} = 20$, it is possible to notice a significant increase in the number of NFV active nodes only for percentages above 50%. These results are in line with results shown previously for the mixed scenario, and we can then in general conclude that context switching costs have a much stronger impact on the cost for NFV implementation than upscaling costs.

E. Comparison with the state of the art

Existing NFV node models for the estimation of expected introduced latency do not consider any processing-resource sharing aspect. The most common adopted model, which we call state-of-the-art (SOTA), merely considers a non-linear increase in the node latency as a function of the node utilization [4], neglecting any upscaling or context switching costs. In this subsection, we compare SFC embedding results obtained with our proposed processing-resource sharing node model, which relies on the costs introduced in Section III-B, with results obtained by adopting the SOTA model proposed in [4]: to do so, we have modified HCA to embed SFCs according to such simplified model. Specifically, considering $P_v = \sum_{f \in F} \frac{c_{f,v}}{\gamma_v}$ as NFV node utilization, latency introduced by any NFV node v, for any VNF instance hosted by that node, is computed in the following way:

$$\text{SOTA}_{\text{lat}}(v) = \frac{P_v - [1 + K(1 - P_v)]P_v^{K+1}}{L(1 - P_v)(1 - P_v^K)} \tag{29}$$

As per [4], we set $K = 100$ and $L = 10$. The computed node latencies are in the order of milliseconds, as the ones obtained with our model when $\omega = 0.4$ ms and $\kappa = 1.75$ ms, which are the values adopted in this subsection. Note also that, to simplify the evaluation, we assume no processing penalties, i.e., $h = 0$.

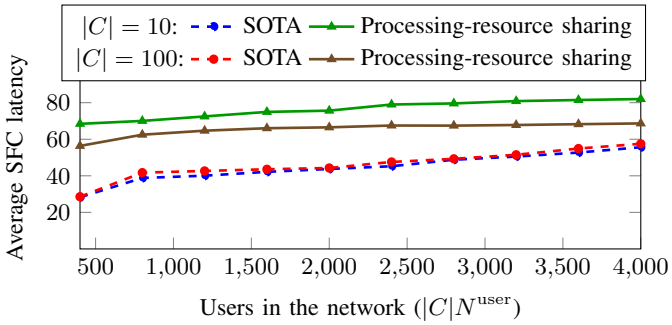

Fig. 13. Average SFC end-to-end latency as a function of the of the overall number of users in the network, while considering SOTA and our proposed *node models* and different numbers of SFCs $|C|$ in the *mixed scenario* ($h = 0$, $N^{\text{iter}} = 1000$)

Since the SOTA model does not take into consideration how the processing capacity, in terms of number of cores, is assigned to different VNFs in an NFV node, it naturally misses to consider any arising processing-resource sharing latency penalty. For this reason, it tends to underestimate the NFV node latency, leading to greater VNF consolidations. This is shown in Figs. 12 and 13.

Fig. 12 shows the impact of the two different node models on the number of active NFV nodes. This is investigated as a function of the overall number of users in the network $|C|N^{\text{user}}$ in a mixed scenario. What can be seen is that the number of active NFV nodes by adopting the SOTA model is lower than in the case our processing-resource sharing model is considered, meaning that the SOTA model consolidates VNFs in less NFV nodes. However, being the curves for $|C| = 10$ and $|C| = 100$ overlapped for the SOTA model, it means that the number of active NFV nodes is invariant to $|C|$. Conversely, our model activates more NFV nodes as $|C|$ increases. This happens because with an increase of $|C|$ more VNF instances must be embedded in the network and more processing-resource sharing penalties arise. This is not true for the SOTA model, where the only relevant parameter is node utilization, which is roughly the same also in case of different $|C|$. Our model, thus, avoids an excessive VNF consolidation (especially when $|C|$ is larger) that could compromise the SFC end-to-end latency, especially for latency-sensitive SFCs.

Fig. 13 shows instead the average experienced end-to-end SFC latency, while considering the two models, in the same mixed scenario. As expected, the average latency is lower while adopting the SOTA model, and does not depend on $|C|$. This latency underestimation is mostly dangerous for latency-sensitive SFCs (e.g. CG SFCs), which would be embedded on paths that are estimated to meet their end-to-end latency requirements, but in reality may lead to much higher latency penalties due to processing-resource sharing.

VI. Conclusion

In this paper, we investigated the impact on the network cost of processing-resource sharing among VNFs in NFV, when multiple SFCs must be embedded in the network. VNF placement and distribution on NFV nodes lead to two different types of costs: upscaling and context switching costs, which affect the placement of VNFs and the embedding of SFCs. We first focused on the mathematical modeling of NFV nodes, VNFs, SFCs and of processing-resource sharing costs. Then, we formulated an ILP model and we designed a heuristic algorithm, called HCA, to evaluate the impact of such costs on VNF consolidation. We showed that HCA allows to obtain a near-optimal solution in a much shorter time than solving the ILP model. We then gathered some numerical results by focusing our attention on the placement of practical SFCs. Results showed that, in the considered ISP network, context switching costs have a greater impact on VNF consolidation than upscaling costs. Besides, such costs strongly affect NFV consolidation when SFCs with a very strict latency requirement, such as Cloud Gaming SFCs, must be embedded in the network. We also showed that this aspect cannot be captured by state-of-the-art node models neglecting processing-resource sharing aspects.

Several issues remain open for future research. In fact, processing-resource sharing is just one of the possible resource sharing issues. Other resource sharing issues concerning memory and storage could be investigated. Moreover, VNF consolidation is not the only possible objective that a network operator is interested to achieve. Many other objectives, also taking into account bandwidth resources on the physical links, could be taken into consideration.

References

[1] M. Savi, M. Tornatore, and G. Verticale, "Impact of processing costs on service chain placement in network functions virtualization," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, Nov. 2015.

[2] W. John, K. Pentikousis, G. Agapiou, E. Jacob, M. Kind, A. Manzalini, F. Risso, D. Staessens, R. Steinert, and C. Meirosu, "Research directions in Network Service Chaining," in *IEEE Conference on SDN for Future Networks and Services*, Nov. 2013.

[3] C. Zeng, F. Liu, S. Chen, W. Jiang, and M. Li, "Demystifying the performance interference of co-located Virtual Network Functions," in *IEEE INFOCOM*, Apr. 2018.

[4] F. C. Chua, J. Ward, Y. Zhang, P. Sharma, and B. A. Huberman, "Stringer: Balancing latency and resource usage in service function chain provisioning," *IEEE Internet Computing*, vol. 20, no. 6, pp. 22–31, Nov. 2016.

[5] M. McCool, "Scalable programming models for massively multicore processors," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 816–831, May 2008.

[6] D. Patterson, "The trouble with multi-core," *IEEE Spectrum*, vol. 47, no. 7, pp. 28–32, 53, Jul. 2010.

[7] F. Fusco and L. Deri, "High speed network traffic analysis with commodity multi-core systems," in *10th ACM SIGCOMM Conference on Internet Measurement*, Nov. 2010.

[8] X. Geng, G. Xu, and Y. Zhang, "Dynamic load balancing scheduling model based on multi-core processor," in *International Conference on Frontier of Computer Science and Technology*, Aug. 2010.

[9] T. Buh, R. Trobec, and A. Cigli, "Adaptive network-traffic balancing on multi-core software networking devices," *Computer Networks*, vol. 69, pp. 19 – 34, Aug. 2014.

[10] C. Li, C. Ding, and K. Shen, "Quantifying the cost of context switch," in *ACM Workshop on Experimental Computer Science*, Jun. 2007.

[11] R. Fromm and N. Treuhaft, "Revisiting the cache interference costs of context switching," 1996.

[12] X. Zhao, J. Yao, P. Gao, and H. Guan, "Efficient sharing and fine-grained scheduling of virtualized GPU resources," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, Jul. 2018.

[13] H. Asai, "Where are the bottlenecks in software packet processing and forwarding? Towards high-performance network operating systems," in *9th ACM International Conference on Future Internet Technologies*, Jun. 2014.

[14] I. Cerrato, G. Marchetto, F. Risso, R. Sisto, and M. Virgilio, "An efficient data exchange algorithm for chained network functions," in *IEEE International Conference on High Performance Switching and Routing*, Jul. 2014.

[15] I. Cerrato, M. Annarumma, and F. Risso, "Supporting fine-grained network functions through Intel DPDK," in *European Workshop on Software Defined Networks*, Sep. 2014.

[16] Q. Li, Y. Jiang, P. Duan, M. Xu, and X. Xiao, "Quokka: Latency-aware middlebox scheduling with dynamic resource allocation," *Journal of Network and Computer Applications*, vol. 78, pp. 253 – 266, 2017.

[17] J. Botero, X. Hesselbach, M. Duelli, D. Schlosser, A. Fischer, and H. de Meer, "Energy efficient virtual network embedding," *IEEE Communications Letters*, vol. 16, no. 5, pp. 756–759, May 2012.

[18] C. Fuerst, S. Schmid, and A. Feldmann, "Virtual network embedding with collocation: Benefits and limitations of pre-clustering," in *IEEE International Conference on Cloud Networking*, Nov. 2013.

[19] N. Chowdhury, M. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *IEEE INFOCOM*, Apr. 2009.

[20] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *IEEE International Conference on Cloud Networking*, Oct. 2014.

[21] H. Moens and F. De Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *International Conference on Network and Service Management*, Nov. 2014.

[22] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *IEEE 4th International Conference on Cloud Networking (CloudNet)*, Oct. 2015.

[23] A. Gupta, M. F. Habib, P. Chowdhury, M. Tornatore, and B. Mukherjee, "On service chaining using virtual network functions in network-enabled cloud systems," in *IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS)*, Dec. 2015.

[24] A. Mohammadkhan, S. Ghapani, G. Liu, W. Zhang, K. Ramakrishnan, and T. Wood, "Virtual function placement and traffic steering in flexible and dynamic software defined networks," in *IEEE International Workshop on Local and Metropolitan Area Networks (LANMAN)*, Apr. 2015.

[25] M. Bari, S. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *ACM International Conference on Network and Service Management (CNSM)*, Nov. 2015.

[26] M. Bouet, J. Leguay, and V. Conan, "Cost-based placement of vDPI functions in NFV infrastructures," in *1st IEEE Conference on Network Softwarization (NetSoft)*, Apr. 2015.

[27] R. Riggio, T. Rasheed, and R. Narayanan, "Virtual network functions orchestration in enterprise WLANs," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015.

[28] F. Carpio, S. Dhahri, and A. Jukan, "VNF placement with replication for load balancing in NFV networks," in *2017 IEEE International Conference on Communications (ICC)*, May 2017.

[29] B. Ren, D. Guo, G. Tang, X. Lin, and Y. Qin, "Optimal Service Function Tree embedding for NFV enabled multicast," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, Jul 2018.

[30] L. Guo, J. Pang, and A. Walid, "Joint placement and routing of Network Function Chains in data centers," in *IEEE INFOCOM*, Apr. 2018.

[31] J. Cao, Y. Zhang, W. An, X. Chen, Y. Han, and J. Sun, "VNF placement in hybrid NFV environment: Modeling and genetic algorithms," in *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, Dec. 2016.

[32] A. Hirwe and K. Kataoka, "Lightchain: A lightweight optimisation of vnf placement for service chaining in nfv," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, June 2016.

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

[34] S. Oechsner and A. Ripke, "Flexible support of VNF placement functions in OpenStack," in *1st IEEE Conference on Network Softwarization (NetSoft)*, Apr. 2015.

[35] J. Ferrer Riera, E. Escalona, J. Batalle, E. Grasa, and J. Garcia-Espin, "Virtual network function scheduling: Concept and challenges," in *International Conference on Smart Communications in Network Technologies (SaCoNeT)*, Jun. 2014.

[36] C. Ghribi, M. Mechtri, and D. Zeghlache, "A dynamic programming algorithm for joint VNF placement and chaining," in *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking*, Dec. 2016.

[37] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and S. Davy, "Design and evaluation of algorithms for mapping and scheduling of virtual network functions," in *1st IEEE Conference on Network Softwarization (NetSoft)*, Apr. 2015.

[38] S. Sahhaf, W. Tavernier, J. Czentye, B. Sonkoly, P. Skoldstrom, D. Jocha, and J. Garay, "Scalable architecture for service function chain orchestration," in *4th European Workshop on Software Defined Networks (EWSDN)*, Sept. 2015.

[39] T. Lukovszki and S. Schmid, "Online admission control and embedding of service chains," in *22nd International Colloquium on Structural Information and Communication Complexity (SIROCCO 2015)*, Jul. 2015.

[40] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic virtual network function placement," in *IEEE 4th International Conference on Cloud Networking (CloudNet)*, Oct 2015.

[41] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive VNF scaling and flow routing with proactive demand prediction," in *IEEE INFOCOM 2018*, Apr. 2018.

[42] L. Peterson, A. Al-Shabibi, T. Anshutz, S. Baker, A. Bavier, S. Das, J. Hart, G. Palukar, and W. Snow, "Central office re-architected as a data center," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 96–101, Oct. 2016.

[43] P. W. Chi, Y. C. Huang, and C. L. Lei, "Efficient NFV deployment in data center networks," in *2015 IEEE International Conference on Communications (ICC)*, Jun. 2015.

[44] H. Sherali and W. Adams, "A reformulation-linearization technique for solving discrete and continuous nonconvex problems," *Springer US books*, 2010.

[45] "Internet2 network infrastructure topology," Oct. 2014. [Online]. Available: http://www.internet2.edu/media_files/422

[46] "Dell SonicWALL SuperMassive 9000 Series Firewall." [Online]. Available: https://www.sonicwall.com/documents/sonicwall-supermassive-next-generation-firewall-series-datasheet-68226.pdf

[47] R. Doriguzzi-Corin, S. Scott-Hayward, D. Siracusa, M. Savi, and E. Salvadori, "Dynamic and application-aware provisioning of chained virtual security network functions," *CoRR arXiv*, vol. abs/1901.01704, 2019.

[48] M. Garcia-Valls, T. Cucinotta, and C. Lu, "Challenges in real-time virtualization and predictable cloud computing," *Journal of Systems Architecture*, vol. 60, no. 9, pp. 726 – 740, Oct. 2014.

[49] R. Shea, J. Liu, E.-H. Ngai, and Y. Cui, "Cloud gaming: architecture and performance," *IEEE Network*, vol. 27, no. 4, pp. 16–21, Jul. 2013.

Marco Savi is a postdoctoral researcher at FBK CREATE-NET, Trento, Italy. He received his PhD degree in Information Technology (Telecommunication engineering) in 2016 from Politecnico di Milano. His research interests mainly focus on the design and optimization of telecommunication networks, especially optical and 5G networks, and cloud computing. Dr. Savi has been involved in some European research projects advancing access and core network technologies.

Massimo Tornatore is an associate professor at Politecnico di Milano, Italy, and an adjunct associate professor at the University of California, Davis, USA. He is author of about 200 technical papers (with 7 best paper awards) and his research interests include the design and engineering of telecom and cloud networks, through optimization and simulation. Prof. Tornatore received a PhD degree in 2006 from Politecnico di Milano.

Giacomo Verticale is assistant professor at Politecnico di Milano, Italy. He obtained his PhD in Telecommunication Engineering in year 2003 from Politecnico di Milano defending a thesis on the performance of packet transmission in UMTS. In the years 1999-2001 he was with the research center CEFRIEL, working on the Voice-over-IP and ADSL technologies. He was involved in several European research projects advancing the Internet technology. His current interests focus on the security issues of the Smart Grid and on NFV.