Tenant-side management of Service Function Chaining: architecture, implementation and experiment on a Future Internet testbed

Giovanni Cuffaro
CNIT
Firenze, Italy
giovanni.cuffaro@cnit.it

Federica Paganelli
Department of Computer Science
University of Pisa
Pisa, Italy
federica.paganelli@unipi.it

Paola Cappanera
Department of Information Engineering
University of Florence
Firenze, Italy
paola.cappanera@unifi.it

***Abstract*—The adoption of Network Function Virtualization and Software Defined Networking technologies allow network infrastructure operator flexibly orchestrating resources to provide tenants with their own virtual network. However, access to computing and network resource management APIs is typically allowed only within the infrastructure domain and rarely disclosed to tenants for security and performance reasons. This may severely limit tenants capability in coping with demands of application-tailored network services, including Service Function Chaining (SFC). While the literature extensively addressed the challenges of SFC in the infrastructure domain, tenant-side SFC management is quite unexplored yet, although discussed also by a standardization group. This work proposes an SFC platform (called SFCLola) providing tenants with a latency-aware SFC management while minimizing support required from infrastructure operators. The platform encompasses two main levels: an end-to-end chain management level featuring a VNF selection algorithm and a forwarding mechanism that can be programmed and enforced within the tenant network of VMs without requiring access to the switches at the network infrastructure data plane. SFCLola has been implemented as software prototype and experimentally evaluated on a multi-DC infrastructure provided by the 5GINFIRE project.**

***Index Terms*—Service Function Chaining, optimization, Virtual Network Function, tenant, Software Defined Networking, intent interface**

I. Introduction

Orchestration of virtual resources and network programmability brought by Network Function Virtualization (NFV) [1] and Software Defined Networking (SDN) [2] allow to create multiple virtual tenant networks over a shared physical infrastructure (Network Function Virtualization Infrastructure - NFVI), whose resources are managed and owned by multiple operators. Each virtual network is logically isolated and can be dedicated to serve different sets of services with diverse application-specific requirements [3]. Such an approach may also assume the flavour of "network slicing", where a network slice consists of a set of infrastructure resources and service functions to meet the needs of an industry vertical or a service.

Such a flexible and application-oriented use of resources implies a multi-domain architecture, distinguishing tenant and infrastructure network domains, where a tenant domain *"provides VNFs, and combinations of VNFs into Network Services, and is responsible for their management and orchestration, including their functional configuration and maintenance at application level"*, whilst the infrastructure network is the *"domain within the NFVI that includes all networking that interconnects compute/storage infrastructure"* [4]. In this work with the term "tenant" we refer to the definition provided in [5]: "*whatever entity (person, organizational unit, organization, etc.) uses the NFVI to provide or manage network services*".

The extent to which the infrastructure provider should expose capabilities for network slice management and setup of dedicated services, such as VNF instances and service chain management is under discussion (e.g., [6]). In this context, access to computing and network resource management APIs is typically allowed only within the domain of infrastructure operators and rarely disclosed to tenants for security and performance reasons [7]. This may severely limit tenants' capability in managing application-tailored network services, including Service Function Chaining (SFC).

Indeed, the NFV ETSI Industry Specification Group (ISG) [5] highlights two main architectural options for service chaining in a virtualized infrastructure:

- *SFC defined in the NFVI domain*. The service chain control features (i.e., SDN Controller or VIM networking functions) are deployed in the NFVI.
- *SFC defined in the tenant domain*. This implies that service chains are defined on top of a virtual network of nodes in the tenant domain and traffic control features and data plane forwarding capabilities are controlled by the tenant. This may be implemented leveraging SDN control capabilities and switching/forwarding features implemented as VNFs in the tenant domain but, depending on infrastructure operators' service offering and security

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.

978-1-5386-9376-6/19/$31.00 ©2019 IEEE

policies, may not be supported by SFC-oriented capabilities provided by the NFVI (e.g. OpenStack SFC API).

While a huge amount of research work has been devoted to SFC in the NFVI domain [8], [9], SFC in the tenant domain is quite unexplored. Wang et al. [7] and Muñoz et al. [10] propose virtualized SDN services to tenant, which, however, are used to program switches in the NFVI.

In this work we propose an SFC platform (called SFCLola) providing tenants with latency-aware SFC management while minimizing the requirements on the infrastructure and, consequently, on the interaction with NFVI management features. More specifically, the contribution of the work is threefold:

- an SFC application (SFCLola) managing chain creation and deletion requests. The chain setup leverages an optimization algorithm that selects VNF instances available from different DCs (data centers) to minimize an end-to-end latency estimated considering both processing and network latency and sends appropriate chaining instructions to forwarding devices SFCLola can interact with SFC API offered by Virtual Infrastructure Managers (VIM), such as OpenStack SFC API or custom intent-based API (as documented in our previous work [11]). In this work we investigate the case where SFC has to be implemented within the tenant domain (see next point).
- a forwarding mechanism to manage chains in the tenant domain. Chaining instructions are implemented as level 3 routing within the virtual network of the VMs managed by the tenant, exploiting Linux policy routing capabilities. In practice, an SFC-aware proxy, called Virtual Flow Forwarder (VFF), is used to steer traffic across VNFs along a chain. The VFF is a Java application that exposes intent-based REST API to receive chaining instructions and translates them into flow forwarding rules enforced through Linux policy routing capabilities.
- Finally, this solution has been implemented as a software prototype and experimentally validated on the multi-DC infrastructure provided by the 5GINFIRE project [12].

It is worth noticing that, although not relying on OpenFlow, our approach is inspired by the SDN paradigm in that: i) control (i.e., SFCLola application) is kept separate from forwarding software (i.e., VFF), ii) programmability is realized through REST APIs at both control and forwarding levels, and iii) an abstract network topology is maintained for taking control decisions.

The proposed solution relies on IPv4 forwarding capabilities to minimize requirements on NFVI. However, the platform is modular and can be extended with alternative forwarding solutions, such as IPv6 Segment Routing [13].

The paper is structured as follows. Section II discusses related work. Section III describes SFCLola design and implementation details, including the adopted VNF selection algorithm and the forwarding mechanism. In Section IV we describe the methodology of experimental evaluation and the topology of the experiment carried out on top of the 5GINFIRE testbed facilities. Section V reports the results of the experiment, including a comparative evaluation with alternative VNF selection strategies. Section VI concludes the paper with a discussion on the proposed approach and future research directions.

II. Related work

Recently, several works have addressed the problem of flexibly management of specific service/application traffic, by proposing SFC solutions for flow classification, policy management and traffic steering [8], [11], [14]. Such solutions typically rely of network programming capabilities in the network infrastructure (i.e. network hypervisors, infrastructure SDN controllers, etc.). However, management and access to APIs of infrastructure SDN controllers is typically managed by infrastructure operators for security and performance reasons, thus offering poor support to tenants' demands of network functions and application-tailored network services [7].

As discussed in [5], several options exist for integrating SDN control functions in an NFV architecture. Moreover, ETSI propose a hierarchical network control architecture distinguishing infrastructure and tenant controllers.

Although enabling tenant-side SFC is key for the flexible management of application-specific network services, this topic has been rarely investigated yet. Wang et al. [7] introduce a "Bring Your Own Controller" (BYOC) paradigm and propose BYOC-VISOR, an SDN virtualization platform that provides customized and scalable SDN services to cloud users. Muñoz et al. [10] propose an integrated SDN/NFV management and orchestration architecture for multi-tenant transport networks where tenant SDN controllers are deployed as VNFs in DCs. However, neither [7] nor [10] address tenant-side SFC management and both approaches relies on the integration with NFVI management features.

Recently, some authors have begun developing and experimenting SFC based on the IPv6 Segment Routing (SR) programming model [15]. SR is a routing architecture that is being developed within the Internet Engineering Task Force (IETF) [13]. Instead of using traditional destination-based hop-by-hop forwarding, SR forwards packets towards their destination by specifying a list of way points called segments. This mechanism can thus be used to steer packets through VNFs. SR implementations exist in the MPLS [16] and IPv6 data planes [17], [18]. SR is a promising technology for SFC support, but we did not adopt it in this work to carry out short-term experiment activities using mature forwarding technologies and minimizing technological requirements on the infrastructure.

III. SFCLola architecture

This section describes the main concepts of the SFCLola architecture. First we introduce the reference service function chaining use case, we sketch the adopted solution strategy and, then, we describe the main components of SFCLola.

The goal of SFCLola is to handle SFC requests within a tenant's virtual network spanning multiple DCs. Our reference use case consists of multiple instances of different types of VNFs

deployed in virtual networks managed by the tenant and hosted at multiple and geographically distributed sites. SFC requests are specified by clients at a high level of abstraction, i.e., a request should include the information for flow classification (e.g., source and/or destination IP and address, protocol type, etc...) and the specification of the chain in terms of ordered sequence of VNF types and related requirements (maximum end-to-end latency, minimum bandwidth).

The problem we address is the setup of the requested service chain within a tenant virtual network, possibly spanning multiple DCs, by, first, selecting VNF instances that minimize end-to-end latency, instructing the data plane accordingly, and, finally, enforcing chaining instructions. As mentioned above, SFC management within the tenant domain implies that traffic routing along chained VNFs should be enforced by programmable forwarding elements that manage traffic flow routing in the tenant data plane, without relying on management capabilities at the NFVI level.

For the sake of clarity, we consider the example of a request chain from source node *SRC* to destination node *DST*, and an ordered sequence of VNF types (VF-1, VF-2, VF-3) (shown in the top side of Fig. 1. The proposed strategy consists of two main steps: VNF Selection and Service Function Chaining.

VNF Selection. The end-to-end SFC request is handled by the SFCLola application that, using an abstracted view of the multi-DC topology, selects the VNF instances that should realize the target service chain (spanning multiple DCs, where appropriate), using an optimization algorithm. As shown in Fig. 1, a possible choice is selecting VF-1 instance in DC1 and VF-2 and VF-3 instances in DC2. SFCLola then prepares and sends appropriate forwarding instructions to each concerned DC. The forwarding instructions dispatched by SFCLola defines a logical path (blue dotted line in Fig. 1). As explained in the following item, in this work we use a Service Function Chaining mechanism based on programming forwarding nodes leveraging linux policy routing. However, the application could use traditional APIs, such as OpenStack SFC APIs, if exposed by the provider. A previous release of the VNF Selection application uses intent-based SFC APIs offered by NFVI providers [11].

Service function chaining. In order to minimize inter-DC connection requirements, the tenant virtual network is an interconnection of L3 networks. As regards SFC, a DC may be responsible of a portion of a chain. In our architecture the forwarding action is implemented as a software component executed by a VM, called Virtual Flow Forwarder (VFF). A VFF acts as a forwarding device in the overlay network, i.e., all the packets exiting from/directed to a VNF node has to be forwarded to the VFF, which decides the next hop, according to the installed forwarding rules. The VFF offers an intent-based REST API for creating or deleting service chains or parts of service chain within a DC. The logical path forwarding instructions sent by SFCLola are mapped by the VFF into a L3 routing path in the virtual network (red line in Fig. 1).

Hereafter we describe the SFCLola application, the VNF Selection algorithm and the VFF.

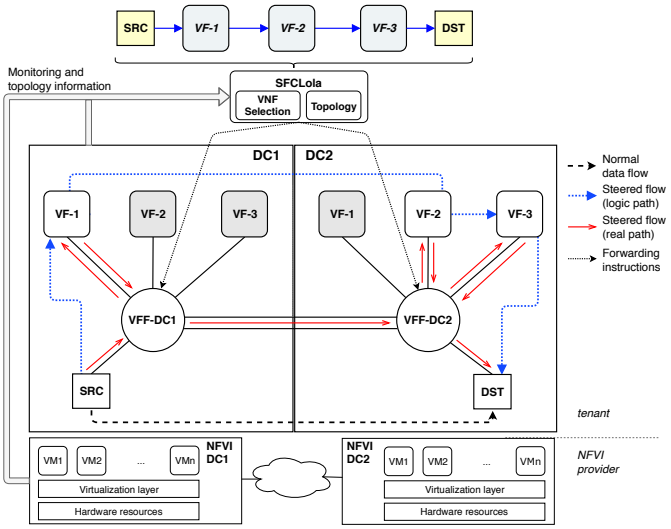

Fig. 1. SFCLola forwarding mechanism in a tenant network in a 2-DC environment. The blue dotted line represents the logical path to implement a service chain as decided by SFCLola (VF-1 in DC1 and VF-2 and VF-3 in DC2), while the red path represents the real path enforced by a VFF element within each intra-DC virtual network.

A. SFCLola application

SFCLola is a JAVA application exposing REST APIs for managing service chains through Create Read Update and Delete (CRUD) operations. In order to accomplish service chain requests, it leverages an optimization algorithm that selects VNF instances available from different clouds that minimize the end-to-end latency estimated considering both processing delays and inter-DC network delay.

The algorithm (described in Section III-B) takes its decision based on an abstracted view of the underlying topology, thus also taking into account the case that the NFVI provider decides to expose only a subset of topological and resource information (as discussed by ETSI in [19]). This abstracted view includes measured inter-DC network latency values for each pair of DCs and, for each DC, the list of VNF instances available and their parameters (i.e., VNF type, processing capacity, measured traffic input rate). We suppose this set of information is periodically provided by the infrastructure and/or measured within the tenant domain (see Section IV for the description of a prototype experiment). These measurements are influenced and reflect possible events at the infrastructure level (i.e. VM migration, availability of VNF instances), therefore the frequency to which monitored information are collected and polled by SFCLola is a key design choice.

If it is possible to satisfy the request, the algorithm will provide an optimal solution as output. Starting from the optimal solution, SFCLola estimates the end-to-end latency and, if this value exceeds the maximum latency tolerated by the request, the request is rejected and an error message is returned to the client. If the solution encompasses more than one DC, SFCLola splits the forwarding instructions and sends to each DC a chain specification composed by: flow classification information, the sub-chain to be realized within

the DC (e.g. VF-2 and VF-3 in DC2) and appropriate ingress and egress connection points towards external DCs if the flow does not originate/terminate in that DC (e.g. DC2 will receive the target flow from DC1 connection point).

B. VNF Selection algorithm

The adopted optimization algorithm, already presented in [20], solves a Resource Constrained Shortest Path problem on an auxiliary layered graph properly defined, which is known to have pseudo-polynomial complexity [21]. Specifically, a layered graph is built for each request as exemplified in Fig.2. The graph contains a layer for each VNF in the request and two extra layers, i.e., layer zero containing the source node s^r of the request and the last layer containing the destination node d^r of the request. Layer i, i.e., an intermediate layer different from the first and the last one, contains all those DCs able to operate the i-th VNF of the chain. Thus, the layered structure of the graph ensures that the order of VNFs specified in the request is preserved. Arcs in the auxiliary graph only link nodes belonging to two consecutive layers. Specifically, there are arcs between the source node at layer zero and the nodes of layer one corresponding to DCs hosting instances of the first virtual function in the request. Then, there are arcs connecting nodes in the layer corresponding to the last virtual function of the request with the destination node, and arcs between a node in layer i to a node in layer $i+1$ if the assignment of VNFs i and $i+1$ in the request to the DCs corresponding to such nodes is feasible. Additional constraints (e.g., incompatibility constraints between DCs and VNFs or maximum allowed network latency on the link between two specific DCs) can be taken into account and properly enforced during the graph construction phase. Once the auxiliary graph has been defined for a given request, the problem of satisfying the request at minimum cost can be solved by finding a constrained shortest path on such a graph from the source node to the destination node. By construction, such a path visits only one node in each layer and the node visited in each intermediate layer i corresponds to the DC hosting the i-th VNF.

The optimization problem is thus formulated mathematically by means of 0-1 decision variables corresponding to the arcs of the auxiliary layered graph. The constraints guarantee that the design variables define a path starting from the source node, ending in the destination node and visiting exactly one node in each intermediate layer. Additional constraints guarantee quality of service requisites on the whole path, such as for example those imposing that the total latency experimented along the whole path does not exceed the maximum latency tolerated by the request. For each request, the objective function in the optimization problem accounts for the minimization of the estimated end-to-end latency which is given by both processing and network latency. The cost of the path connecting the source and the destination nodes of a request is given by the sum of the latency at the intermediate nodes and on the used network links.

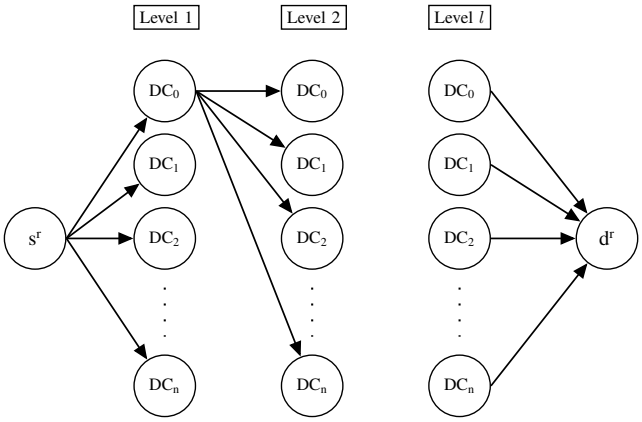

Fig. 2. Layered auxiliary graph for request r

C. Virtual Flow Forwarder

The VFF is implemented as a software running on a VM, which exploits Linux kernel routing capabilities and offers a set of REST APIs for installing or deleting forwarding rules. The REST API offered by the VFF has been designed as an intent-based interface, so that the client only provides functional specifications of the actions to be accomplished. SFC instructions are delivered as JSON messages encoding abstract forwarding instructions specifying the ids of VNF instances to be chained within that DC, antecedent hop (i.e. source or previous DC) and subsequent one (i.e., next DC or final destination). The VFF translates these instructions into forwarding rules. In the current implementation flows can be classified through a combination of the following parameters: source/destination IP addresses and ports and transport protocol. For instance, a request for creating a new chain is specified 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.

Since the chaining sequence across VNFs is enforced by the VFF, which is the only element in an intra-DC overlay network aware of service chaining, endpoint nodes and VNFs must be configured to send their traffic to the VFF. Then, the VFF will forward the packet flow to the next hop in the chain, no matter the IP destination of the end-to-end service.

Traffic steering through VNFs is achieved through the joint use of Linux policy routing [22] and the packet marking capability provided by the netfilter framework, which extends the traditional destination-based routing strategy. More specifically, in each VFF node we set netfilter rules so that packets are marked based on a combination of filters (e..g., flow classification information, incoming interface, source MAC address). Marks are used by the Linux kernel to select the appropriate routing tables, according to previously configured policy routing rules (i.e., routing tables are associated to marks through the *fwmark* parameter). As a prerequisite we create routing tables for each VNFs in the VFF DC, so that packets are forwarded to that VNF.

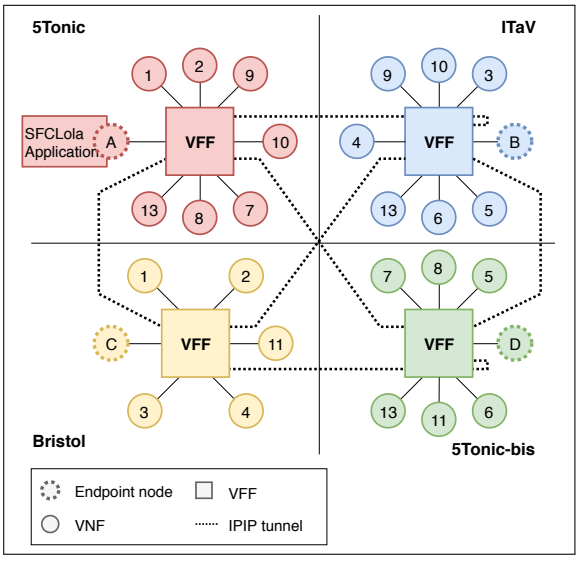

Fig. 3. Experiment topology for SFCLola evaluation on the 5GINFIRE infrastructure: a tenant network consisting of 33 VMs (divided into endpoint nodes, VNFs and VFFs) on a multi-site and multi-operator environment has been reproduced using the facilities of four different 5GINFIRE testbeds (5TONIC, 5TONIC-bis, ITaV, Bristol)

SFCLola sends JSON messages to VFF through REST APIs to ask creating or deleting (portions of) service function chains. The VFF translates this request into the insertion/deletion of the appropriate set of flow marking rules in PREROUTING (see example in 1) forcing the use of routing tables so to steer the flow through the desired VNFs. This implies that given a chain, or a portion of a chain, of length n to be realized inside a DC, if the final destination node is within the DC network $n-1$ flow marking rules are added, n rules are added otherwise (i.e., the final next hop is a VFF in another DC).

When the chain implementation consists of a sequence of VNFs instantiated in different DCs, the delivery of packets to another DC is managed at VFF level by sending packets with destination to the VFF of the external DC through IPinIP encapsulation. This allows easily building a logical tunnel between the two VMs hosting the VFFs. Moreover, netfilter allows easily marking flows coming from an IPinIP interface.

```
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

2  -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 RETURN
```

Listing 1. Example of iptables 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

Listing 1 shows a minimal set of iptables rules that are produced by a VFF upon a chain creation request. These rules are installed so to classify the traffic and mark the corresponding packets. Each mark is associated with a specific routing table allowing the routing module to forward the traffic to the correct next hop (i.e. next VNF) in the chain.

TABLE I
VNF DEPLOYMENT IN 5GINFIRE MULTI-DC TOPOLOGY

VF \ DC	5Tonic	ITaV	Bristol	5Tonic-bis
VF-1	✓		✓	
VF-2	✓		✓	
VF-3		✓	✓	
VF-4		✓	✓	
VF-5		✓		✓
VF-6		✓		✓
VF-7	✓			✓
VF-8	✓			✓
VF-9	✓	✓		
VF-10	✓	✓		
VF-11			✓	✓
VF-13	✓	✓		✓
VNF capacity	100 Mbps	150 Mbps	80 Mbps	120 Mbps

IV. Experiment Methodology

In this section we describe the methodology and the setup of the experiment on the 5GINFIRE infrastructure facilities to experimentally evaluate the SFCLola solution. 5GINFIRE [12] is a 5G NFV-enabled experimental testbed consisting of multi-site cloud and vertical-specific facilities endowed with open source Management and Orchestration (MANO) features and a set of tools for the automated instantiation, deployment and lifecycle management of experiments and resources.

A. Experiment Topology

We deployed 33 VMs on 4 logical networks provided by the following testbeds: 5TONIC (providing 2 networks, called 5TONIC and 5TONIC-bis, very close to each other), ITaV in Portugal and the University of Bristol's 5G testbed. The logical topology is shown in Fig. 3. On each DC we instantiated a VFF, an endpoint node capable to be used both as traffic source or destination and a set of VNFs, as specified in table I. We used a VNF emulation software that delays the incoming traffic proportionally to the traffic rate and a custom parameter (named Processing Capacity) representing the computational power of the VNF. At the bottom of table I the processing capacities (in Mbps) of the deployed VNFs are summarized (every VNF belonging to the same DC has the same capacity but VNFs of the same type in different DCs have different capacity). Monitoring information required by the algorithm is measured through ad-hoc developed software components, deployed in the tenant domain, that periodically post measurements on the Gnocchi database. SFCLola uses the REST APIs provided by Gnocchi to retrieve metrics. Further details on measured QoS (latency) of inter-DC links are provided in the Appendix.

B. Metrics

The reference workflow of the experiment activities consists in creating a set of chain requests that are sent sequentially to SFCLola, waiting an interval long enough to let the monitoring

system sense the changes and, then, collect the measurements on the infrastructure to evaluate the following metrics:

- *Computation time*: time spent by the optimization algorithm to handle a VNF selection request.
- *VFF load*: we analyzed the impact of chain setup and flow forwarding at VFF tracking CPU load and incoming traffic rate variations.
- *E2E latency*: time encompassed by a packet going from source node to the its final destination (through the VNF chain in case of an active chain).
- *Latency estimation error*: error between the estimated latency, used by the VNF selection algorithm, and the end-to-end latency measured after the chain setup and during traffic flow.

V. Results

This section describes the main results of the experiment activities. First, we report on the results of the tests measuring the impact on VFFs resources by varying the load in terms of flow routing rules and incoming traffic rate. Then, we describe a set of tests for evaluating the error in the estimation of the end-to-end latency used by the VNF selection algorithm. Finally, we compare the adopted VNF selection algorithm with two baseline strategies: a round robin approach and a greedy one.

A. VFF Load

We performed a set of tests to monitor the CPU load, while progressively adding chains and then sending the corresponding traffic flows. As described above, a new chain request requires a set of rules to be installed at concerned VFFs. For each chain, the number of rules to be installed is proportional to the number of hops (i.e. selected VNFs instances on the DC for implementing the chain). Then, when the traffic flows have started, the VFF CPU has to manage incoming packets by performing rule matching against packet header and then by forwarding packets. The experiment topology and chain requests have been configured so to run the experiment in a 4-DC setting, where only one VFF in one DC is the monitored one (we selected the one in ITaV testbed) and in charge of the whole steering task, while the other DCs host the traffic sources (so that also the tunneling mechanism is always solicited). In table II we show the results of a test conducted with a set of 30 chain requests, each one composed by 4 VNFs and with a fixed bandwidth of 1 Mbps per traffic flow. The first row shows the CPU load (min, max and average values) measured before the experiment (system in idle status). The experiment distinguishes three main phases: rule installation, traffic steering and rule deletion.

In this experiment, 240 rules are installed, 2 for each hop in the chain (one rule for traffic steering, one for exiting the rule evaluation procedure). The maximum CPU load is around 7% and is reached when flow rules are installed (phase 1). The second row reports the CPU load values referring to the time interval when all traffic flows are active and no rule management operation is performed (phase 2). The CPU load is almost stable around 3,4%. We also traced the CPU load measured around the traffic peak occurrence (165 Mbps), which is around 2,9%. Fig. 4 provides more details on the bitrate and CPU load evolution during phase 1 and phase 2 of the experiment. To conclude the test, the 30 flows have been deleted (at a rate of 4 deletions per second) reaching a peak CPU load on the monitored VFF of about 18,3%.

TABLE II
CPU load and bitrate at VFF with 30 chains (4 VFs, 1 Mbps)

	CPU load			Incoming bitrate (Mbps)		
Timeinterval	min	max	avg	min	max	avg
Idle	0.10%	0.65%	0.28%	0.014	0.017	0.015
Phase 1	1.00%	7.20%	3.04%	0.01	145.0	79.1
Phase 2	2.87%	4.54%	3.40%	134.6	165.5	151.0
Phase 3	-	18.32%	-	-	-	-

Phase 1: installations of the flows (1 request each 5 seconds), phase 2: mere traffic steering, i. e. all traffic flows running together (for 60 seconds), phase 3: deletion of the flows

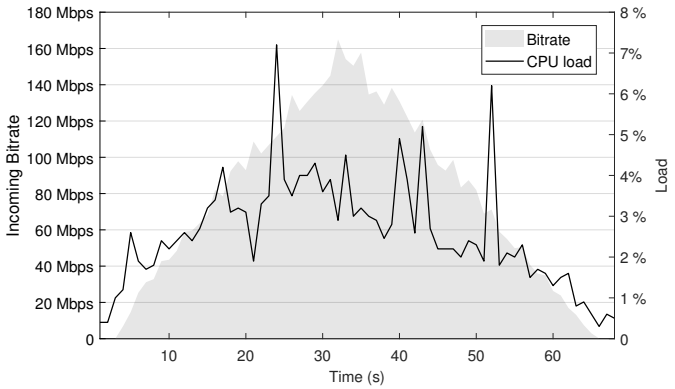

Fig. 4. CPU load at VFF due to chain installation and traffic steering with 30 chains (4 VNFs, 1 Mbps)

B. End-to-end Latency estimation

SFCLola takes its decisions using an estimated latency value for the new incoming chain, computed by considering the contributions of inter-DC network latencies and an estimation of processing delay introduced by each VNF. The latter is computed considering the processing delay measured at each VNF and the additional delay introduced by the incoming request. Inter-DC network latency values are obtained measuring the latency between each pair of VFFs at 10 seconds intervals and averaging these values over a 30 seconds wide window.

It is therefore relevant evaluating the accuracy of latency estimation with respect to the actual measurements of end-to-end delay of traffic flows steered through the chains. To this purpose, we performed several iterations of the following experiment:

1) We generate a set of 20 chain requests with a number of VNFs between 2 and 4.
2) The requests are submitted to SFCLola at the rate of one each 30 seconds.

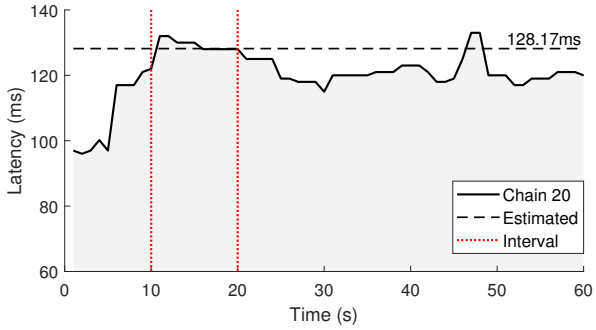

Fig. 5. Measured end-to-end latency of traffic flow of chain no. 20. The vertical dashed lines delimit the time interval selected for comparing estimated and measured latency values.

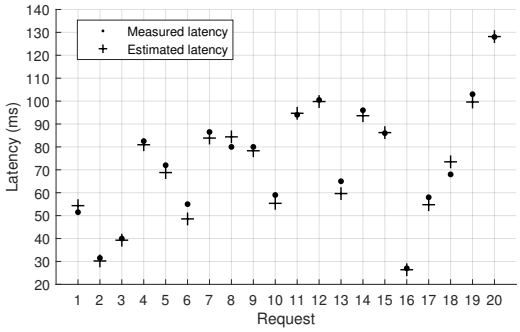

Fig. 6. Comparison between measured and estimated latency in a set of 20 requests.

3) For each request, the corresponding traffic flow is generated using the iPerf3 tool sending UDP traffic for a duration of 600 seconds and end-to-end latency is periodically measured.

It is worth observing that more than one flow can pass through a VNF at the same time. In this experiment we use destination ports to distinguish flows and we use an ad-hoc developed application (called UDP_Ping) to measure latency along a chain sending UDP probe packets. The actual end to end latency is calculated as follows: latency from source to destination node is measured by the UDP Ping application each second, and a median value is computed over the measurements in the central 10 seconds-interval. Fig. 5 shows the measured end-to-end latency of the traffic flow for request n. 20. This figure also helps clarifying the adopted methodology, in that it highlights the time interval (from second 10 to second 20) selected to gather end-to-end latency measurements for comparison with the estimated ones.

Fig. 6 shows a comparison between measured and estimated latency for each chain contained in a set of 20, sent one after the other every 30 seconds. The estimation error averaged over the chains in the request set is around 2.5 ms (i.e. 4%). This error strongly depends on the inter-dc network delay variations that can be experienced in the infrastructure and, consequently, on the way latency values are measured and aggregated, as explained above. We repeated this test five times, with similar results. The resulting average estimation error is 3.8%.

C. Comparative evaluation of VNF selection algorithms

We ran a set of tests to compare the VNF selection algorithm used in SFCLola with two alternative selection strategies: a greedy approach and a round robin one. The greedy approach works by choosing at each iteration the VNF instance with the lowest processing time, among the VNF available in all the DCs, thus disregarding the contribution of inter-DC network latency to the overall end-to-end delay. When using the round robin strategy VNF instances are selected in turn, thus distributing the load among DCs. This experiment is analogous to the one described in V-B with the only difference that each set of requests is sent to SFCLola 3 times, one for each selection strategy. Since the experiment is performed on a testbed, it is obvious that the status of the resource infrastructure may slightly vary at each iteration.

TABLE III
VNF SELECTION ALGORITHMS - COMPARATIVE EVALUATION RESULTS

	Percentage difference SFCLola vs Greedy		Percentage difference SFCLola vs Round Robin	
Set	avg	σ	avg	σ
1	-12.75%	25.12%	-28.47%	21.05%
2	-17.08%	21.60%	-28.51%	21.86%
3	-10.07%	23.10%	-29.11%	24.80%
4	-15.89%	23.07%	-22.85%	24.07%
5	-9.21%	16.59%	-25.91%	19.62%
6	-12.63%	21.00%	-25.41%	22.91%
7	-14.36%	24.12%	-24.74%	24.88%
8	-15.47%	22.40%	-25.99%	17.75%
9	-13.95%	18.41%	-25.88%	27.74%
10	-12.16%	16.90%	-28.11%	18.62%
Overall	-13.73%	21.19%	-26.91%	22.05%

Table III shows the difference (in percentage) between the latency obtained using the SFCLola optimized selection vs the greedy approach and a round robin selection strategy. Data are averaged by request set and for each set we observe a negative value which means SFCLola approach, on average, obtains a lower latency compared with the greedy and the round robin strategies. The overall latency improvement versus the greedy approach is more than 13% and more than 26% if compared to the round robin selection strategy.

We measured a peak gain on latency around 60% going from 123 ms using round robin and 122 ms with the greedy strategy to 48 ms using SFCLola, observing a decrease of about 74 ms. The measurement data collection and aggregation followed the same strategy described in section V-B. Fig. 7 shows the histogram of all the relative differences measured in our experiment between latency measurements of solutions provided by SFCLola and those by alternative approaches. We can state that in more than 75% of the requests we obtained an improvement over latency (i.e., negative values) and even when we measured a worse latency (i.e., positive values), the relative difference in measured latency is mostly displaced in the range 0%-20%, thus confirming the overall improvement provided by SFCLola.

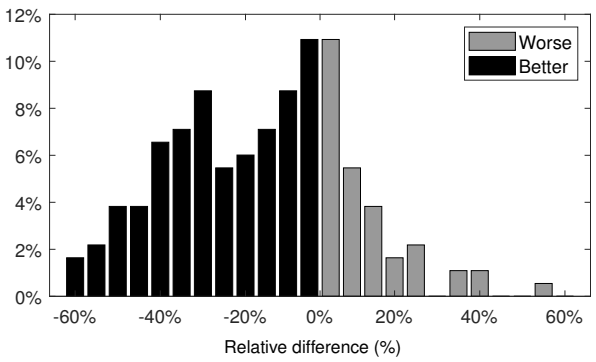

Fig. 7. Histogram of the percentage deltas computed considering all the requests sent. SFCLola vs Greedy approach.

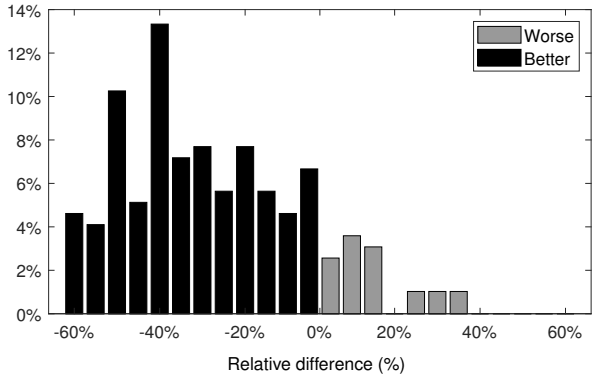

Fig. 8. Histogram of the percentage deltas computed considering all the requests sent. SFCLola vs Round Robin approach.

It is worth clarifying that all iterations start in a similar network and VNF load scenario, but different VNF selection decisions taken by the algorithms gradually change the experiment resource load. Therefore it may happen that, given a n-th request, SFCLola chooses a chain setup that is optimal for the current situation, but can show a worse latency with respect to the n-th decision taken by another algorithm in a load scenario that evolved differently.

Fig. 9 shows the absolute difference of latencies considering all the requests sent (sorted by value for sake of clarity) comparing the greedy strategy with the SFCLola approach. Most of the values show an improvement over latencies reflecting what is shown by the histogram in Fig. 7 and Fig. 8 for the comparison with the greedy and round robin strategies, respectively.

VI. Conclusions

This work tackled the problem of Service Function Chaining management within a multi-DC virtual tenant network. The proposed SFC solution can be deployed on top of the tenant network of virtual nodes (VMs) and does not require SFC-oriented support from the infrastructure, except from the basic requirement of L3 connectivity among VMs sub-networks in

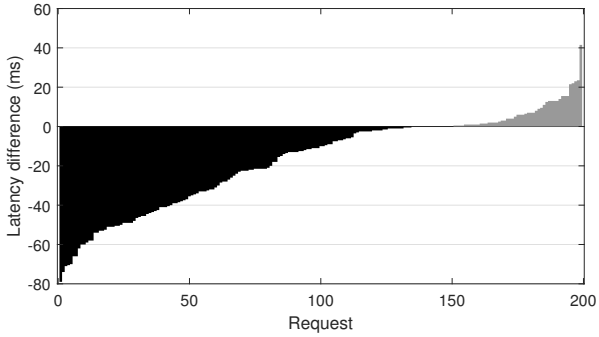

Fig. 9. Latency differences in milliseconds between SFCLola optimisation strategy and the greedy approach. Gray values represent positive differences while black negative ones (improvements).

different DCs. We presented a modular service platform where two main hierarchically related levels are defined: i) an end-to-end chain management level (called SFCLola application) that offers a service chain management REST API and performs a VNF selection decision, and ii) a forwarding mechanism (VFF) that can be programmed and enforced within the tenant domain.

The SFCLola application maintains a logical and abstract view of the network topology, minimizing the need of sensitive infrastructure information. Once SFCLola has selected the VNF instances for chain setup, appropriate forwarding instructions are sent to concerned VFFs. The VFFs expose an intent-based interface allowing to easily specify requests for installation and deletion of chain forwarding rules, independently from the specific forwarding technology. In this work, IPv4 stable Linux policy routing capabilities have been used, but the platform can be easily extended to exploit alternative technologies, such as IPv6 Segment Routing, as we plan to do as future work. Moreover, security and isolation issues are not in the scope of the paper and may be considered for future work.

Last but not least, SFCLola has been implemented as software and this work reports on the results of a test campaign performed on top of 5GINFIRE infrastructure facilities. The experiment allowed us to demonstrate the correct operation of SFCLola and evaluate the impact in terms of resource consumption at VFF nodes (which are the only SFC-aware nodes in an intra-DC portion of the tenant network). Such impact appeared sustainable, although approximately proportional to traffic load and chain number. Experimental measurements allowed us assessing the error between the estimated latency computed on the abstract topology and the measured end-to-end latency along the established chain, which resulted in an average error of 3.8%. Finally, the comparison with alternative heuristic strategies showed the benefits of the proposed approach that, on the given experiment scale, required a few milliseconds computation time. More precisely, considering a set of 250 random requests with a number of VNFs between 2 and 4 we measured an average computation time of 2.6ms and a standard deviation of 1.89ms. The experiment has been

performed using UDP traffic and software emulating a VNF processing delay, in the future experiment activities will be performed using realistic VNFs and TCP traffic.

In addition to planned activities mentioned above, future work will be devoted to improve the VNF selection algorithm to take into account bandwidth constraints at inter-DC links and evaluate the impact on previously installed and active chains when performing a VNF selection decision for an incoming service chain request. Moreover, when a request cannot be accommodated due to the maximum latency constraint, the current SFCLola implementation returns to the client with the achievable minimum latency. SFCLola could thus be extended to support a QoS negotiation mechanism allowing to partially accommodate SFC requests and optimizing resource usage.

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.

APPENDIX

Table IV shows some descriptive statistics measures derived from a population of Round Trip Time (RTT) measurements collected in a 2-days interval at 10 seconds frequency and then aggregated as average values over 30 seconds-windows by our monitoring software. Please consider that RTT values are divided by two to roughly obtain a one-way delay and values above 300 ms are filtered out. Network links between DCs are pretty stable but we experimented limited periods in which a few packets were lost and latencies jumped to values above 1 second.

TABLE IV
STATISTICS OF LATENCIES BETWEEN DCS

Link	Average	σ	95%ile	99%ile
5Tonic - ITaV	14.99 ms	4.44 ms	23.17 ms	35.38 ms
5Tonic - Bristol	20.21 ms	0.86 ms	20.67 ms	26.00 ms
5Tonic - 5Tonic-bis	1.15 ms	4.85 ms	0.83 ms	24.36 ms
ITaV - Bristol	34.76 ms	4.90 ms	43.78 ms	58.21 ms
ITaV - 5Tonic-bis	5.53 ms	5.84 ms	26.17 ms	44.76 ms
Bristol - 5Tonic-bis	20.82 ms	3.43 ms	21.50 ms	32.02 ms

REFERENCES

[1] C. Cui, H. Deng, D. Telekom, U. Michel, and 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, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[3] P. K. Chartsias, A. Amiras, I. Plevrakis, I. Samaras, K. Katsaros, D. Kritharidis, E. Trouva, I. Angelopoulos, A. Kourtis, M. S. Siddiqui, A. Vies, and E. Escalona, "SDN/NFV-based end to end network slicing for 5G multi-tenant networks," in *2017 European Conference on Networks and Communications (EuCNC)*, June 2017, pp. 1–5.

[4] ETSI, "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV ETSI GS NFV 003," https://www.etsi.org/deliver/etsi_gs/NFV-VE/001_099/005/01.01.01_60/gs_NFV-EVE005v010101p.pdf, 2015.

[5] 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.

[6] L. Geng, Slawomir, L. Qiang, K. Makhijani, A. Galis, and L. M. Contreras, "Problem Statement of Common Operation and Management of Network Slicing," Internet Engineering Task Force, Internet-Draft draft-geng-coms-problem-statement-04, Mar. 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-geng-coms-problem-statement-04

[7] H. Wang, A. Srivastava, L. Xu, S. Hong, and G. Gu, "Bring your own controller: Enabling tenant-defined SDN apps in IaaS clouds," in *IEEE Conf. on Computer Communications (IEEE INFOCOM 2017)*, May 2017, pp. 1–9.

[8] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, and T. Magedanz, "Service function chaining in next generation networks: State of the art and research challenges," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 216–223, February 2017.

[9] F. Callegati, W. Cerroni, C. Contoli, R. Cardone, M. Nocentini, and A. Manzalini, "Sdn for dynamic nfv deployment," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 89–95, October 2016.

[10] R. Munoz, R. Vilalta, R. Casellas, R. Martinez, T. Szyrkowiec, A. Autenrieth, V. Lopez, and D. Lopez, "Integrated sdn/nfv management and orchestration architecture for dynamic deployment of virtual sdn control instances for virtual tenant networks [invited]," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 7, no. 11, pp. B62–B70, Nov. 2015.

[11] M. Gharbaoui, C. Contoli, G. Davoli, G. Cuffaro, B. Martini, F. Paganelli, W. Cerroni, P. Cappanera, and P. Castoldi, "Experimenting latency-aware and reliable service chaining in next generation internet testbed facility," in *2018 IEEE Conf. on Network Function Virtualization and Software Defined Network (NFV-SDN)*, November 2018.

[12] A. Gavras, S. Denazis, C. Tranoris, H. Hrasnica, and 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.

[13] Filsfils et al., "Segment Routing Architecture," RFC 8402, https://tools.ietf.org/html/rfc8402, July 2018.

[14] A. M. Medhat, G. A. Carella, M. Pauls, and T. Magedanz, "Extensible framework for elastic orchestration of service function chains in 5g networks," in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2017, pp. 327–333.

[15] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, "The segment routing architecture," in *2015 IEEE Global Communications Conference (GLOBECOM)*, Dec 2015, pp. 1–6.

[16] A. Bashandy, C. Filsfils, S. Previdi, B. Decraene, S. Litkowski, and R. Shakir, "Segment Routing with MPLS data plane," https://datatracker.ietf.org/doc/html/draft-ietf-spring-segment-routing-mpls-18, Internet Engineering Task Force, Internet-Draft draft-ietf-spring-segment-routing-mpls-18, Dec. 2018, work in Progress.

[17] A. Abdelsalam, F. Clad, C. Filsfils, S. Salsano, G. Siracusano, and L. Veltri, "Implementation of virtual network function chaining through segment routing in a linux-based nfv infrastructure," in *2017 IEEE Conference on Network Softwarization (NetSoft)*, July 2017, pp. 1–5.

[18] D. Lebrun and O. Bonaventure, "Implementing ipv6 segment routing in the linux kernel," in *Proceedings of the Applied Networking Research Workshop*. ACM, 2017, pp. 35–41.

[19] 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, 2018.

[20] B. Martini, F. Paganelli, P. Cappanera, S. Turchi, and P. Castoldi, "Latency-aware composition of virtual functions in 5g," in *2015 IEEE 1st Conf. on Network Softwarization (NetSoft)*. IEEE, 2015, pp. 1–6.

[21] M. R. Garey and D. S. Johnson, *Computers and intractability: A Guide to the Theory of Np-Completeness*. NeW York: W.H. Freeman, 1979.

[22] "Linux Advanced Routing & Traffic Control HOWTO," [Online]. Available: http://lartc.org/howto/. Accessed 21 March, 2019.