Submitted version of the following published paper:
F. Paganelli, P. Cappanera, G. Cuffaro,"Tenant-defined service function chaining in a multi-site network slice",
Future Generation Computer Systems,Volume 121,2021, Pages 1-18, ISSN 0167-739X,
https://doi.org/10.1016/j.future.2021.03.002

Tenant-defined Service Function Chaining in a Multi-Site Network Slice

Federica Paganelli[a,b], Paola Cappanera[c], Giovanni Cuffaro[b]

[a] *Department of Computer Science, University of Pisa, Italy*
[b] *CNIT, Research Unit at the University of Florence, Italy*
[c] *Department of Information Engineering, University of Florence, Italy*

Abstract

Service Function Chaining (SFC) enables dynamic and adaptive network service provisioning. However, virtual infrastructure providers are reluctant to disclose SFC APIs and infrastructure monitoring information to tenants, therefore undermining tenants' capability in dynamically and flexibly managing application-tailored network services. In this work we provide a two-fold contribution to the problem of tenant-defined service function chaining on a multi-site virtual infrastructure, which has not been widely investigated yet. First, we propose a VNF Selection algorithm that relies on an abstracted network model thus minimizing topological and monitoring information required from the infrastructure. Our algorithm selects a set of VNF instances that minimize the end-to-end latency (accounting for network and processing delay) and also guarantees that already established chains do not violate their maximum latency constraint due to increased load at VNFs brought by new chain requests. Second, we describe an SFC solution composed of an SFC application leveraging our VNF Selection algorithm and a forwarding mechanism allowing chain management and traffic steering control within the tenant network without accessing infrastructure

[☆] This work was partially supported by the 5GINFIRE research project funded by the European Union, under grant agreement No. 732497 of the Horizon 2020 research and innovation program

[*] Corresponding author
Email addresses: federica.paganelli@unipi.it (Federica Paganelli), paola.cappanera@unifi.it (Paola Cappanera), giovanni.cuffaro@cnit.it (Giovanni Cuffaro)

control API. Validation and comparative evaluations are performed through simulation-based testing. Finally, the proposed approach is assessed through a set of experiments carried out on a multi-site testbed infrastructure.
Keywords: `Network Function Virtualization, Service function chaining, optimization, tenant network, Software Defined Networking, intent interface.`

---

1. Introduction

Network Function Virtualization (NFV) [1] and Software Defined Networking (SDN) [2] are considered key technological paradigms in modern networking systems for efficiently coping with emerging challenges in managing complex and unpredictable traffic patterns in different scenarios, ranging from Internet of Things, cloud networking and mobile data traffic toward new fifth generation (5G) networks [3]. The joint use of NFV and SDN are expected to provide a virtualized network infrastructure endowed with programmable connectivity and flexible on-demand service provisioning. The abstraction concepts brought and implemented by NFV and SDN technologies (i.e. software vs hardware, control vs data plane, etc.) are also expected to facilitate creating and managing multiple logical networks on top of a common physical infrastructure.

Recently, *network slicing*, which consists in creating and managing multiple logical self-contained networks on top of a common physical infrastructure, has recently emerged as a key technology of 5G networks [4]. Indeed, network slicing aims to exploit network programmability and virtualization to create on demand and cost-efficient end-to-end network slices to provide differentiated connectivity services to fulfil requirements of distinct applications, customers and/or operators [5]. Different types of networks slices may be defined by taking into account how management and control capabilities can be split between network providers and tenants, as discussed in [6]. In this work we refer to the case of *tenant-managed slices*, where the tenant may control resources and allocated functions since it has been granted access to a (limited) set of operations

and/or configuration actions [6].

Tenant-managed slices are complex to be established and maintained since information disclosure and access control have to be agreed between tenants and providers [6]. Security, privacy and performance issues are major obstacles to the adoption of multi-tenancy approaches [7, 8, 9]. In particular, the provisioning of open interfaces that map tenant-issued operations into commands for the infrastructure has to be carefully decided, since it can bring potential vulnerabilities and, therefore, favour attacks [8]. A conservative approach may consist in limiting tenants' access to computing and network resource management APIs, thus potentially undermining tenants' capability in coping with on-demand and specific requirements of application-tailored network services, especially Service Function Chaining (SFC). In fact, network infrastructure operators may decide to expose an abstracted view of infrastructure resources to hide internal implementation and status details as well as fine-tune deployment decisions within their own organization domain boundaries [10]. However, this approach may limit a tenant's capability in provisioning dynamic and adaptive network services, such as those realized as network function chains.

SFC is the problem of defining and instantiating an ordered list of network functions and the subsequent "steering" of traffic flows through those functions [11]. Examples of network functions are firewalls, load balancers, WAN accelerators and intrusion detection devices. These functions can be deployed as Virtual Network Functions (VNFs) [1], so to leverage lifecycle management features brought by virtualization.

Related work on SFC (e.g. [12, 13, 14]) mostly focuses on the infrastructure level, i.e. it is assumed that service chain control is operated by network providers and, thus, related control and forwarding features (e.g. SDN controller, VIM networking functions, programmable switches) are deployed in the infrastructure. The problem of SFC management within the tenant domain has been rarely addressed so far. A few works [15, 9] cope with SDN-based service offered to tenants, but such approaches require access to programmable switches in the network infrastructure.

In this work, we tackle the problem of tenant-managed SFC in a multi-domain network slice. More specifically, we assume that a network slice might span multiple clouds and networks owned by different providers, as also assumed in [16] and [17]. Moving from the assumption that a set of network functions have already been deployed in the tenant slice, our goal consists in *i)* supporting the tenant in selecting, among pre-deployed VNF instances, those that fulfil an incoming service request and related Quality of Service (QoS) requirements, and *ii)* realizing the service chain by leveraging control and forwarding capabilities defined within the tenant domain.

Related literature include works proposing centralized [18, 17] or distributed approaches [19, 20] to the problem of service chain embedding in a multi-domain infrastructure. These works account, even though to different extents, for the limitations imposed on information disclosure to tenants, but they focus on VNF placement rather than VNF selection (i.e., selecting previously deployed VNF instances for realizing a service function chain). Instead, only a few works [21, 22, 23, 24, 25] have addressed the problem of VNF selection, which is significant for orchestrating the usage of available VNF instances for coping with application QoS requirements [26] and/or load balancing purposes [22]. However, previous work does not consider the problem from a tenant's point of view in a multi-provider network. To the state of our knowledge, this is the first work that targets the problem of tenant-managed SFC accounting for tenants' limitation on information and control access, while providing both simulation and experimental validation.

More specifically, the contribution of this work is two-fold:

1. A theoretical contribution consisting in an optimization algorithm for selecting VNF instances considering latency-aware requirements for service provisioning. The proposed VNF selection strategy moves from a problem statement that takes into account tenant role and limited disclosure of network topology information and network control features. We formulate the VNF Selection problem by relying on an abstracted network topology

model that has the advantage of simplifying the problem complexity, while modelling a realistic scenario where network providers expose only an abstracted view of resources and network status [10].

2. An applied contribution consisting in the development and experimentation of an SFC application implementing the above mentioned VNF selection strategy on a multi-site testbed. In order to validate the proposed approach in a realistic scenario, we performed an experiment on a multi-Data Center (DC) slice in top of the testbed infrastructure facilities provided by the 5GINFIRE project [27]. The experiment included also the design and development of a tenant-based SFC forwarding mechanism, inspired by the architectural design proposed by the European Telecommunications Standard Institute (ETSI) NFV working group in [28]. In the proposed mechanism, the forwarding layer receives flow programming instructions through intent-based REST APIs and translates them into flow forwarding rules enforced by Linux policy routing capabilities.

A preliminary version of the algorithm was presented in a previous work [29]. Here we extend such preliminary contribution as follows: i) a processing latency model at VNF nodes based on queuing theory has been introduced; ii) the VNF selection algorithm (called *SelSFC*) has been extended to evaluate at each selection decision also the maximum latency constraint of previously established chains (*SelSFC-P* algorithm); iii) a simulation-based performance evaluation is included to validate the work at a larger scale with respect to the experimental findings. A preliminary description of the experiment activities, focused on the performance of our forwarding mechanism and latency estimation, is provided in [30], while this work complements such description by focusing on the tools developed for experiment purposes and extending the comparative evaluation of the VNF Selection algorithm.

The remainder of this paper is organized as follows. In Section 2, we describe the problem statement in a reference scenario and, then, we formulate the VNF Selection problem. In Section 3, we evaluate the algorithm through simulation.

Then, the experiment is described as follows: the architectural design and implementation choice of the experiment are discussed in Section 4, while results are discussed in Section 5. In Section 6, we discuss related work and Section 7 concludes the paper with insights for future work.

2. VNF Selection for tenant-defined SFC

This section formulates the VNF Selection problem for tenant-managed SFC in a multi-domain network. First, we clarify our reference network scenario, also leveraging ETSI [28, 10] and IETF specifications [31, 32]. Second, we formulate the problem of VNF instance selection for tenant-managed SFC on an abstracted multi-DC topology. Leveraging an auxiliary layered graph, the problem is formulated as a weight-constrained shortest path problem.

2.1. Network scenario

The ETSI specifications [1, 33] provide a reference architecture for VNF-based network service deployment and management, whose main entities are as follows: *(i)* Virtual Infrastructure Managers (VIM) that manage physical and virtual software resources of NFV Infrastructures (NFVI); *(ii)* VNF Managers that handle the lifecycle of VNFs; and *(iii)* the NFV Orchestrator (NFVO) that manages the lifecycle of network services.

As regards VNF-based service chaining, ETSI distinguishes two main architectural options [28]:

- *SFC defined in the NFVI*: service chains are managed by the infrastructure provider through infrastructure features (i.e., SDN Controller or VIM networking functions).
- *SFC defined in the tenant domain*: service chains are defined on top of a virtual network of nodes in the tenant domain. The tenant domain is thus an overlay network of VNFs and tenant-defined SFC can be realized through forwarding and control elements deployed as VNFs as well.

To clarify these concepts, we refer to the SFC architecture elaborated by IETF [31, 32], which is divided into a control plane and a data plane. The control plane manages the chains, computes forwarding paths, called Service Function Paths (SFP), and consequently instructs SFC data plane functional elements on how to process packets (i.e., classification and forwarding rules). The data plane is composed of:

- *Service Functions*: functions responsible for specific treatment of received packets (in this work we can suppose them equivalent to VNFs). Multiple instances of different function types can exist in the same administrative domain.
- *Service Function Path (SFP)*: it is a constrained specification of where packets should go.
- *Classifier*: an element that performs flow classification.
- *Service Function Forwarder (SFF)*: it is responsible for forwarding traffic to one or more connected functions and handling traffic coming back from these connected functions, to route packets according to the specified Service Function Path.

Although IETF and ETSI have not officially agreed on a shared terminology, they provide complementary architectural views and, for the sake of conciseness, we refer to [28, 13] for detailed definitions and agreement on terms.

Considering both the above mentioned ETSI and IETF specifications, in Fig. 1 we depict a reference architecture for tenant-based SFC in a multi-DC environment. The tenant overlay network is a virtual network spanning multiple DCs. Each DC can be a separate administrative domain and be managed by a dedicated VIM, but we assume a single NFVO is in charge for the setup of the tenant virtual network and the allocation and configuration of associated resources (e.g., VMs and virtual links). We also assume that in the tenant network multiple instances of different VNF types have been deployed.

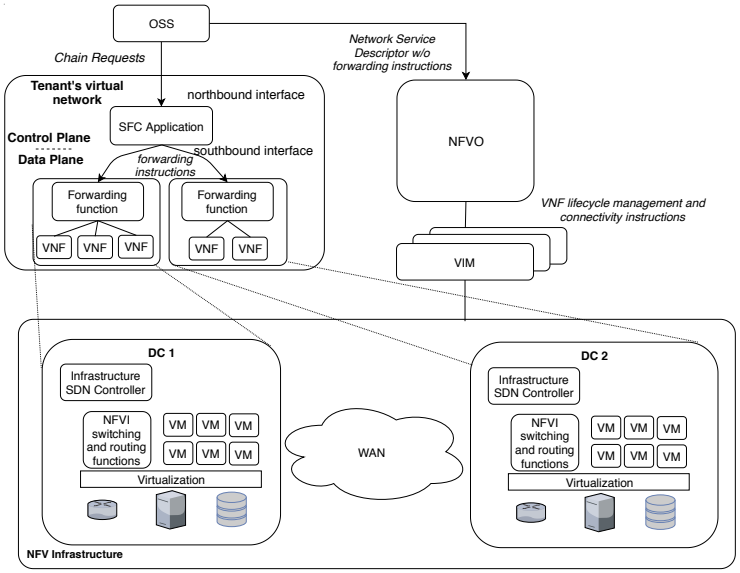

Figure 1: Tenant-defined SFC architecture in a multi-provider virtualized infrastructure

An *SFC Application* is the control element in charge of handling incoming service chain requests from the operator's operations and business support system (OSS/BSS). It selects the VNF instances to realize the requested chain and program the *Forwarding Functions* to steer the traffic across selected VNF instances through appropriate forwarding instructions. SFC requests are specified by clients as abstract chains and include information required for flow classification (e.g., source and/or destination IP and address, protocol type, etc...), a chain specification in terms of ordered sequence of VNF types and QoS requirements (e.g., maximum end-to-end latency, minimum bandwidth). The *SFC Application* takes its decision on service chain request feasibility and VNF instance selection using an abstracted view of the underlying topology. We assume that monitoring information related to elements of the abstract topology can be collected from the VNFs or from the VIMs. The VNF Selection decision is taken by the SFC Application leveraging the VNF Selection algorithm described hereafter.

It is worth highlighting that the IETF specifications do not recommend any

specific control protocol or technology, thus assuming the separation of control and data planes but not forcing the adoption of neither SDN controllers nor OpenFlow specifications [31, 32]. Moreover, these specifications do not define how much specific a SFP should be, allowing the control plane to specify a level of indirection between a fully abstract notion of service chain (i.e., a sequence of abstract service functions) and the fully specified notion of exactly which SFF/SFs the packet will visit when it actually traverses the network. Leveraging such possibility of defining service chain specifications at different abstraction levels, in Section 4 we will describe our implementation choices for the REST intent-based north- and south-bound interfaces of the *SFC Application*.

2.2. VNF Selection Problem statement

The problem treated in this work consists in the setup of a service chain within a virtual tenant network spanning multiple DCs. We assume that multiple instances of different types of VNFs are pre-deployed in the tenant-based virtual network and hosted at multiple and geographically distributed sites. SFC requests are specified by clients at a high level of abstraction. We cannot assume that detailed information of the underlying infrastructure is available at the tenant level, since different administration domains are concerned (as discussed also in [10]). We therefore handle a service chain request by modelling a VNF selection problem on top of an abstracted topology.

This abstracted view models DCs as nodes. Each DC node has a list of VNF instances available and their parameters (i.e., VNF type, processing capacity, measured traffic input rate).

Starting from a realistic network configuration where nodes represent forwarding, storage or compute elements in DCs, and links connect such nodes, we build an abstract network $G = (D, E)$ where nodes represent DCs and each arc $(i, j) \in E$ between DC i and DC j represents a path in the original network between nodes i and j. Specifically, arc $(i, j) \in E$ corresponds to the path with minimum latency among all the paths connecting nodes i and j in the original network. All arcs belonging to E are bidirectional. W is the set of VNF types

available for the whole network. The notation used is summarized in Table 1.

Table 1: Summary of notation

Sets	
D	set of nodes in the abstract network (each node corresponds to a DC)
E	set of arcs in the abstract network
	arc (i, j) corresponds to a path from DC i to DC j
W	set of VNF types available in the network
Node parameters	
V_i	set of VNF instances deployed on DC i
Arc parameters	
l_{ij}	network latency of arc (i, j) expressed in ms
b_{ij}	available bandwidth of arc (i, j) expressed in Gbps
Parameters of VNF instance v	
$\tau_v \in W$	the type of VNF instance v
μ_v	processing capacity of VNF v (Mb/s)
Λ_v	current load as the sum of flows on VNF instance v (Mb/s)
T_v	average processing delay of VNF instance v (ms)
Request parameters	
o^r	origin node of traffic request r - ingress node
d^r	destination node of traffic request r - egress node
$H^r = \{w_1^r, w_2^r, .., w_{\lvert H^r \rvert}^r\}$	ordered sequence of VNF types composing r
	($\lvert H^r \rvert$ is the length of the chain and $w_h^r \in W$ is the type of h-VNF in request r)
l^r	maximum end-to-end delay tolerated by r
b_{hh+1}^r	minimum data rate capacity (bandwidth) accepted by r
	from the h-th VNF to the $(h+1)$-th VNF

2.2.1. Data Centers and VNF instances

We assume DC i can run a subset V_i of VNF instances with type in W. Overall, each DC is represented as a container of VNF instances of different types. The performance of each VNF instance v depends on several factors, e.g., software implementation, CPU, memory, disk space resources assigned to the virtual machine running v, etc.. In this work this information is abstracted and provided as a derived metric, i.e. a nominal processing data rate (called processing capacity).

We assume VNF instance v is characterized by its load Λ_v, i.e. given by the sum of currently active flows that cross that instance. We estimate the

delay experienced by a packet processed by that VNF instance by applying a well-known queuing theory model. We model VNF instances as $M/M/1$ queues. Given VNF instance v, its processing capacity is denoted as μ_v. Arrival rates of packets belonging to a flow s are modelled as Poisson processes with rate λ_{sv}. In a given instant in time, if a VNF instance is crossed by a set S of flows, the total arrival rate of traffic entering VNF instance v is defined as Λ_v where

$$\Lambda_v = \sum_{s \in S} \lambda_{sv}. \tag{1}$$

We suppose that flows are conditionally accepted and served so that all VNF instances remain in a stable condition, i.e. given the utilization factor

$$\rho_v = \Lambda_v / \mu_v \tag{2}$$

the following condition holds:

$$\rho_v < 1. \tag{3}$$

If the overall request demand undermines this assumption, we suppose that appropriate scaling and migration strategies have to be put in place by infrastructure operators for maintaining the nominal VNF processing capacity.

For FIFO (first in, first out) and PS (processor sharing) disciplines, the average processing delay T_v of VNF instance v is modelled as

$$T_v = 1/(\mu_v - \Lambda_v). \tag{4}$$

2.2.2. Inter-DC Network

We assume that each arc $(i, j) \in E$ is characterized by the latency l_{ij} of arc (i, j) expressed in ms and b_{ij}, available bandwidth of arc (i, j) expressed in Gbps.

Latency is due to the propagation, transmission and queuing delay. The minimum value of the latency between a pair of DCs i, j is thus derived in this work by accounting for the propagation delay to travel the distance between DC locations plus a penalty for each hop encountered in the path from i to j [34] to

take into account of transmission and queuing delays. In Section 3, we report with more details on the delay model used to generate network delay samples for simulations.

Due to network abstraction, the bandwidth of an arc (i, j) is the minimum bandwidth over all the links on the minimum latency path from i to j in the original network. Leveraging the discussion in [35, 36] we suppose that bandwidth and latency values related to inter-DC connections are made available by NFVI providers through appropriate APIs or are measured within the tenant domain.

2.2.3. Service Chain Request

Each service chain request r is characterized by the parameters described in Table 1, namely the origin and destination nodes (o^r and d^r), the maximum tolerated end-to-end delay (l^r), the ordered sequence of VNFs composing it (H^r) and the minimum bandwidth required b^r which could be specific for each connection between pair of VNFs.

Given a request chain r to be accommodated in the system, the problem consists in choosing the VNF instances that can realize the given chain so that the estimated end-to-end latency is minimized. End-to-end delay is estimated by summing processing delays at each VNF instance and network delays of each edge realizing the chain (i.e., edge between the source node and the DC hosting the first VNF instance, edge between each pair of consecutive VNF instances, and the edge between the final VNF instance and the destination node). The processing delay is calculated according to eq. (4) considering also the load provided by request r itself. The problem is mathematically formulated in Section 2.3.

2.3. VNF Selection optimization algorithm

In this section, we model the VNF selection problem accounting for the extra delay that each new request generates on previous chains. This model is referred to as *SelSFC-P*. Model *SelSFC-P* uses the auxiliary layered graph defined in

[29]. Specifically, the layered graph allows to guarantee that the VNFs belonging to a request r are selected in the order specified by the chain representing the request. Thus, the layered graph has a layer for each VNF in request r where layer h corresponds to the h-th VNF with type w_h^r in the chain, and each layer consists of all the DCs on which an instance v of type w_h^r (i.e., $\tau_v = w_h^r$) has been deployed. In addition to layers $h \in \{1, \ldots, |H^r|\}$ where layer h corresponds to the h-th VNF in the request r, the layered graph has two extra layers: a first layer (namely, layer 0) hosting the origin node o^r of request r and a last layer (namely, layer $|H^r| + 1$) hosting the destination node d^r of request r. Arcs in the layered graph link (i) the origin node in the first layer with each DC in the layer corresponding to the first VNF in the chain, (ii) each DC in the layer corresponding to the last VNF in the chain to the destination node in the last layer of the graph, (iii) each DC in an intermediate layer h with each DC in the next layer, namely, $h + 1$ for each layer h in $\{1, \ldots, |H^r| - 1\}$. Thus, for each request r, a path in the layered graph from origin node o^r to destination node d^r, by construction, visits exactly one node in each layer, and the node visited in each intermediate layer h with $h \in \{1, \ldots, |H^r|\}$ identifies the DC from which an instance of VNF in position h of the chain has been selected.

Table 2 gives the additional notation required to take into account the extra delay that the management of request r generates on the set of flows S (indexed by s) still active in the network when request r is considered. As detailed hereafter, the problem is formulated so that this extra delay does not make already active requests violating their maximum latency constraint.

Specifically, according to assumptions and notations introduced above, the extra delay e^{sr} that request r generates on request s is defined as follows:

$$e_v^{sr} = \frac{1}{\mu_v - \sum_{s \in S} \lambda_{sv} - \lambda_{rv}} - \frac{1}{\mu_v - \sum_{s \in S} \lambda_{sv}}$$

Model *SelSFC-P* makes use of the following binary variables:

Table 2: Additional notation

S	set of requests still active when request r is considered
I_{rs}	set of VNF instances belonging both to r and s, $\forall s \in S$
p_v^r	position that the type of VNF instance v occupies in request r
d_v^s	DC on which VNF instance v has been selected in request s
δ^s	residual delay that request s can tolerate
e_v^{sr}	extra delay VNF v of request r generates on request s

$$\mathrm{x}^r_{ihjh+1} = \begin{cases} 1 & \text{if arc linking DC } i \text{ in layer } h \text{ to DC } j \\ & \text{in layer } h+1 \text{ is in the path of request } r \\ 0 & \text{otherwise} \end{cases}$$

defined $\forall i \in D \cup \{o^r\}, j \in D \cup \{d^r\}, h \in \{0\} \cup \{1, \ldots, |H^r|\}$.

$$\min \sum_{h=0}^{|H^r|} \sum_{i \in D \cup \{o^r\}} \sum_{j \in D \cup \{d^r\}} c_{ij}^{h+1} \cdot x_{ihjh+1} \tag{5}$$

$$\sum_{j \in D} x^r_{o^r 0 j 1} = 1 \tag{6}$$

$$\sum_{j \in D} x^r_{j|H^r|d^r|H|^r+1} = 1 \tag{7}$$

$$\sum_{j \in D \cup \{o^r\}} x^r_{jh-1ih} - \sum_{j \in D \cup \{d^r\}} x^r_{ihjh+1} = 0 \qquad \forall i \in D, \forall h \in \{1, \ldots, |H^r|\} \tag{8}$$

$$\sum_{v \in I_{rs}} e_v^{sr} \sum_{j \in D} x^r_{jp_v^r - 1 d_v^s p_v^r} \leq \delta^s \qquad \forall s \in S \tag{9}$$

$$x^r_{ihjh+1} \in \{0,1\} \qquad \forall i \in D \cup \{o^r\}, \forall j \in D \cup \{d^r\}, \forall h \in \{0, \ldots, |H^r|\} \tag{10}$$

The objective function (5) minimizes the total cost of the selection which takes into account latency and processing time. Specifically, the cost c_{ij}^{h+1} of an arc between node i in level h and node j in level $h+1$ is given by the sum of the latency l_{ij} between DC i and DC j, and the processing time of VNF v,

with type specified in position $h+1$ of the chain, when selected from DC j; namely, $c_{ij}^{h+1} = l_{ij} + T_v$ with $v \in V_j$ and $\tau_v = w_{h+1}^r$ (for all the arcs entering the destination node d^r the relative cost accounts only for latency). Constraints (6), (7), and (8) are the classical flow conservation constraints that define the path for request r on the layered graph from the origin node o^r to the destination node d^r. Specifically, they guarantee that exactly one arc is selected among those outgoing from the origin node (6), exactly one arc is selected among those entering the destination node (7), and for any DC node in each intermediate layer of the network, they assure that the ingoing flow equals the outgoing one (8). Constraints (9) guarantee that for each request s still active when the selection problem for request r is addressed, the extra delay e^{sr} that the selection of any VNF v which is in common between s and r (i.e., in I_{rs}) would cause if it were selected, for request r, on the same DC from which it has been selected for request s, namely d_v^s, does not exceed the maximum residual delay δ^s. Finally, constraints (10) restrict the variables to be binary.

For each request, the feasibility check concerning bandwidth and latency is done respectively in the pre-processing and post-processing phases. Specifically, during the pre-processing phase we impose that variable x_{ihjh+1}^r is fixed to zero (the corresponding arc cannot be used in the auxiliary graph) if the bandwidth b_{ij} of link (i,j) is smaller than the minimum data rate capacity (bandwidth) from the h-th VNF to the $(h+1)$-th VNF accepted by r, namely b_{hh+1}^r. Moreover, after the processing of request r, during the post-processing phase, we check that the end-to-end delay experienced by request r does not exceed the maximum tolerated end-to-end delay l^r. If the check fails, request r is discarded.

In the next sections, model *SelSFC-P* is compared with a simplified version of it, namely model *SelSFC*, in which constraints (9) in charge of controlling the extra-delay on still active flows are relaxed [29].

3. Simulation

In this section, we present extensive simulation results to evaluate the proposed VNF selection algorithm and compare it with alternative selection strategies.

3.1. Simulation Settings

We developed a simulator in Java and used ILOG CPLEX 12.8 to solve the *SelSFC* and *SelSFC-P* problems formulated as an ILP. An AMD A10-7850K Radeon R7 12 Cores machine with 32 GB RAM running Ubuntu LTS 16.04 was used to run simulations. We considered the Pan-European network topology (28 nodes), whose topological parameters have been gathered from the literature [37].

The distribution of VNF instances and related attributes (VNF type and processing capacity) have been configured with different settings, which are described at the beginning of each experiment.

Network latencies between pairs of DCs are sampled from a translated gamma distribution $\Gamma(\alpha, \beta)$ [38] in order to obtain most of the latency values close to the minimum one with some rare peaks. The minimum latency value between a pair of DCs (i, j) is derived considering the propagation delay between the two DCs plus a penalty for each hop encountered in the path from i to j to account for queuing and transmission delays [34]. The minimum distance path is calculated using the Dijkstra algorithm. The propagation speed through an optical link is approximated to $204Km/s$.

The interarrival time between requests is sampled from an exponential distribution $\exp(\lambda)$ with rate λ= 1/5 req/unit time. Request duration, called hereafter time to live (*ttl*), follows a normal distribution $\mathcal{N}(\mu, \sigma^2)$ where μ is the mean (called *ttl_mean*) and σ^2 is controlled through a parameter called *ttl_std*. Each SFC request specification contains source and destination nodes, an ordered sequence of VNF types, maximum allowed latency and minimum bandwidth. SFC requests are generated as follows: source and destination nodes are

uniformly distributed among the set of nodes, the chain length is a configurable parameter that is set according to the experiment type as detailed hereafter. We suppose that at most 10 VNF types are available in the network. For each position in the chain, the VNF type is randomly selected from the range [1,10] and repetitions are not permitted (although managed by the algorithm). The minimum bandwidth can be a fixed value or uniformly distributed within a given range, as detailed for each experiment.

Random generation of different parameters use seeds that are on their turn randomly generated from a unique seed (primary seed). This allows to easily repeat sets of experiment with the same configuration. On the other side, choosing a seed implies generating a specific set of SFC requests sent to the system. This may influence the experiment results, especially if simulations run with an insufficient number of requests. With sets of 1000 requests, we experimented that changing the primary seed leads to a variation in the measured metrics of 1% ca., which we consider acceptable.

3.2. Performance metrics

The evaluation focuses on the following set of metrics:

- *End-to-end latency*: latency from a source node to a final destination traversing the VNF chain.
- *Acceptance Rate*: ratio between the number of accepted requests and the total number of requests sent within a given time period.
- *Computation time*: time required by the optimization algorithm for solving a VNF selection problem instance.
- *Distribution of Request Load across DCs*: how fairly the request load is spread across different DCs.

Two alternative VNF selection strategies have also been implemented for performing a comparative evaluation: a greedy approach (referred to as Greedy) and a round robin one (referred to as RR). Greedy works by simply choosing

VNF instances with lowest processing time, without considering inter-DC network latency. In the RR strategy, DCs offering instances for a given VNF type are selected in turn. While we run both algorithms in all experiments, results obtained with Greedy have been reported for all experiments, while results gathered by RR are discussed only for experiments aiming at measuring load distribution across DCs, where RR is a reference baseline.

3.3. End to end latency of selected chain

Hereafter, we discuss tests performed for evaluating the end-to-end latency of accepted chains with increasing request loads. In each test 10,000 consecutive requests are sent to the system. The load on the system (represented with an *average DC load* metric calculated as the average of the load across all DCs in the topology) is increased by varying for each test the values of *ttl_mean* in the range [1,000, 7,000] with step 1000 so that we increase the number of requests active on average in the system at a given time instant. The value of *ttl_std* is kept constant at 30. The maximum latency is kept unbounded, so that no requests are refused due to the maximum latency constraint (in such case *SelSFC* and *SelSFC-P* lead to the same solution and therefore for this experiment we use the term *SelSFC* to refer to both algorithms).

The network of DCs has been configured as follows: the capacity for each VNF type has been assigned randomly in the range [0,80] with a step of 20 in each DC; the chain length is randomly picked from the range [2,7]. The simulation settings are configured so that the acceptance rate has minimal impact on the tests, in fact we observed that only a very small fraction of requests is refused due to VNF capacity constraints (3% in the test with *ttl_mean* set to 7,000).

In order to better clarify the simulation settings, Fig. 2 shows how the average DC load increases with the *ttl_mean* value, i.e. the average duration of a request. The figure also eases clarifying how the average DC load increases with time ticks, i.e., as far as requests are submitted to the system. After an initial transient phase characterized by an increasing trend of the average DC

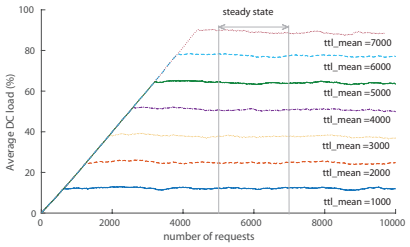

Figure 2: Experiment settings: average DC load vs number of consecutive requests. A steady state is selected between requests no. 5000 and no. 7000.

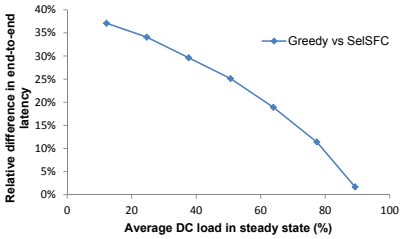

Figure 3: Comparative evaluation of end to end latency in steady state - Greedy vs *SelSFC*

load, this value remains almost stable until the end of simulation (the number of completed requests is balanced by new ones). We therefore analyse the test results considering only the interval encompassing the delivery of the 5000th request to the 7000th one, which we consider as a steady state (average DC load almost stable).

Fig. 3 compares the relative difference in end-to-end latency obtained by Greedy vs *SelSFC* against average DC load considering only the steady state. Obtained results clearly show that *SelSFC* achieves better performance in terms of end-to-end latency with respect to Greedy. When the average DC load value increases from 40% to 80%, the relative difference in end-to-end latency obtained by Greedy goes from 30% to 10%. Such improvement is inversely proportional to the DC load, since processing latency increases with DC load, while the impact of network latency on end-to-end latency is progressively eroded.

We also observed how latency varies with the chain length [2,10] and request bandwidth values (1,5,10Mb/s). Fig. 4 shows that *SelSFC* achieves smaller average latency values, but this gain on Greedy progressively decreases with increasing bandwidth, due to the increasing load on DCs. In particular, we can observe that with low required bandwidth values (1 Mb/s) the improvement brought by optimization increases with the chain length, since the impact of

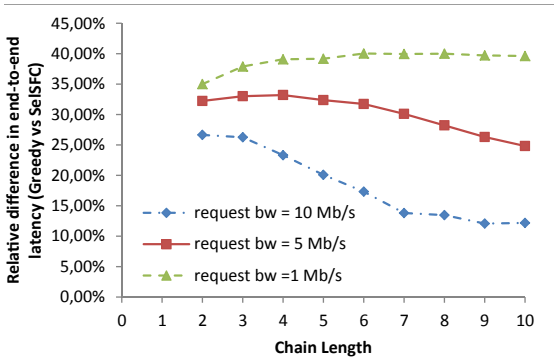

Figure 4: Comparative evaluation of end to end latency vs chain lenght - *SelSFC* vs Greedy

inter-DC network latency on end-to-end latency increases with chain length. On the contrary, with higher bandwidth values (10 Mb/s) the processing delay provides a higher contribution to the end-to-end latency with increasing chain length. An intermediate behaviour is shown with flows of 5 Mb/s.

3.4. Acceptance rate

Acceptance rate has been evaluated with the following simulation settings. For each DC the capacity for each VNF type has been assigned randomly in the range [0,80] with a step of 20. The request set has been generated with a *ttl_mean* value set to 1000 and the *ttl_std* value equal to 300 in order to have a high variance in request duration and a bandwidth value with a uniform distribution in the range [1,10]. For each simulation (consisting of 1000 consecutive requests) we configured the chain length with a fixed value in the range [2,10]. First we run a test with unbounded maximum latency, so that only processing capacity constraints are taken into account in accommodating requests. Then, we run a sequence of tests progressively decreasing the maximum latency value (1000, 800, 600, 400 ms). Figg. 5 6 and 7 show the overall acceptance rate for *SelSFC*, *SelSFC-P* and Greedy, respectively.

When the maximum latency requirements is not specified, all algorithms show a similar behaviour in the way acceptance rate decreases with chain length (and consequently with the request load). By imposing more stringent latency

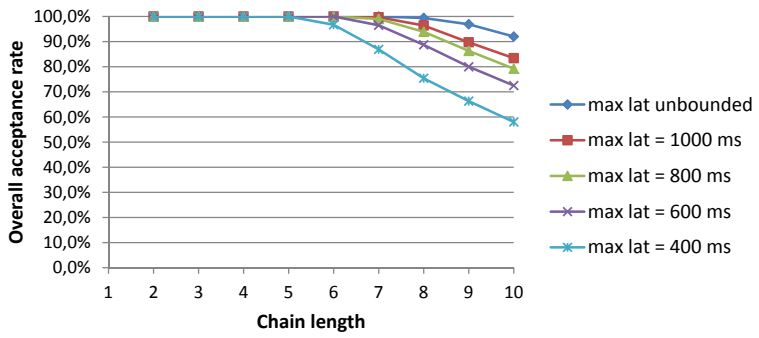

Figure 5: Acceptance rate with *SelSFC*

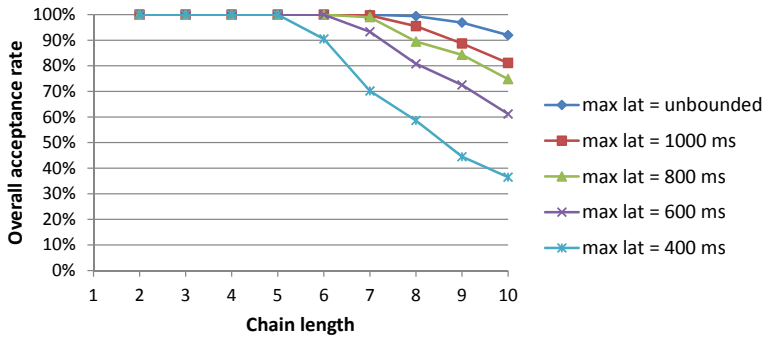

Figure 6: Acceptance rate with *SelSFC-P*

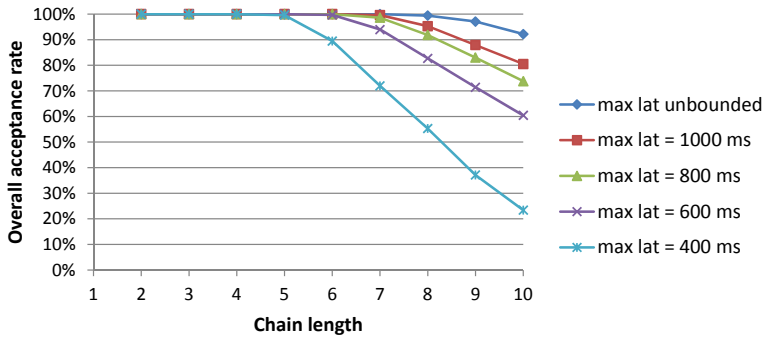

Figure 7: Acceptance rate with Greedy

requirements, *SelSFC* accepts more requests than *SelSFC-P* (e.g., 58% vs 36.5 % with 10-long chains) since it guarantees the fulfilment of latency requirement only for new requests, and therefore a number of latency violations for already accepted chains may occur as new requests are accommodated (as shown in Fig. 8). Greedy and *SelSFC-P* show an almost similar behaviour for higher latency values, but *SelSFC-P* clearly outperforms Greedy when the maximum latency value is set to 400 ms.

End to end latency is calculated considering also the inter-DC network delay, which may fluctuate over time and it has more impact on the end-to-end latency as far as the chain length increases since the chain path will more probably cross different DCs. We therefore evaluated the number of violations in already accepted chains that *SelSFC* allows when the maximum latency constraint is 400 ms by keeping the network latency value constant (equal to the average value). Results are shown in Fig. 8. The percentage of chains that during the simulation cycle violates for a while the latency constraint is calculated over the number of accepted chains. The amount of latency violations is below 15% of accepted chains for chains up to 7 VNFs long. For longer chains, the number of violations increases. However, with constant network delays, the difference in acceptance rate between *SelSFC* and *SelSFC-P* decreases and goes below 5%.

Analogous tests have been performed by varying how VNF type availability and capacity are distributed across DCs. Obtained results confirm the trend previously discussed and shown in Figg. 5 6 and 7 and therefore they are not reported here. For the sake of completeness, we also evaluated the acceptance rate of the RR strategy, which, however, is radically outperformed by both *SelSFC* and *SelSFC-P* algorithms, since it does not apply any resource orchestration criterium. We therefore omit these results.

3.5. Computation time

We used the configuration described in Section 3.4 to evaluate the time complexity of *SelSFC* and *SelSFC-P* against chain length. Fig. 9 shows the average computation time calculated on 100 requests with maximum latency

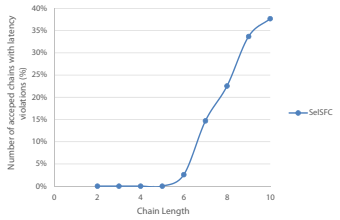

Figure 8: Violations of maximum latency constraint after acceptance (percentage over the number of accepted chains)

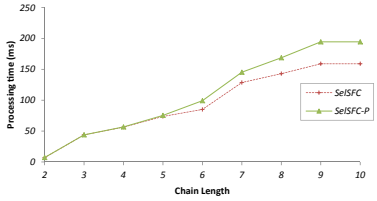

Figure 9: Computation time vs chain length

set to 600 ms. When the load on the VNFs is low, the maximum latency constraint is easily fulfilled, also for already placed requests and the computation time of the two algorithms is very close (chain length < 5). When the load increases due to the chain length, the fulfilment of maximum latency constraints for already deployed chains is more challenging for *SelSFC-P*, thus requiring a higher computation effort.

3.6. Distribution of load across DCs

Hereafter, we describe the tests performed to evaluate how fairly *SelSFC* and *SelSFC-P* distribute the request load with respect to the overall processing capacity offered by DCs.

We calculated the overall capacity of a DC as the sum of processing capacity at all VNF instances available in that DC, and at each iteration DC load is calculated as the ratio between the sum of load assigned to all VNF instances in that DC and the DC overall capacity. We take the RR algorithm as a baseline, since it provides a quite fair distribution even for relatively small loads. The DCs have been configured so to have all VNF types available in each DC with a capacity in the set {50,80}. Capacity values have been assigned so that the following two conditions hold: 1) all DCs have the same overall capacity value and 2) all VNF types have the same total capacity over all DCs.

We ran a set of tests (1000 iterations each) by progressively increasing the

requests' load by varying requests' *ttl_mean* in the set {200,400,600,800,1000} and the bandwidth of each request {1,3,5}. The value of *ttl_std* has been set to 50. The maximum latency constraint has been kept unbounded so that no requests are refused for violating the latency constraints and the algorithms show the same acceptance rate, *SelSFC* and *SelSFC-P* thus show the same behaviour and therefore the term *SelSFC* is used to refer to both. At the end of the iteration cycle, we calculate the load on each DC averaged over all iterations and then calculate the standard deviation of this metric over all DCs. In Table 3, we report some representative experiment results for the case with minimum load (ttl=200 and bandwidth = 1 Mb/s and a resulting relative DC load equal to 1%), medium load (ttl=600 and bandwidth = 3 Mb/s and a resulting average relative DC load equal to 8.6%) and a higher load (ttl=1000 and bandwidth = 5 Mb/s and a resulting average relative DC load equal to 22.82%). The table clearly shows how the relative standard deviation of the average DC load decreases with increasing load for all algorithms. As expected, *SelSFC* distributes loads less fairly than Greedy and RR (load distribution is not explicitly handled in the optimization model), but this difference decreases with increasing request load.

Table 3: Distribution of load across DCs

	relative standard deviation (%)		
Request set parameters	***SelSFC***	**Greedy**	**RoundRobin**
ttl=200, bw = 1 Mb/s	67.85	19.17	1.51
ttl=600, bw = 3 Mb/s	25.95	6.59	0.60
ttl=1000, bw = 5 Mb/s	9.01	4.54	0.45

4. Experiment setup

Hereafter we describe the environment used for experimentally evaluating a latency-aware SFC application using our proposed VNF selection algorithm in

a realistic scenario built on top of a European multi-site testbed offered by the 5GINFIRE consortium [27, 39].

5GINFIRE offers an NFV-enabled experimental platform that implements Management and Orchestration (MANO) functionalities for automated deployment and lifecycle management of experiment resources and related network services. The orchestration service is based on Open Source MANO (OSM), which is a functional implementation of a MANO software stack compliant with the ETSI specifications [40].

More specifically, the experiment aims at evaluating the capability of the SFC application in: *i)* elaborating service chain requests by computing the latency-optimized VNF chains leveraging the VNF Selection algorithm and an up-to-date knowledge on the abstracted infrastructure status obtained through monitoring, and *ii)* setting-up VNF chains across network and cloud domains by properly interacting with the forwarding functions in the tenant domain.

Hereafter, we describe the resources used for the experiment setup and the software purposely developed and deployed on the testbed infrastructure.

First, we introduce the software applications developed for supporting the SFC capability: the SFC application in the control plane, called *SFCLola application*, and the forwarding function in the data plane, *Virtual Flow Forwarder* (VFF), and, then, tools that have been developed for monitoring and testing purposes.

4.1. SFCLola application

SFCLola is a Java application that exposes RESTful operations for creating, retrieving, updating or deleting a service chain. When a new service chain request arrives, the application executes the VNF Selection algorithm, fed with an up-to-date view of the abstracted topology. Such view is built by SFCLola by periodically interacting with the underlying infrastructure (i.e. VIMs) and VNFs (or VNF Element Managers) to collect the following measurements and deployment information: inter-DC network latencies, VNF types and instances deployed at each DC and related processing latency and capacity. As output,

the algorithm provides a solution or a not feasible solution reply. If a solution is given, an estimated end-to-end latency is also provided which is then compared to the maximum latency constraint. If the constraint is fulfilled, the application produces appropriate instructions for the affected DCs to setup the chain and program traffic forwarding capabilities accordingly, otherwise the request is not accepted. If the solution is composed of VNF instances placed in more than one DC, SFCLola sends to each DC a sub-chain specification composed by: flow classification information, the sub-chain of VNF instances of that DC and appropriate ingress and egress connection points towards external DCs if the flow does not originate/terminate in that DC. SFCLola implements multiple interfaces to provide these instructions to the infrastructure: through standard APIs, such as OpenStack SFC APIs as well as custom intent-based SFC APIs as described in [41, 42]. However in both cases SFC APIs are supposed to be offered by NFVI providers. Since this work focuses on SFC in the tenant domain, in the following we introduce the VFF and describe its interface with SFCLola.

4.2. Virtual Flow Forwarder

Traffic steering is an essential part of SFC since it consists in directing the traffic to reach the intermediate VNF instances that realize a specific service function chain [14]. The survey by Hantouti et al. [14] provides an overview of main existing approaches, classified into three categories: (i) based on packet headers, (ii) based on packet tags and (iii) based on SDN programmable switching capabilities. However, these approaches typically rely on infrastructure-level network control capabilities, while this experiment refers to the scenario described in Fig. 1, i.e., SFC defined in the tenant domain. Indeed, the NFV-enabled testbed implementing OSM neither offered dynamic service chaining capabilities to tenants nor provided disclosure of VIM-level network programming capabilities (e.g. OpenStack SFC APIs) for security reasons. Recently, SFC approaches based on Segment Routing (SR) [43]. SR allows specifying how packets should be forwarded and processed by adding a sequence of segments in

the packet headers. Although SR is a promising solution for SFC traffic steering, it has been only recently standardized [44] and available implementations present some limitations, for instance [45] makes the strong assumption that a VNF can be used only for one chain.

We therefore implemented a forwarding mechanism for SFC in the tenant domain that exploits Linux policy routing capabilities and manages traffic flow routing within the tenant data plane, without relying on management capabilities at the NFVI level. In practice, a VM acts as an SFC-aware proxy that steers traffic across VNFs of target chains.

We implemented this capability as a Java application, called Virtual Flow Forwarder (VFF), that offers intent-based REST API to receive chaining instructions and translates them into flow forwarding rules enforced through Linux policy routing capabilities. According to the intent-based abstraction, implementation details are hidden to external clients and received commands are translated into actual forwarding rules by the VFF. Dynamic traffic steering is realized combining Linux policy routing [46] and the packet marking capability provided by the netfilter framework to define custom routing strategies in addition to the traditional destination-based one.

Forwarding rules are specified as follows. As a prerequisite, a set of routing tables are created, one for each VNF instance within that DC, that mandate forwarding to a given VNF IP address. Netfilter rules are used to mark flows matching a given combination of filters (e.g., flow classification information, incoming interface and/or source MAC address) and associate selected flows to specific routing tables through the *fwmark* parameter), so to create per-flow routing rules, and forward packets to the desired VNF. Given a chain specification provided by the SFCLola application, for each hop in the chain a forwarding rule is added in the VFF networking stack. The VFF manages the setup (or deletion) of this request by parsing the request and inserting (or deleting) the appropriate set of flow marking rules in the PREROUTING chain, which hooks incoming packets (see example below in Section 4.3) so to steer the flow through the VNF instances composing the chain.

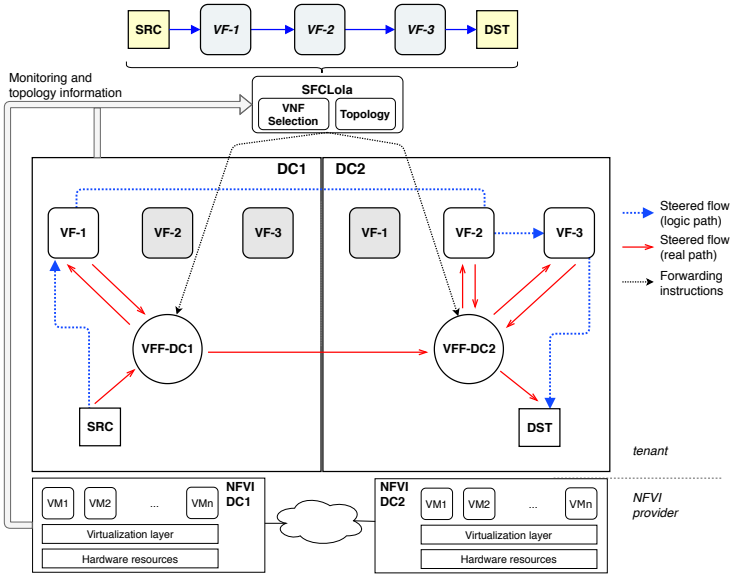

Figure 10: SFCLola Application and VFF for tenant-based SFC management in a 2-DC environment. Given a service chain request, SFClola defines a logical path across selected VNF instances (blue dotted line). Within each intra-DC virtual network, a VFF enforces a forwarding path (red line).

4.3. Example

Fig. 10 exemplifies how the two software work in a two-DC environment. The SFCLola application handles RESTful requests for chain management (north-bound interface). For the chain request shown in the figure, a possible solution is VF-1 instance in DC1 and VF-2 and VF-3 instances in DC2.

SFCLola sends appropriate forwarding instructions to each DC through its south-bound interface. The forwarding instructions dispatched by SFCLola define a logical path (blue dotted line in Fig. 10). According to the intent-based APIs offered by the VFF, SFCLola delivers forwarding instructions formatted as JSON messages to the VFF of each DC. The forwarding instructions specify the portion of path to be handled by the VFF in terms of: flow classification information, ordered sequence of VNF instances to be chained (identified by host's name or public IP), optional ingress and egress connection points towards VFFs in external DCs. Listing 1 shows the forwarding instructions sent to the VFF

in DC1.

```
{
    "source": "10.4.32.12",
    "destination": "10.154.8.115",
    "dest_port": 9000,
    "protocol": "udp",
    "vnfChain": "VF-1,DC2"
}
```

Listing 1: Example of intent-based forwarding instruction sent by SFCLola to VFF in DC1 to steer the traffic going from 10.4.32.12 to 10.154.8.115 to VF-1 and then to DC2.

Each VFF translates this request into a set of forwarding rules. These rules classify the traffic and mark the corresponding packets. Listing 2 shows the netfilter rules handling the chain hop from SRC to VF-1 in Fig. 10, which marks selected packets with a value "1" that has been previously associated with a routing table forwarding packets to VF-1.

```
-A PREROUTING -t mangle
 -s 10.4.32.12/32 -d 10.154.8.115/32 -p udp
 -m mac --mac-source FA:16:3E:5F:0A:84
 --dport 9000 -j MARK --set-mark 1
```

Listing 2: Example of netfilter rules to steer the traffic going from 10.4.32.12 to 10.154.8.115 and coming from the VNF with MAC FA:16:3E:5F:0A:84 through the VNF associated with mark 1.

4.4. Monitoring and experiment tools

Hereafter, we describe the tools that have been developed for supporting monitoring and testing activities.

A timeseries database based on Gnocchi has been deployed to acquire and store measurements and made them available to the SFCLola application and off-line analysis via REST APIs. Several components are used to gather monitoring information.

4.4.1. VFF monitoring

In addition to the steering function, the VFF is in charge of posting some useful information to the metric database including CPU load and traffic input

rate at the VFF node and inter-DC latency, periodically measured as a function of the RTT between a given VFF and the VFFs of the other DCs. Inter-DC network latency values are obtained measuring latency between each pair of VFFs each 10 seconds and averaging these values over a 30 second window. However, latency and system metrics polling intervals can be customized in the configuration file.

4.4.2. VNF Emulator

The VNF Emulator is a Java application that emulates the packet-delaying behaviour of a generic VNF. It introduces a delay that is proportional to the traffic rate and a configurable parameter (named Processing Capacity). Packet dropping and rate change behaviours were not in the scope of this work.

4.4.3. UDP_Ping

UDP_Ping is a tool developed in Python that measures the latency between two nodes, considering also possible intermediate nodes (i.e., a VNF chain). This tool periodically sends some probe packets (UDP datagrams) along the chain path. Optionally, this component can also be configured to produce traffic flows for a given service duration. In this case, the tool starts an iPerf3 client/server pair respectively on the selected source and destination endpoints. Traffic duration and bitrate can be specified in the request.

UDP_Ping on the source node periodically sends UDP probe packets to the destination node (ip source, ip destination and destination port must be the same of the chain request to be monitored). Probe packets' payload contains a timestamp injected by the source node at delivery time and flow identification information (e.g., destination port). The packets are steered through the VNF chain to the destination node and then sent back to the source node where the round trip time (RTT) is computed. Probe packets should thus be recognized as packets to be forwarded through the VNF chain but then, when they arrive at the destination node, they should be sent back to the source node. To do this, packets are sent using port 5002 as source port (configurable), this convention on

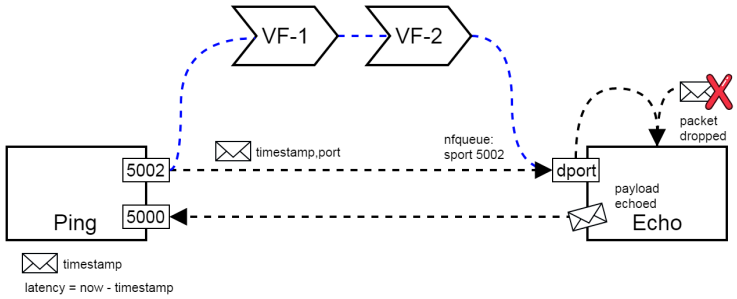

Figure 11: UDP_Ping components and operation for measuring the delay for delivering packets from source (Ping) to destination (Echo) through a chain

source port is used to filter UDP traffic at the destination side. The UDP_Ping component running on the destination node (called Echo) echoes packets with the matching source port back to the source node to the 5000 destination port. Figure 11 shows the overall UDP_Ping operation. At the source node, the payload is parsed and the RTT is computed as the difference between the time of packet reception and the timestamp stored in the packet payload. However, this RTT is composed of a delivery time from source to destination through the chain and a return time from destination to source with no intermediate VNFs. In order to estimate the delay through the chain UDP_Ping sends also probe packets with destination port 5002 (this port should not be used by SFCLola experiment flows). This allows measuring the RTT between source and destination node bypassing all the forwarding rules of the VFF. Half of this RTT is then subtracted from the previous RTT to estimate the one-way delay from source to destination through the chain.

4.4.4. VNF Proof

The VNF Proof has been developed in Python using nfqueue [47] and scapy [48] features for packet tampering to provide a proof of the service chaining operation. It is deployed on all VMs hosting VNFs and its main task consists in modifying a packet payload when the packet is received by the VM. More specifically, the application receives packets from nfqueue, tampers the payload by appending the hostname of the VNF instance to it and accepts the packet

letting the kernel forward it to the next hop.

4.5. Experiment resources and topology

The experiment software has been deployed on four different sites, owned by different providers: 5TONIC by Telefonica (organized in two different NFV/SDN domains, called 5TONIC and 5TONIC-bis in this work), ITaV in Portugal and the University of Bristol's 5G testbed (Bristol in brief). Sites are connected through a VPN overlay network. The VPN server is located at 5TONIC site and each site has associated its own credentials and IP address range. Through appropriate VNF and Network Service descriptors we submitted to the OSM orchestrator a request for 33 VMs, each VM with 2 vCPU and 1 GB RAM running Ubuntu 16.04. On each site, the VFF software was installed and configured in one VM, while another VM was used as an endpoint node (UDP_Ping and Echo installed). The remaining VMs were used to deploy VNF Emulator instances, configured to emulate different VNF types at different processing capacity, as shown in Table 4.

5. Experiment results

In this section, we report on the experiments run on the 5GINFIRE multi-site overlay network. First, we assessed the correct operation of the forwarding mechanism provided by VFF by using the VNF Proof module. We manually submitted a few chain requests through a Web GUI and verified the content of packets received at destination (tampered by crossed VNFs) with the one sent by the source node. In [30] we also evaluated CPU load at VFF due to chain installation and traffic steering by varying the number of flow forwarding rules and incoming traffic rate. Such impact appeared sustainable, although approximately proportional to traffic load and number of chains managed by a VFF.

Since the *SelSFC* algorithm uses a latency estimation for the new incoming chain computed by adding inter-DC network latency values and the estimated

Table 4: VNF configuration for the experiment

DC / VNF	5Tonic	ITaV	Bristol	5Tonic-bis
VNF-1	x		x	
VNF-2	x		x	
VNF-3		x	x	
VNF-4		x	x	
VNF-5		x		x
VNF-6		x		x
VNF-7	x			x
VNF-8	x			x
VNF-9	x	x		
VNF-10	x	x		
VNF-11			x	x
VNF-12	x	x		x
Capacity	100 Mbps	150 Mbps	80 Mbps	120 Mbps

processing latency at each VNF instance), we also performed a set of experiments to evaluate how good this estimation is with respect to measured values and network delays' variations in a realistic network scenario. As reported in [30], we carried out several tests and found an average estimation error of 3.8%.

In the following paragraphs, we describe the tests performed for evaluating: i) end-to-end latency of established chains and ii) load distribution achieved with respect to the Greedy and RR strategies. The maximum latency constraint has been set to a high value (1500 ms) so that all chains are accepted and the compared algorithms work to place the same sequence of requests. Under this condition, *SelSFC* and *SelSFC-P* behave in the same way, the reported results are thus named under the *SelSFC* strategy.

The experiment workflow consists in the following sequence of steps:

- We randomly generated a set of 20 chain requests, with a chain length in the range [2,4] and VNF types randomly picked up from the list of available ones.
- Requests are submitted to SFCLola with a 30 second-interval.
- Traffic flows are generated using the traffic generation feature provided by UDP_Ping leveraging iPerf3. Flows have a duration of 700 seconds.

The experiment is conducted by repeating the above described workflow ten times for each selection strategy. It is worth noticing that since the experiment is performed on a realistic multi-site testbed, the infrastructure status (especially network delay and bandwidth) may slightly vary at each iteration.

We compared the end-to-end latency obtained using *SelSFC* vs alternative approaches (Greedy and RR). Results, computed as the difference (in percentage) of measured latency values have been averaged over the requests in the set. Table 5 reports the results for each iteration (a negative value means that *SelSFC*, on average, obtains a lower latency compared with Greedy and RR). By averaging over 10 iterations, *SelSFC* outperforms Greedy and RR since it obtains a reduction in end to end latency of 13% and 26%, respectively. To further

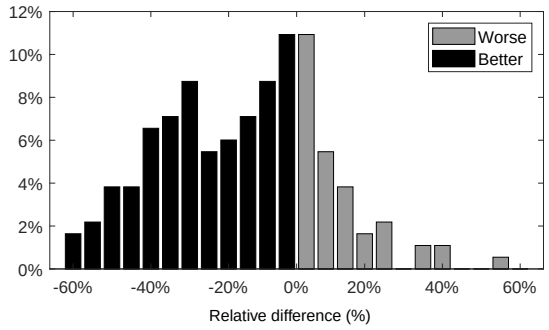

Figure 12: Histogram of the percentage deltas computed considering all the requests sent. *SelSFC* vs Greedy.

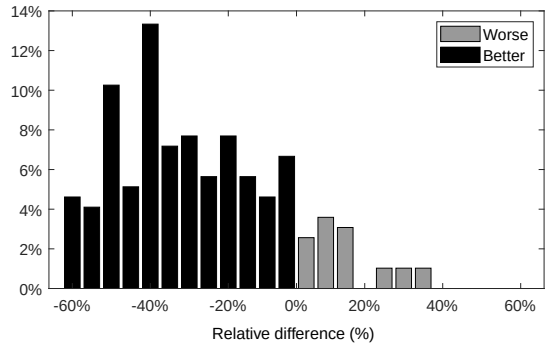

Figure 13: Histogram of the percentage deltas computed considering all the requests sent. *SelSFC* vs RR.

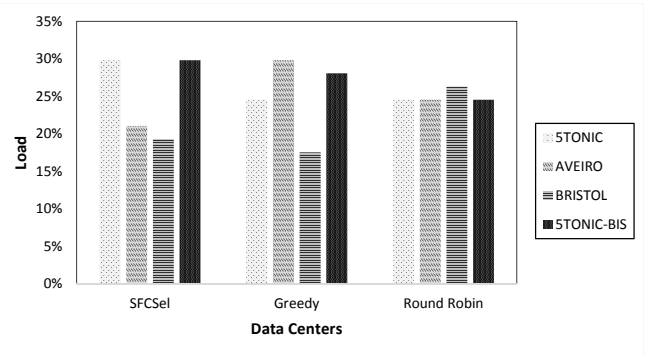

Figure 14: Load distribution across DCs

Table 5: End-to-end latency of established chains - comparative evaluation

	Average percentage difference selection vs Greedy	Average percentage difference *SelSFC* vs RR
1	-12.75%	-28.47%
2	-17.08%	-28.51%
3	-10.07%	-29.11%
4	-15.89%	-22.85%
5	-9.21%	-25.91%
6	-12.63%	-25.41%
7	-14.36%	-24.74%
8	-15.47%	-25.99%
9	-13.95%	-25.88%
10	-12.16%	-28.11%
Overall	-13.73%	-26.91%

consolidate these results, we run a set of simulation tests (using the simulation software and methodology described in Section 3) on a topology resembling the experiment one and mimicked the request sets used in the experiment. We performed 100 iterations and obtained quite similar results (-16% in end-to-end latency for *SelSFC* vs Greedy and -26% for *SelSFC* vs RR).

We also analysed how many times *SelSFC* provides a better solution than Greedy and RR. Figg. 12 and 13 show the distribution of relative difference samples (*SelSFC* vs Greedy and RR, respectively). As shown in Fig. 12, *SelSFC* obtained an improvement of latency (i.e., negative values) over Greedy in more than 75% of the requests. When *SelSFC* obtains a worse latency (i.e., positive

Table 6: Distribution of load across DCs

load distribution	***SelSFC***	**Greedy**	**RoundRobin**
relative standard deviation (%)	4.7	6.5	1.2

values), the relative difference in measured latency is mostly within the 0%-20% range. Fig. 13 shows that in 90% of the requests *SelSFC* provided a shorter end-to-end latency than RR did.

The fact that in some cases Greedy and RR perform better than *SelSFC* is expected. The reason is that all iterations start from the same VNF load configuration and a similar network status, but one request after the other the algorithms take different decisions, thus distributing the load across DCs in a different way. Therefore, even if the sequence of request chains is the same for the three algorithms, the infrastructure context gradually evolves differently.

Finally, we evaluated how the three different approaches distribute the overall VNF traffic processing load (i.e., the sum of traffic flows to be managed by each VF in the request set). Results for a test iteration are shown in Fig. 14. Of course, RR evenly distributes the load across DCs. Both *SelSFC* algorithm and Greedy quite fairly distribute the load. This is more evident looking at Table 6, which reports the standard deviation of the DC load for each strategy, i.e. it provides a measure of how DC loads sampled at each iteration are spread out from the mean value. Although *SelSFC* and Greedy are not specifically designed for load balancing purposes, these results are achieved since the processing delay varies proportionally to the traffic load and thus load balancing is achieved as side effect on the long run.

6. Related Work

The problem of how effectively deploying and managing network services conceived as VNF chains has raised a considerable interest in the research community. In this context, a huge literature has proposed strategies for resource allocation, which is typically distinguished into VNF chain composition, VNF chain embedding, and scheduling problems [49, 50].

In the area of VNF chain embedding, many works have proposed optimization algorithms that jointly address VNF placement and routing problems for embedding VNF chains into physical substrates to optimize QoS and/or cost

metrics. A survey on VNF and chain placement solutions is provided in [51]. Service chain embedding in a multi-domain infrastructure has been investigated by several authors, proposing centralized [18, 17] as well as distributed approaches [19, 20] for function placement and path selection.

On the contrary, only a few works have addressed the problem of VNF selection (i.e., selecting previously deployed VNF instances for realizing a service function chain).

In [21] the middlebox selection and routing problem is formulated as an integer programming model with the objective of maximizing the total throughput over all target flows. A Markov approximation based algorithm is proposed to solve the problem efficiently. The work in [21] considers the selection of one middlebox per flow and does not handle chains of middleboxes.

Medhat et al. [22] deal with the problem of selecting network function instances while creating a Service Function Path in a multi-DC environment and propose a heuristic that considers both load balancing across instances and latency requirements.

In [23] both offline planning and online routing problems in an SDN controller are considered. The first problem is formulated as a constrained version of the maximum concurrent flow problem, while a time dependent dual based online routing algorithm is proposed for the second problem. These algorithms are evaluated via simulation-based experiments on DC and ISP topologies. Both [22] and [23] consider only the network latency disregarding delay introduced by intermediate functions.

Ma et al. [24] specify the workflow and selection strategy of a SFC controller that targets maximum overall system efficiency metrics. The performance of the VNF selection strategy is evaluated in an emulated environment built with Mininet and OpenDayLight.

In [25] the problem of optimally selecting VNF instances and constructing the paths without violating the predefined orders is formulated as an ILP model that minimizes the resource consumption costs for flows. The problem is solved by transforming the original network into a Logical Function Graph (LFG)

with an approach similar to the one adopted by the authors in [29]. Also in this case processing delay introduced by the network functions is not considered in calculating the end-to-end delay.

Although some of these works refer to inter-DC scenarios (e.g. [22]) or ISP network topologies (e.g., [25, 21]), it is assumed that the SFC algorithm works using a detailed network topology and no abstractions are made. On the contrary, in this work we assume that network operators and network service providers are different entities and that the network service provider has thus access to an abstracted view of the underlying topology. Such assumption has been considered only in a few works focusing on VNF placement [52, 53] or on monitoring across multiple administrative domains [54], but, to the state of our knowledge, not yet in the VNF selection problem. In addition, in this work we evaluate the algorithm in a simulation environment as well as in a testbed, while most above mentioned works do either simulation or experimental validation.

As regards abstraction in API design, abstracted interfaces have been designed in several works for programming the data plane with chaining instructions. Davoli et al. [41] propose a reference architecture and an intent-based North Bound Interface for end-to-end service chain management across multiple technological domain, including anInternet of Things (IoT) infrastructure deployment, a cloud-based application domain, and a transport domain over a geographic network interconnecting the first two domains. In [55] a distinction is made between user intents and provider intents and a mechanism is proposed to decompose multidomain intent issued by users into local intent graphs that are compiled and installed in local regions. Both [41] and [55] deal with a problem different to ours, since, contrarily to our reference scenario, the VIM layer is assumed offering SFC capabilities.

Only a few works have investigated tenant-defined SFC. Wang et al. [9] propose a "Bring Your Own Controller" (BYOC) concept and implement it with BYOC-VISOR, an SDN platform providing cloud users with customized SDN services. An integrated SDN/NFV management and orchestration architecture for multi-tenant transport networks is proposed in [15]. This architecture is

implemented by deploying tenant SDN controllers as VNFs in DCs and assessed in a multi-partner testbed. However, both [9] and [15] implement tenant-side SFC management leveraging NFVI management features.

7. Conclusions

In this work, we tackled the problem of tenant-defined service function chaining on a multi-provider distributed virtual infrastructure. In order to cope with restrictions that infrastructure providers typically impose on a tenant's access to infrastructure monitoring and traffic steering APIs, this work provided the following main contributions.

First, we proposed a VNF Selection algorithm that relies on an abstracted network model thus minimizing topological and monitoring information required from the infrastructure. Our algorithm is formulated with constraints guaranteeing that already established chains do not violate maximum latency constraints as the processing delay at VNF instances increases with the request load. Simulation tests shows encouraging results and improvements in acceptance rate and end-to-end latency of accepted chains in comparison with alternative greedy and round robin strategies. In regards to efficiency, we evaluated how the computational time varies with the chain length, but remains within acceptable values for the simulation settings (chains of maximum 10 VNFs and a 28-node topology). Load distribution has not been considered as an optimization objective, but this aspect should be considered in a future evolution of the algorithm or a heuristic solution.

Second, we designed, implemented and evaluated in an experimental environment an SFC solution that provides an implementation of the SFC architecture elaborated by ETSI in [28]. It consists in an SFC application leveraging our VNF Selection algorithm and a forwarding mechanism allowing chain management and traffic steering control within the tenant domain. In this work, we describe the setup of the experiments performed on top of 5GINFIRE infrastructure facilities and the software components developed for such purpose.

The experiment allowed us demonstrating the operation of SFCLola application and the VFF forwarding mechanism and their interoperation. The comparison with alternative heuristic strategies has also been carried out in the experimental testbed. Although at a smaller scale, results mainly confirmed simulation results.

Regarding the algorithmic aspects, we plan to improve the model of end-to-end latency considering also transmission delays and a more fine grained processing delay model at VNFs. We will also further investigate approaches for building the abstracted network and related layered graph derived from the physical network topology to more robustly handle the dynamic change of topology characteristics (e.g., available bandwidth) and possible multiple virtual links between DCs. In addition to above mentioned planned activities, we also plan to extend SFCLola in order to support a QoS negotiation mechanism allowing to accept maximum SFC requests only partially accommodating maximum latency and minimum bandwidth requirements, to improve acceptance rate and optimize resource usage. The forwarding mechanism implemented in the VFF application has the advantage of using stable technologies available in standard Linux distributions, although some limitations are due to the actual deployment realized for the experiment purposes (centralized solution with limited scalability and fault tolerance). We will thus investigate alternative forwarding mechanisms (e.g. IPv6 segment routing) and distributed architecture (e.g., SFC-aware VNFs). Further experiments will be carried out with realistic VNFs and TCP as well as UDP flows. Finally, further study is needed to analyse also infrastructure provider requirements and, more specifically, how tenant-side orchestration mechanisms may influence infrastructure usage and resource orchestration policies enforced by NFVI providers.

Acknowledgment

The authors acknowledge all partners of the 5GINFIRE project for their support to the experiment. The authors also thank Mr. Luca Capannesi from

the University of Florence for his technical support.

References

[1] C. Cui, H. Deng, D. Telekom, U. Michel, H. Damker, Network functions virtualisation, in: SDN and OpenFlow World Congress, Darmstadt, Germany, 2012.

[2] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, S. Uhlig, Software-defined networking: A comprehensive survey, Proceedings of the IEEE 103 (1) (2015) 14–76.

[3] W. Stallings, Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud, Pearson Education, 2015.

[4] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, H. Flinck, Network slicing and softwarization: A survey on principles, enabling technologies, and solutions, IEEE Communications Surveys Tutorials 20 (3) (2018) 2429–2453.

[5] K. Makhijani, J. Qin, R. Ravindran, L. Geng, L. Qiang, S. Peng, X. de Foy, A. Rahman, A. Galis, Network Slicing Use Cases: Network Customization and Differentiated Services, draft-netslices-usecases-02 (Work in Progress), https://tools.ietf.org/html/draft-netslices-usecases-02 (2017).

[6] L. M. Contreras, D. R. López, A Network Service Provider Perspective on Network Slicing, [Online]. Available: https://sdn.ieee.org/newsletter/january-2018/a-network-service-provider-perspective-on-network-slicing. Accessed 21 March (2019).

[7] NGMN 5G security group, 5g security recommendations package 2: Network slicing, [Online]. Available: https://www.ngmn.org/publications/5g-security-recommendations-package-2-network-slicing.html (2016).

[8] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, J. Folgueira, Network slicing for 5g with sdn/nfv: Concepts, architectures, and challenges, IEEE Communications Magazine 55 (5) (2017) 80–87. doi:10.1109/MCOM.2017.1600935.

[9] H. Wang, A. Srivastava, L. Xu, S. Hong, G. Gu, Bring your own controller: Enabling tenant-defined sdn apps in iaas clouds, in: IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, 2017, pp. 1–9. doi:10.1109/INFOCOM.2017.8057137.

[10] ETSI, Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Report on architecture options to support multiple administrative domains, ETSI GR NFV-IFA 028 V3.1.1, `http://www.etsi.org/deliver/etsi_gr/NFV-IFA/001_099/028/03.01.01_60/gr_NFV-IFA028v030101p.pdf` (2018).

[11] J. Halpern, C. Pignataro, Service function chaining (sfc) architecture, IETF RFC 7665, `https://tools.ietf.org/html/rfc7665`.

[12] F. Callegati, W. Cerroni, C. Contoli, R. Cardone, M. Nocentini, A. Manzalini, Sdn for dynamic nfv deployment, IEEE Communications Magazine 54 (10) (2016) 89–95. doi:10.1109/MCOM.2016.7588275.

[13] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, T. Magedanz, Service function chaining in next generation networks: State of the art and research challenges, IEEE Communications Magazine 55 (2) (2017) 216–223. doi:10.1109/MCOM.2016.1600219RP.

[14] H. Hantouti, N. Benamar, T. Taleb, A. Laghrissi, Traffic steering for service function chaining, IEEE Communications Surveys & Tutorials 21 (1) (2018) 487–507.

[15] R. Munoz, R. Vilalta, R. Casellas, R. Martinez, T. Szyrkowiec, A. Autenrieth, V. Lopez, D. Lopez, Integrated sdn/nfv management and orchestration architecture for dynamic deployment of virtual sdn control instances

for virtual tenant networks [invited], IEEE/OSA Journal of Optical Communications and Networking 7 (11) (2015) B62–B70.

[16] N. F. S. de Sousa, D. A. L. Perez, R. V. Rosa, M. A. Santos, C. E. Rothenberg, Network service orchestration: A survey, Computer Communications 142-143 (2019) 69 – 94. doi:https://doi.org/10.1016/j.comcom.2019.04.008. URL http://www.sciencedirect.com/science/article/pii/S0140366418309502

[17] G. Sun, Y. Li, D. Liao, V. Chang, Service function chain orchestration across multiple domains: A full mesh aggregation approach, IEEE Transactions on Network and Service Management 15 (3) (2018) 1175–1191. doi:10.1109/TNSM.2018.2861717.

[18] D. Dietrich, A. Abujoda, A. Rizk, P. Papadimitriou, Multi-provider service chain embedding with nestor, IEEE Transactions on Network and Service Management 14 (1) (2017) 91–105. doi:10.1109/TNSM.2017.2654681.

[19] A. Abujoda, P. Papadimitriou, Distnse: Distributed network service embedding across multiple providers, in: 2016 8th International Conference on Communication Systems and Networks (COMSNETS), 2016, pp. 1–8. doi:10.1109/COMSNETS.2016.7439948.

[20] Q. Zhang, X. Wang, I. Kim, P. Palacharla, T. Ikeuchi, Vertex-centric computation of service function chains in multi-domain networks, in: 2016 IEEE NetSoft Conference and Workshops (NetSoft), 2016, pp. 211–218.

[21] H. Huang, S. Guo, J. Wu, J. Li, Joint middlebox selection and routing for software-defined networking, in: 2016 IEEE International Conference on Communications (ICC), 2016, pp. 1–6. doi:10.1109/ICC.2016.7510816.

[22] A. M. Medhat, G. Carella, C. Lck, M. Corici, T. Magedanz, Near optimal service function path instantiation in a multi-datacenter environment, in: 2015 11th International Conference on Network and Service Management (CNSM), 2015, pp. 336–341. doi:10.1109/CNSM.2015.7367379.

[23] Z. Cao, M. Kodialam, T. V. Lakshman, Traffic steering in software defined networks: Planning and online routing, SIGCOMM Comput. Commun. Rev. 44 (4) (2014) –. doi:10.1145/2740070.2627574.
URL http://doi.acm.org/10.1145/2740070.2627574

[24] Y.-W. Ma, J.-L. Chen, J.-Y. Jhou, Adaptive service function selection for network function virtualization networking, Future Generation Computer Systems 91 (2019) 108 – 123. doi:https://doi.org/10.1016/j.future.2018.06.051.
URL http://www.sciencedirect.com/science/article/pii/S0167739X17328406

[25] J. Pei, P. Hong, K. Xue, D. Li, Resource aware routing for service function chains in sdn and nfv-enabled network, IEEE Transactions on Services Computing (2018) 1–1doi:10.1109/TSC.2018.2849712.

[26] J. Wang, B. He, J. Wang, T. Li, Intelligent vnfs selection based on traffic identification in vehicular cloud networks, IEEE Transactions on Vehicular Technology 68 (5) (2019) 4140–4147. doi:10.1109/TVT.2018.2880754.

[27] 5GINFIRE H2020 Project Web Site, [Online]. Available: https://5ginfire.eu/. Accessed February 27 (2020).

[28] ETSI, Network function virtualization Ecosystem - Report on SDN Usage in NFV Architectural Framework ETSI GS NFV-EVE 005, https://www.etsi.org/deliver/etsi_gs/NFV-VE/001_099/005/01.01.01_60/gs_NFV-EVE005v010101p.pdf (2015).

[29] B. Martini, F. Paganelli, P. Cappanera, S. Turchi, P. Castoldi, Latency-aware composition of virtual functions in 5g, in: Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft), 2015, pp. 1–6. doi:10.1109/NETSOFT.2015.7116188.

[30] G. Cuffaro, F. Paganelli, P. Cappanera, Tenant-side management of service function chaining: architecture, implementation and experiment on a future

internet testbed, in: Proceedings of the 2019 IEEE Conference on Network Softwarization: Unleashing the Power of Network Softwarization, NetSoft 2019, Vol. 8806702, 2019, pp. 124–132.

[31] J. Halpern, C. Pignataro, Service function chaining (sfc) architecture, RFC 7665, RFC Editor (October 2015).

[32] M. Boucadair, Service function chaining (sfc) control plane components and requirements, Internet-Draft draft-ietf-sfc-control-plane-08, IETF Secretariat, http://www.ietf.org/internet-drafts/draft-ietf-sfc-control-plane-08.txt (October 2016).
URL http://www.ietf.org/internet-drafts/draft-ietf-sfc-control-plane-08.txt

[33] ETSI, Network Function Virtualization Management and Orchestration, GS NFV-MAN 001 V1.1.1, http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf (2014).

[34] M. Olbrich, F. Nadolni, F. Idzikowski, H. Woesner, Measurements of path characteristics in planetlab, Tech. Rep. Tech. Rep. TKN Technical Report TKN-09-005, Technical University Berlin, http://www.tkn.tu-berlin.de/fileadmin/fg112/Papers/TKN_Technical_Report_TKN-09-005.pdf (2009).
URL http://www.tkn.tu-berlin.de/fileadmin/fg112/Papers/TKN_Technical_Report_TKN-09-005.pdf

[35] N. Sambo, L. Valcarenghi, P. Iovanna, G. Imbarlina, F. Ubaldi, T. Pepe, A. Garcia-Saavedra, G. Landi, I. Pascual, R. Martinez, J. Mangues-Bafalluy, C. Vitale, C. Chiasserini, A. Ksentini, F. Klamm, C. Turyagyenda, Mobile transport and computing platform for 5g verticals: Resource abstraction and implementation, in: 2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2018, pp. 1–6. doi:10.1109/NFV-SDN.2018.8725613.

[36] H. Yan, Y. Li, W. Dong, D. Jin, Software-defined wan via open apis, IEEE Access 6 (2018) 33752–33765. doi:10.1109/ACCESS.2018.2833211.

[37] A. Betker, C. Gerlach, R. Hülsermann, M. Jäger, M. Barry, S. Bodamer, J. Späth, C. Gauger, M. Köhn, Reference transport network scenarios, MultiTeraNet Report (2003).

[38] J.-C. Bolot, End-to-end packet delay and loss behavior in the internet, in: ACM SIGCOMM Computer Communication Review, Vol. 23, ACM, 1993, pp. 289–298.

[39] A. Gavras, S. Denazis, C. Tranoris, H. Hrasnica, M. B. Weiss, Requirements and design of 5g experimental environments for vertical industry innovations, in: Wireless Summit (GWS), 2017 Global, IEEE, 2017, pp. 165–169.

[40] B. Nogales, I. Vidal, D. R. Lopez, J. Rodriguez, J. Garcia-Reinoso, A. Azcorra, Design and deployment of an open management and orchestration platform for multi-site nfv experimentation, IEEE Communications Magazine 57 (1) (2019) 20–27. doi:10.1109/MCOM.2018.1800084.

[41] G. Davoli, W. Cerroni, S. Tomovic, C. Buratti, C. Contoli, F. Callegati, Intent-based service management for heterogeneous software-defined infrastructure domains, International Journal of Network Management 29 (1) (2019) e2051, e2051 nem.2051. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nem.2051, doi:10.1002/nem.2051.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.2051

[42] M. Gharbaoui, C. Contoli, G. Davoli, G. Cuffaro, B. Martini, F. Paganelli, W. Cerroni, P. Cappanera, P. Castoldi, Experimenting latency-aware and reliable service chaining in next generation internet testbed facility, in: 2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2018, pp. 1–4. doi:10.1109/NFV-SDN.2018.8725783.

[43] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, P. Francois, The segment routing architecture, in: 2015 IEEE Global Communications Conference (GLOBECOM), 2015, pp. 1–6. doi:10.1109/GLOCOM.2015.7417124.

[44] Filsfils et al., Segment Routing Architecture, RFC 8402, `https://tools.ietf.org/html/rfc8402` (July 2018).

[45] A. Mayer, S. Salsano, P. L. Ventre, A. Abdelsalam, L. Chiaraviglio, C. Filsfils, An efficient linux kernel implementation of service function chaining for legacy vnfs based on ipv6 segment routing, in: 2019 IEEE Conference on Network Softwarization (NetSoft), 2019, pp. 333–341.

[46] Linux Advanced Routing & Traffic Control HOWTO, [Online]. Available: `http://lartc.org/howto/`. Accessed 21 March (2019).

[47] nfqueue Documentation, [Online]. Available: `https://docs.rs/nfqueue`. Accessed 21 March (2019).

[48] Scapy Project, [Online]. Available: `https://scapy.net/`. Accessed 21 March (2019).

[49] J. G. Herrera, J. F. Botero, Resource Allocation in NFV: A Comprehensive Survey, IEEE Transactions on Network and Service Management 13 (3) (2016) 518–532. doi:10.1109/TNSM.2016.2598420.

[50] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, R. Boutaba, Network function virtualization: State-of-the-art and research challenges, IEEE Communications Surveys Tutorials 18 (1) (2016) 236–262. doi:10.1109/COMST.2015.2477041.

[51] A. Laghrissi, T. Taleb, A survey on the placement of virtual resources and virtual network functions, IEEE Communications Surveys Tutorials 21 (2) (2019) 1409–1434. doi:10.1109/COMST.2018.2884835.

[52] P. Cappanera, F. Paganelli, F. Paradiso, Vnf placement for service chaining in a distributed cloud environment with multiple stakeholders, Computer Communications 133 (2019) 24 – 40.

doi:https://doi.org/10.1016/j.comcom.2018.10.008.
URL http://www.sciencedirect.com/science/article/pii/S0140366418303104

[53] V. Eramo, F. G. Lavacca, T. Catena, M. Polverini, A. Cianfrani, Effectiveness of segment routing technology in reducing the bandwidth and cloud resources provisioning times in network function virtualization architectures, Future Internet 11 (3) (2019) 71.

[54] W. Y. Poe, I. Vaishnavi, F. Tusa, J. Melin, A. Ramos, System architecture of intelligent monitoring in multi-domain orchestration, in: 2017 European Conference on Networks and Communications (EuCNC), 2017, pp. 1–5. doi:10.1109/EuCNC.2017.7980673.

[55] S. Arezoumand, K. Dzeparoska, H. Bannazadeh, A. Leon-Garcia, Md-idn: Multi-domain intent-driven networking in software-defined infrastructures, in: 2017 13th International Conference on Network and Service Management (CNSM), 2017, pp. 1–7. doi:10.23919/CNSM.2017.8256016.