# An experimental study on latency-aware and self-adaptive service chaining orchestration in distributed NFV and SDN infrastructures

M. Gharbaoui[a,*], C. Contoli[f], G. Davoli[b,c], D. Borsatti[b,c], G. Cuffaro[c], F. Paganelli[c,d], W. Cerroni[b,c], P. Cappanera[e], B. Martini[c,g]

[a] *Scuola Superiore Sant'Anna, Pisa, Italy*
[b] *University of Bologna, Italy*
[c] *CNIT, Italy*
[d] *University of Pisa, Italy*
[e] *University of Florence, Italy*
[f] *University of Urbino "Carlo Bo", Italy*
[g] *Universitas Mercatorum, Rome, Italy*

**Abstract**

Network Function Virtualization (NFV) and Software Defined Networking (SDN) changed radically the way 5G networks will be deployed and services will be delivered to vertical applications (i.e., through dynamic chaining of virtualized functions deployed in distributed clouds to best address latency requirements). In this work, we present a service chaining orchestration system, namely LASH-5G, running on top of an experimental set-up that reproduces a typical 5G network deployment with virtualized functions in geographically distributed edge clouds. LASH-5G is built upon a joint integration effort among different orchestration solutions and cloud deployments and aims at providing latency-aware, adaptive and reliable service chaining orchestration across clouds and network resource domains interconnected through SDN. In this paper, we provide details on how this orchestration system has been deployed and it is operated on top of the experimentation infrastructure provided within the Fed4FIRE+ facility and we present performance results assessing the effectiveness of the proposed orchestration approach.

*Keywords:* SDN; NFV; Orchestration; Federated testbed; Service chaining.

*Corresponding author
*Email address:* `molka.gharbaoui@santannapisa.it` (M. Gharbaoui)

## 1. Introduction

Novel vertical applications emerging in different industry fields, such as e-health, hyper-connected smart cities, and industrial automation, will improve several aspects of society and human lives by taking advantage of the features offered by 5G (and beyond) mobile networks, IoT, and pervasive cloud deployments (i.e., edge micro-clouds in combination with traditional clouds) [1]. This scenario will foster new business opportunities for telco service providers, which will be able to address the increasingly stringent service requirements from vertical applications in terms of reliability, availability and latency performance [2][3].

Software-Defined Networking (SDN) [4] and Network Function Virtualization (NFV) [5] promise to satisfy such operational requirements by fostering a more flexible network service deployment thanks to virtualization technologies and network programmability (i.e., *softwarization*) [6][7][8]. NFV allows for elastic network service deployments as virtual partitions out of a convergent cloud-network infrastructure. Such *network slices* are composed of dynamically-established Virtual Network Functions (VNFs) and virtual link interconnections. The micro-clouds at the edge can be also exploited to elastically deploy VNFs that need a tight interaction with the users to support latency-sensitive applications with, e.g., prompt traffic optimizations or bit rate adaptations. In this scenario, SDN can effectively provide programming abstractions that can be exploited for the dynamic enforcement and in-line steering of data traffic through (virtual) network paths.

With the *softwarization* of telecommunication infrastructures and the increasing network management abstractions, service paths and workflows can be established in a more efficient way by flexibly composing VNFs and dynamically interconnecting them through SDN (i.e., *service chaining*) [9][10][11]. This requires that telco service providers effectively put in place dynamic resource provisioning and orchestration mechanisms on top of a distributed NFV/SDN infrastructure to (i) assure service pervasiveness and timeliness, (ii) minimize latency, and (iii) maximize service availability [8]. However, the heterogeneity of the infrastructures, the high dynamicity of services and the geographical distribution of cloud sites pose new challenges from a system integration perspective in terms of (i) resource control/management capabilities, (ii) adaptive usage of multi-technology resources, and (iii) fulfillment of end-to-end latency requirements considering the impact of both processing and network delays in distributed infrastructures [12][13].

In this work, we present the outcome of a joint integration effort that brought together diverse solutions and technologies to run a latency-aware and adaptive service chaining mechanism in a distributed SDN/NFV infrastructure fully regulated by an end-to-end orchestration system, i.e., LASH-

5G. Such a service chaining orchestration system was conceived to run over geographically distributed edge clouds interconnected through SDN (i.e., spanning multi-domain/multi-technology infrastructures) and establish service chains relying on domain orchestration solutions to address latency, adaptability and availability requirements. More specifically, the proposed orchestration system takes advantage of heterogeneous and enhanced resource control/management capabilities offered by the underlying cloud and network resource domains to dynamically provide service chains while (i) optimally selecting VNFs over the path that minimizes the offered end-to-end latency across the clouds and the network resource domains; (ii) triggering the set-up and update of (pieces of) service chains in each of the underlying resource domains; (iii) promptly adapting established service chain paths to address reliability based on the current load of the network (i.e., self-adaptive service chaining); and (iv) continuously collecting monitoring data from underlying resource domains so as to promptly trigger chain adaptation actions and to effectively make latency-aware VNF selection decisions.

This paper also presents performance results of the proposed service chaining orchestration system collected through a large set of experiments carried out using a testbed reproducing a composite yet realistic 5G infrastructure as a distributed NFV deployment interconnected through a SDN-based WAN. The experiments were performed on top of the Fed4FIRE experimentation infrastructure provided within the Fed4FIRE+ Horizon 2020 Project, offering a federation of open, accessible and high-available testbed facilities to support a wide variety of different research and innovation activities, including 5G-related experimentations and testing over heterogeneous systems [14]. The experimental tests aim at assessing the actual performance of the orchestration system in addressing service chain requests, in computing latency-optimized service chains by correctly processing monitoring data, in enforcing chaining decision while leveraging the network and cloud resource control functionalities, and, finally, in dynamically adjusting established service chain paths to react to network congestions.

The main contribution of this work is the design, development and deployment of a latency-aware and adaptive service chaining orchestration system as a result of an integration effort of different SDN/NFV solutions and technologies, which makes the proposed system suitable for end-to-end deployment in real scenarios. Indeed, we provide end-to-end service chaining orchestration in combination of fine-tuned network resource orchestration on top of distributed SDN/NFV deployments interconnected through a SDN-based WAN running in an experimental testbed with OpenStack and different SDN controllers (i.e., ONOS, Ryu). The most relevant enabling integration aspects that were addressed to achieve the goal include: *i)* the definition of common high-level interfaces to abstract from solutions dependent on the specific infrastructures, *ii)* the definition of relevant common performance metrics together with the deployment of the related monitoring systems, and *iii)* the integration of the involved experimental platforms.

This work extends the authors' previous work by deploying and integrating the single elements of the end-to-end service chaining orchestration system that were presented separately and evaluated through simulations or laboratory testbeds [15][16][17]. In the integrated experimentation we took advantage of the bare-metal experimentation capabilities provided by the Fed4FIRE framework, and were able to use a significant amount of physical nodes available in one of the federated testbeds as we did in [18]. The exclusive use of physical servers allowed us to experiment the integration of our orchestration system on a distributed NFV (i.e., cloud) environment featured by SDN network capabilities. With respect to [19, 20] we present more performance results that we collected through a comprehensive testing campaign.

The remainder of this paper is organized as follows. Section. 2 discusses related work and highlights our contribution with respect to the state of the art. In Section. 3 we present the service chaining orchestration reference scenario. In Section. 4 we describe the architecture of the proposed service chaining system. In Section. 5 we detail the workflows of the LASH-5G orchestration system. In Section. 6 we report on first experiences in the deployment of our system on top of Fed4FIRE+ federated testbeds and in Section. 7 we provide feedbacks from our experimenters' point of view. Finally, Section. 8 concludes the paper providing insights on future work.

## 2. Related Work

Several works in the literature show that the adoption of SDN/NFV, possibly in combination with an orchestration layer, provides increasing flexibility and scalability to dynamic service chaining. Such works typically provide SDN-based solutions for steering packets across service functions. In particular, [21] confirms the capabilities of SDN in supporting flexible service chaining and insists on the need for a framework that handles the lifecycle management of service function chains, as the one we propose in this work. [22] proposes a service-oriented SDN controller that deploys programmable data delivery routes connecting multiple chains of VNFs. The work mainly focuses on the networking aspects of service chains and neglects the dynamics that characterize the edge cloud domains. On the contrary, in this paper we introduce a centralized entity that takes into account both the cloud and WAN resources for a more efficient deployment of the VNFs. [23] also mainly focuses on traffic steering through a given set of VNFs. Moreover, authors solve the VNFs placement and traffic flows separately while in this work we propose an optimization algorithm that jointly addresses both problems. [24] proposes the NIMBLE architecture for SDN-based middlebox management. The proposed approach addresses challenges relevant to load-balancing and middlebox composition. However, the focus is only on traffic re-routing and flow-entries limitation. Finally, as discussed in [21] a number of research challenges are still open, especially taking into account the availability status of the network (e.g., switch/link capacity usage) that might affect the actual QoS performance (e.g., throughput) experienced by data while traversing service chain paths. For this reason, as we do in this work, recently a number of research works tackled different aspects of the service chain orchestration as an effective solution to dynamically address service requirements from applications and users [25].

Related work in the area of service chaining and VNF selection assumes that the placement of VNFs has already been done and focuses on the chaining problem [26]. In [27] a heuristic is proposed to select VNF instances in a multi-DC environment by accounting both for load balancing across instances and latency requirements. However, only the network latency is considered, disregarding the processing delay introduced by functions. In [28] an integer programming model and a Markov approximation based algorithm are proposed to solve the middlebox selection and routing problem with the objective of maximizing the total throughput over all target flows, but the problem accounts for the selection of one middlebox per flow and does not handle chains of middleboxes as our work does. The work presented in [29] deals with a different optimization problem (i.e., the minimization of the total link occupation bandwidth) and does not handle ordered sequences of VNFs as our system does. A joint network and server load balancing algorithm is proposed in [30] which first adopts a greedy strategy to construct service chains and then attempts to improve the obtained solution through a searching technique. However, the proposed model and implementation prototype are designed for an intra-DC network, while our solution targets a multi-DC environment. A VNF selection strategy is proposed in [31] that aims to maximize an overall system efficiency metric. This approach is evaluated in an emulated environment leveraging Mininet and OpenDayLight. A time-efficient solution of the VNF selection problem is targeted in [32] by proposing a set of heuristics based on the Lagrangian relaxation of an integer minimum cost flow problem. More recently, Pei et al. [33] formulated a VNF selection and routing path calculation problem and solved it by using Deep Belief Networks and an optimal algorithm to generate training data. With respect to this related work, our VNF selection strategy adopts an abstracted network model which requires minimal topological and monitoring information of infrastructure resources. In a multiple domain scenario, network infrastructure operators may decide to hide internal implementation details, thus exposing an abstracted view of the infrastructure status and fine-tuning deployment decisions within their own domain. Our approach thus well accommodates such scenario, since our VNF selection algorithm leverages an abstracted topology to select VNFs from multiple DCs, while VIMs and WIMs implement the service chaining instructions leveraging full disclosure of their domain.

In [34] authors provide a 5G/Edge-based service chaining orchestration solution with NFV and SDN and elaborate on a resilient and adaptive framework, following an approach inspired by software-defined networking. Authors address a different problem in allowing users to construct network service chains in the mobile and edge computing environments and in considering locality criteria and policy rules while embedding service chains. A similar approach is used in [35] where authors present a network service chaining model in 5G wireless architecture that integrates cloud and fog computing with advanced technologies such as SDN and NFV. With respect to our work, they specifically focus on data management and security aspects, and present performance results obtained through simulations. In [36] Authors theoretically formulate dynamic network service chaining in SDN/NFV infrastructures. They formulate an optimal algorithm that iteratively decides the sending rate of service and multi-path routing for dynamic service chaining in an integrated computational and network resource environment with also considerations on cost-utility trade-off. With respect to our work, they address different aspects of dynamic service chaining (e.g., incoming and departure data rate, maximize service utility). However, as the previous one they present performance results of the proposed algorithm obtained through simulations. In addition, all the above works do not consider latency-based criteria, nor at the network neither at the computational level as this works does.

Additional works on service chaining orchestration systems and solutions consider latency performance of service chains with also reliability features. In [37], authors formulate an ILP model for SFC placement and resource allocation while trying to address different end-to-end latency requirements. An SFC controller is proposed in [38] to optimize the placement of service chains in Fog environments while reducing the end-to-end latency. Finally, latency optimization techniques for virtualized resource allocations in distributed network edge environments are presented in [39].

With respect to our work, none of all the above proposed dynamic service chaining solutions were experimentally assessed as we do in this work, with all the implications and efforts required to develop and deploy a realistic integrated system. Indeed, we could deploy a set-up with OpenStack and different SDN controllers and perform fine-tune settings and monitoring of both virtual functions and network resources and, ultimately, decide service chaining based on a dynamic view of underlying virtual infrastructure, in terms of latency and switch load, in such a practical set-up.

In terms of experimentation in SDN/NFV distributed environments, noteworthy initiatives have been carried out within FIRE (Future Internet Research and Experimentations) in Europe [40] and GENI (Global Environment for Network Innovations) in the U.S. [41]. In particular, such initiatives offer SDN and Cloud facilities to enable experiments on emerging network, service and application scenarios, such as Next Generation Internet (NGI) (e.g., Fed4FIRE [42][43]) and 5G (e.g., SoftFIRE [44], 5GinFIRE [45]). In the context of SoftFIRE, a framework for elastic orchestration of service chains is proposed in [25]. Although authors present their solution integrated with an ETSI-compliant NFV management and orchestration platform (i.e., OpenBaton [46]), they do not address neither the inter-DC scenario including a Wide Area Network (WAN), nor latency-awareness and reliability features for established service chains. Other relevant experimental works in this topic have been carried out within EU-funded research and innovation projects. However, the deployment used for experiments does not include the WAN (i.e., inter-DC) network scenario [47] or does not reproduce realistic cloud and network deployments [48][49] since making use of emulation platforms at data plane level, e.g., Mininet. On the other hand, some experimental initiatives have tackled the deployment and the management of vertical services that may span multiple provider domains [50][51]. However, in these works, neither latency-awareness features nor end-to-end monitoring to guarantee reliability are addressed for established service chains. Moreover, fine-tuned network programmability is not exploited in the cloud as this work does, leveraging a full SDN control at end-to-end level.

In conclusion, this work advances the state of the art since it performs experiments in end-to-end service chaining orchestration in combination of fine-tuned network resource orchestration using a testbed reproducing a composite yet realistic 5G infrastructure as distributed SDN/NFV deployments interconnected through a SDN-based WAN. In addition, it evaluates both cloud processing delays and network delays while addressing end-to-end latency requirements. Finally, the proposed mechanism goes beyond the coordinated establishment of cloud and network services, including also adaptation actions with respect to the current availability status of network resources in order to mitigate issues (e.g., congestion) derived from their concurrent usage by a multitude of services. On top of that, such contributions have been obtained in an experimental testbed with OpenStack and different SDN controllers (i.e., ONOS, Ryu) to demonstrate and discuss dynamic service chaining in a practical set-up through (i) fine-tuned network settings in the cloud, (ii) end-to-end full SDN control and (iii) monitoring of both virtual functions and network resources to optmimally decide VNF selections and to promptly react based on a dynamic view of the underlying virtual infrastructure.

## 3. Reference Scenario

In this section, we present the reference scenario that motivates the use of a service chaining orchestration system in NFV/SDN infrastructures.

NFV/SDN deployments of telco service providers typically feature multi-site edge clouds (i.e., DCs) interconnected via a WAN. On top of this virtualized and programmable telco infrastrucure, network slices are deployed that are composed of VNFs (e.g., load balancer, packet inspection, traffic optimization, content servers, caching) running at edge clouds and interconnected through virtual links. Examples may include (i) the provisioning of VNFs for content delivery during a live sport event (e.g., mixer, transcoder, compressor) with end-user traffic steered across them while minimizing the cost and satisfying QoS requisites [52], (ii) the deployment of service chains across VNFs in Fog environments for video surveillance or smart waste management services while optimizing resource allocations and latency performance [38]. It worths pointing out that real network slice deployments for verticals typically require that multiple instances of each kind of required VNFs are deployed in network slices. As shown in Fig. 1, these VNF instances are deployed in different edge clouds for reliability or for pervasiveness reasons (i.e., increase service coverage over a wider geographical area, run back-up services for high-critical applications, serve users with lower latency performance)[53].

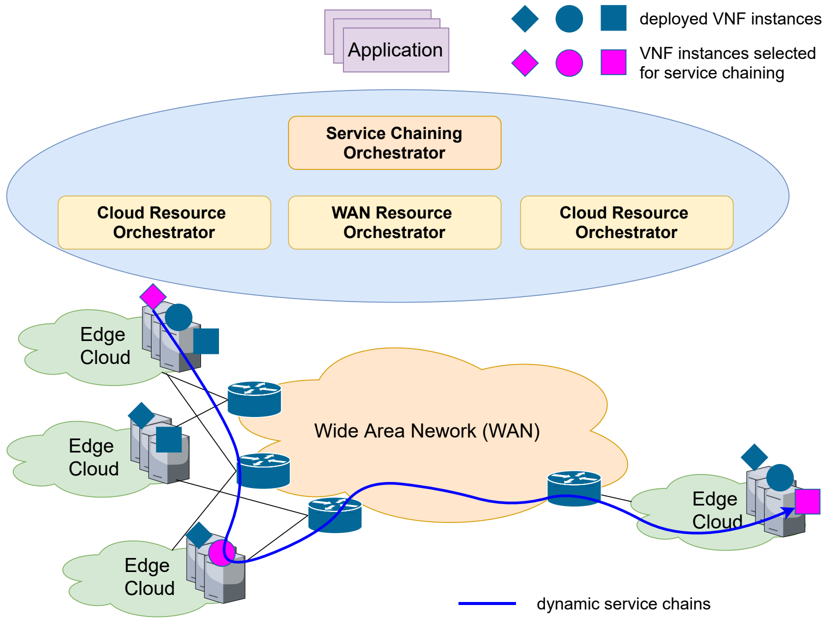

Application
deployed VNF instances
VNF instances selected for service chaining
Service Chaining Orchestrator
Cloud Resource Orchestrator
WAN Resource Orchestrator
Cloud Resource Orchestrator
Edge Cloud
Edge Cloud
Edge Cloud
Edge Cloud
Wide Area Nework (WAN)
dynamic service chains

Figure 1: Service chaining orchestration scenario.

Telco service providers can achieve the most out of network slicing if they can effectively and dynamically establish services for vertical applications as dynamic service chains on top of VNFs while addressing specified requirements (e.g., QoS, latency constraints, security). In particular, it is important to assure that VNFs are traversed by user traffic in a specified order and that such an order is also adapted based on real-time status information according to the current context (e.g., load by users traffic, latency performance) [10] in order to guarantee appropriate user experience. The service chain requests are expected from verticals (mediated by the OSS/BSS) in terms of an ordered sequence of VNF types to traverse and QoS demands for interconnections (abstract service chain). Those chain specifications are the result of verticals' internal operations carried out within their application service delivery activities (e.g., as part of optimizations through Service Oriented Architecture principles).

In this scenario, as shown in Fig. 1, two levels of orchestration are needed to allow latency-aware and adaptive service chaining in distributed NFV/SDN infrastructures.

Firstly, beyond appropriate decisions on where to place VNFs (out of scope in this paper), there is the need to orchestrate VNF instances across distributed edge clouds to establish service chains dynamically and concurrently while addressing specified requirements (i.e., *service chaining orchestration*). Indeed, the most proper VNF instances need to be selected (out of multiple running instances) for each type of VNF and in the order specified in the request. In addition, the proper network paths supporting virtual links need to be set-up while assuring QoS demands (e.g., required bandwidth). At this level, different targets can be pursued while doing the VNF selection (e.g., minimize the end-to-end latency) or while operating the chains (e.g., adapt running chains based on the context by adding or removing VNFs ).

Secondly, there is the need to orchestrate the virtual resources offered by each resource domain, (i.e., processing resources in edge clouds and network resources in the interconnection WAN) to support the set-up and the operation of service chains according to the requirements. For this reason also resource-level orchestrators are envisioned, i.e., *cloud resource orchestrator*, responsible for managing (part of) chains running within edge cloud DCs, and the *WAN resource orchestrator*, responsible for managing traffic flows to reach DCs that host VNF instances. SDN can be used in both domains as a network technology to offer network abstractions and programmability to dynamically steer data traffic and enforce traffic rules in the network nodes.

A coordination among the resource-level and service chaining orchestrators is needed to (i) instruct resource-level orchestrators on how to realize (pieces of) chains in the infrastructure, and (ii) inform service chaining orchestrator about performance offered by the infrastructure (e.g., delays at processing and network levels) and about any relevant changes in chain configurations (e.g., network paths supporting virtual links).

In the following sections, we describe the two-levels orchestration system (i.e., LASH-5G system) we developed and put in operation in a distributed NFV/SDN infrastructure set-up to achieve latency-aware and adaptive service chaining.

## 4. Service Chaining Orchestration System Design

The general architecture of the proposed LASH-5G orchestration system is shown in Fig. 2. It has been designed starting from the principles of the ETSI NFV MANagement and Orchestration (MANO) framework [54], enriching them with service chaining orchestration and original resource orchestration features in distributed (i.e., multi-site) NFV/SDN environments.

The LASH-5G system is devised to run on top of an NFV Infrastructure (NFVI) composed of edge cloud SDN Domains, i.e., Data Center (DC) domains where VNFs are deployed by means of a cloud-computing platform (e.g., OpenStack) and where SDN is used as network control technology. An SDN WAN infrastructure domain is assumed to provide inter-DC connectivity. Each SDN domain adopts its own controller, which is assumed to provide a fully-featured northbound interface (e.g., via REST API) that allows to dynamically program traffic flow steering rules across network nodes.

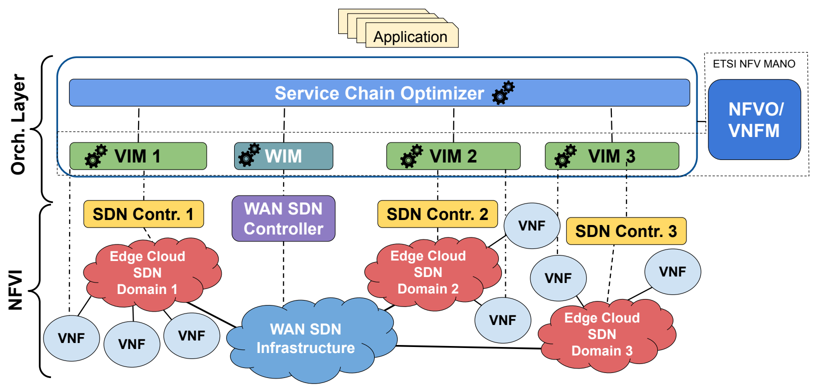


Figure 2: Orchestration system architecture for latency-aware and adaptive service chaining orchestration.

The Orchestration layer is conceived as an extension of ETSI MANO framework [54] including NFV Orchestrator (NFVO), VNF Manager (VNFM), Virtual Infrastructure Manager (VIM) and WAN Infrastructure Manager (WIM) functionalities as well as additional functional blocks proposed in this work to enable latency-aware and self-adaptive service chaining orchestration. As per the standard NFV architecture [54], the NFVO is in charge of coordinating operations to provide needed virtual resources for Network Services and VNFs deployments. The VNFM is responsible for the lifecycle management of the VNFs instances i.e., instantiation, configuration, scaling, upgrade and termination operations. The VIM and the WIM are the management/control entities in charge of provisioning and configuration of (i) compute, storage and network resources required by VNFs in the data centers/cloud domains, and (ii) network resources in the WAN domain required by virtual links, respectively.

The proposed additional functional blocks are assumed to supplement NFVO and VNFM functionalities with dynamic service chaining capabilities and adaptability as part of advanced Network Service lifecycle management extended to service chains [25].

Firstly, the Chain Optimizer (CO), as service chaining orchestrator, supports NFVO with service chaining request handling and with optimal selection of service VNF components. As discussed in the ETSI GS NFV-EVE specification document [55], several architectural options exist in the MANO architecture for dynamic management of service chaining, including the operation of a service chaining application. In this context, in LASH-5G, the CO acts as a service chaining application that, leveraging network connection topology information in terms of VNF instances and virtual links descriptors received from the MANO stack, selects the VNF instances and virtual links to setup optimised forwarding paths for different traffic flows. These chaining rules are then delivered to the VIMs and WIM to actually establish service chains (including the virtual links) in turn leveraging SDN controllers in the respective domains. Thus, with respect to the architectural options discussed in [55], the CO plays the role of a SDN service chaining application which, however, is not tightly bounded to a specific controller. Instead, it leverages the application-control interfaces exposed by the VIM and WIM orchestrators. This level of abstraction is required since the CO is a functional block that enhances the MANO stack with the capability of dynamically specifying the forwarding path of a network service for target flows to cope with QoS-aware optimization objectives and distributing the associated forwarding instructions to WIM/VIMs.

Secondly, the VIM and WIM are actually extended management functionalities that go beyond ETSI NFV MANO specifications [56] [57]. Indeed, they include additional features and orchestration capabilities such as advanced monitoring, intent-based interface and adaptive configurations of (part of) service chains.

Compared to the WIM as described in the ETSI NFV MANO specifications, the WIM orchestrator uses the intent-based approach to provide the automation in configuring the SDN network and in keeping its status according to the expressed intent. Indeed, the WIM orchestrator is enhanced with a monitoring module that periodically checks the status of the switches in the WAN, and in case of performance degradation autonomously triggers the automatic redirection of the service chain paths supporting intents. In [57] the performance monitoring of multi-site connectivity services is stated to be supported by a performance management interface. However, only monitoring data on network links are contemplated and neither operations nor functionalities on how to use those monitoring data are envisioned. In line with ETSI specifications yet enhancing the ETSI MANO framework, the WIM orchestrator leverages the monitoring interface at the SDN controller to collect network statistics (not only related to the network links but also to network devices, established flows) and use those data to adapt the installed flows to the current network status thus avoiding degradation.

Compared to the VIM as described in the ETSI NFV MANO specifications, the VIM orchestrator offers enhanced abstractions and service chain management capabilities that make it more suitable for interacting with a CO that is unaware of the implementation details in each DC/cloud domain. In particular, the VIM orchestrator is capable of deploying a whole service chain (or part of it) in a given DC/cloud domain by properly selecting the required set of VNFs and applying the necessary SDN traffic steering rules, all with a single interaction with the CO.

The operation of the CO and of the extended VIM/WIM (i.e., VIM/WIM orchestrators) are detailed in the following subsections.

## *4.1. Chain Optimizer*

The Chain Optimizer (CO) is a service chaining orchestration engine that receives service chain requests from the NFVO (on behalf of the OSS/BSS) and selects available VNF instances to setup the service chains so that QoS requirements are fulfilled and the end-to-end latency is minimized.

The selection decision is made by executing a VNF selection algorithm. The CO can host multiple selection algorithms. In the current version, it contains the implementation of a VNF selection optimization algorithm proposed in a previous work [15] and a greedy algorithm implementation used for comparative evaluation (described in Section 7). This algorithm receives chain request information and infrastructure monitoring data as input, and solves the problem of selecting the VNF instances that minimize an estimated end-to-end latency calculated considering both processing delays and network delays. This optimization problem is formulated as a Resource Constrained Shortest Path problem on an auxiliary layered graph. The algorithm works using an abstracted view of the underlying infrastructure topology, which is built by periodically collecting inter-DC latency values, type and processing latency of VNF instances deployed at each DC, from NFVI and VNF monitoring API, respectively. The solution provided by the algorithm, if existing, specifies for each VNF in the chain request which DC should provide the corresponding VNF instance so that QoS requirements (maximum latency and minimum bandwidth) are satisfied and the end-to-end latency is minimized. The algorithm has been implemented in C++ and uses the IBM ILOG CPLEX library for linear programming. Once a solution for the VNF selection problem is found, the CO sends appropriate chain setup instructions to the WAN and DC domain resource orchestrators to setup the service chain. The CO interacts with these domain orchestrators through high-level, intent-based REST APIs specifically devised to abstract from infrastructure-dependent solutions, fostering better integration toward end-to-end deployment. Such APIs allow the CO to send instructions for managing the lifecycle of a service chain (i.e. creation, updating and deletion) and enforcing traffic steering operations using an application-oriented semantic, rather than dealing with technology-specific, low-level network details. The CO is implemented as a Java application that offers a REST API for CRUD (Create, Read, Update, Delete) operations on service chains.

## *4.2. WAN Infrastructure Manager Orchestrator*

The WAN Infrastructure Manager (WIM) Orchestrator is an SDN-enabled WAN domain orchestration logic running on top of an Open Network Operating System (ONOS) controller and providing the programmable provision of service chain paths across the WAN by means of an intent-based northbound REST interface [58]. More specifically, it supports the use of a template-based approach where a simple JSON message can be filled in by the CO to specify the parameters necessary for the configuration of the service chains. Hence, the set-up of service chain paths in the WAN to connect VNFs in different DCs can be triggered by the CO by specifying to the WIM orchestrator the list of DCs to be traversed. Then, the WIM orchestrator derives the DC domain gateways to be connected and performs mapping operations by identifying the network path and, accordingly, enforces the forwarding rules to the switches along the identified path. In line with [59], the WIM orchestrator also offers reliable service chains by adapting (i.e., redirecting) service paths, or a segment thereof, to recover from network congestion events, with an overall benefit in terms of high-availability and effective resource utilization. For this purpose, the WIM orchestrator monitors the switches load status by deriving switch link throughput from statistics periodically collected and processed by the SDN controller. Then, those statistics are given as input to a load-balancing algorithm, which compares the current switches load to a fixed threshold, and in case, redirects the flows traversing overloaded switches to other available switches in the network, thus avoiding any congestion. Finally, the WIM is also responsible for the collection of network latency information in order to retrieve the inter-DCs delays. Those delays are then made available to the CO to enhance its resource orchestration capabilities by computing a minimum-latency service graph. The details of the WIM design and the supported northbound interface specification can be found in [16][60].

### *4.3. Virtual Infrastructure Manager Orchestrator*

The Virtual Infrastructure Manager (VIM) Orchestrator is an SDN-enabled DC/cloud domain orchestration logic providing advanced network management capabilities in cloud computing environments. The VIM orchestrator exposes an intent-based northbound REST interface that allows to specify a service chain by means of a high-level descriptive syntax, agnostic to the specific SDN technology adopted [17]. This makes it suitable to manage different DC domains in a multi-technology environment, e.g., leveraging different SDN controllers. The VIM orchestrator is also capable of dynamically applying changes to an existing service chain without having to delete and re-deploy it from scratch. This allows to dynamically adapt service chains to the current context of users or services (e.g., current location of users in a mobility scenario) or to varying needs of the service provider (e.g., different resource management policy), and, ultimately, to avoid or prevent SLA violations. Furthermore, the REST API provided by the VIM orchestrator allows the CO to collect information about the currently deployed VNFs and their estimated processing latency, computed based on the current workload. The details of the VIM orchestrator design and the intent-based chain specification can be found in [61][17].

## 5. Orchestration Workflows

In Fig.3, we show the workflows of LASH-5G orchestration system while performing the three key operations for latency-aware and adaptive service chaining orchestration. First, we present the *service chain deployment* related to the operations for the service chain set-up involving the CO and the WIM/VIM orchestrators. Second, we present the workflows for adapting established service chains based on the current context. More specifically, we show the *service chain update* with the adding/removing of VNFs to previously established chains to address changing needs of the operators or different user demand profiles. Third, we present the *service chain network adaptation* with the dynamic tuning of the network paths in the WAN supporting virtual links based on the current status (e.g., load) of the network.

Such procedures are ETSI-compliant from the architectural point of view according to [55][57] while offering an advance to the ETSI MANO framework in terms of specific workflows for both SDN-based service chain deployment/update operations and for SDN-based network adaptation operation. Those advances are supported by the peculiar features of CO, VIM and WIM orchestrators described in Section 4.1, 4.2, 4.3.

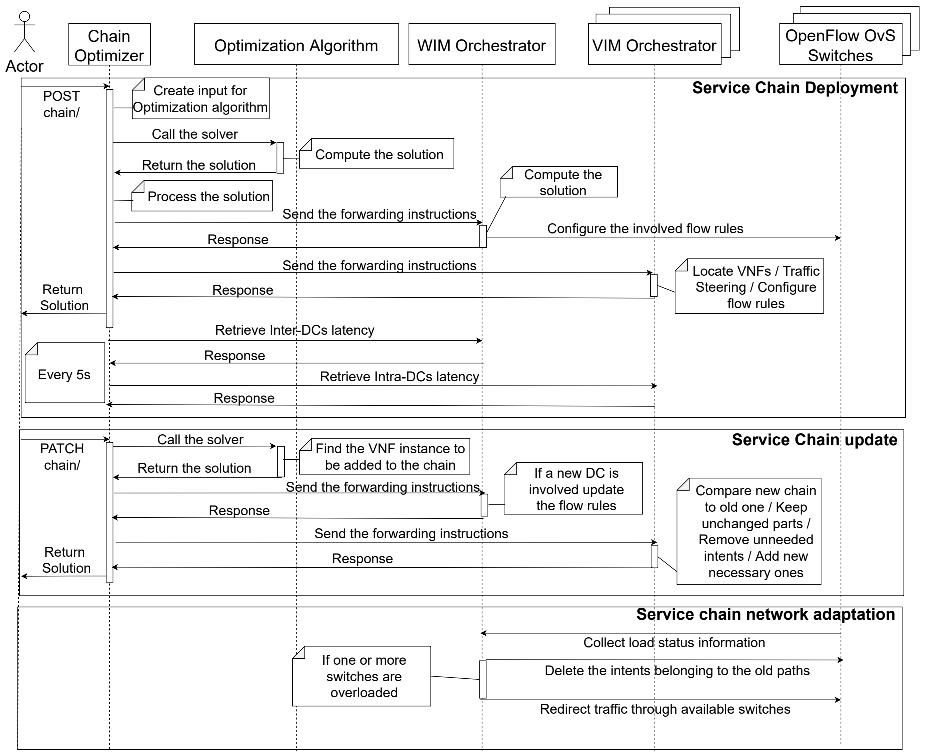
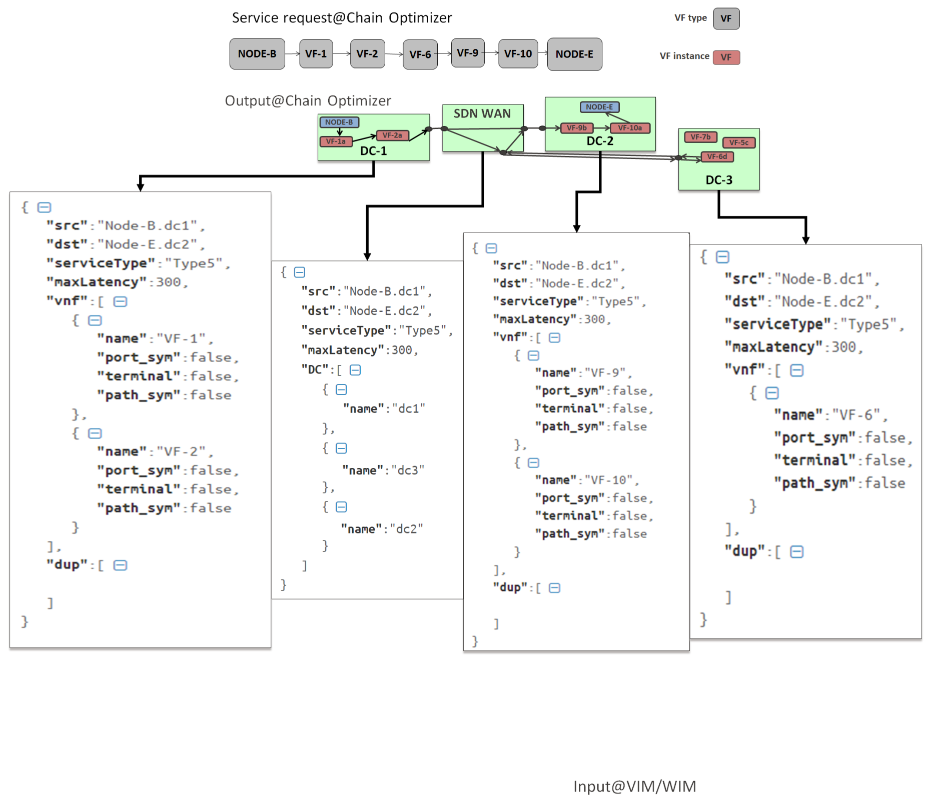

Actor
Chain Optimizer
Optimization Algorithm
WIM Orchestrator
VIM Orchestrator
OpenFlow OvS Switches
Service Chain Deployment
POST chain/
Create input for Optimization algorithm
Call the solver
Compute the solution
Return the solution
Process the solution
Compute the solution
Send the forwarding instructions
Configure the involved flow rules
Response
Send the forwarding instructions
Locate VNFs / Traffic Steering / Configure flow rules
Response
Return Solution
Retrieve Inter-DCs latency
Response
Every 5s
Retrieve Intra-DCs latency
Response
Service Chain update
PATCH chain/
Call the solver
Find the VNF instance to be added to the chain
Return the solution
If a new DC is involved update the flow rules
Send the forwarding instructions
Response
Compare new chain to old one / Keep unchanged parts / Remove unneeded intents / Add new necessary ones
Send the forwarding instructions
Return Solution
Response
Service chain network adaptation
Collect load status information
If one or more switches are overloaded
Delete the intents belonging to the old paths
Redirect traffic through available switches

Figure 3: Orchestration Workflows for service chain deployment, service chain update and network adaptation

## *5.1. Service chain deployment*

This operation starts by generating a set of service chain requests with different requirements in terms of bandwidth and maximum latency and sending them to the CO through a CO-Client GUI. The CO handles each request and computes a solution of the VNF selection optimization problem instance by executing the VNF Selection optimization algorithm. If this step is successful, a response is returned to the CO-Client GUI with the solution (i.e., set of cloud domains hosting the selected VNFs to be connected) along with the computed end-to-end latency and computation time of the optimization algorithm. In order to setup the computed chain, the CO generates a set of instructions from the computed solution for the WIM and the VIMs. Fig. 4 shows an example for a service chain of 5 VNFs. The solution assigns the first two VNFs to DC-1, the third VNF to DC-3 and the fourth and fifth VNF to DC-2. The figure shows the corresponding JSON messages that the CO sends to the VIMs and WIM through the REST APIs offered by these orchestrators. The WIM receives the ordered sequence of cloud domains to be connected across the WAN. It computes the network path(s) and, ac-



```
{
  "src":"Node-B.dc1",
  "dst":"Node-E.dc2",
  "serviceType":"Type5",
  "maxLatency":300,
  "vnf":[
    {
      "name":"VF-1",
      "port_sym":false,
      "terminal":false,
      "path_sym":false
    },
    {
      "name":"VF-2",
      "port_sym":false,
      "terminal":false,
      "path_sym":false
    }
  ],
  "dup":[

  ]
}
```

```
{
  "src":"Node-B.dc1",
  "dst":"Node-E.dc2",
  "serviceType":"Type5",
  "maxLatency":300,
  "DC":[
    {
      "name":"dc1"
    },
    {
      "name":"dc3"
    },
    {
      "name":"dc2"
    }
  ]
}
```

```
{
  "src":"Node-B.dc1",
  "dst":"Node-E.dc2",
  "serviceType":"Type5",
  "maxLatency":300,
  "vnf":[
    {
      "name":"VF-9",
      "port_sym":false,
      "terminal":false,
      "path_sym":false
    },
    {
      "name":"VF-10",
      "port_sym":false,
      "terminal":false,
      "path_sym":false
    }
  ],
  "dup":[

  ]
}
```

```
{
  "src":"Node-B.dc1",
  "dst":"Node-E.dc2",
  "serviceType":"Type5",
  "maxLatency":300,
  "vnf":[
    {
      "name":"VF-6",
      "port_sym":false,
      "terminal":false,
      "path_sym":false
    }
  ],
  "dup":[

  ]
}
```

Figure 4: Example of interworking between the Chain Optimizer and the VIMs/WIM Orchestrators

cordingly, sets-up the forwarding rules in the involved switches. Then, the CO sends the forwarding instructions (i.e., the ordered sequence of VNFs to connect) to each involved VIM. The VIMs are responsible for discovering in which compute nodes the VNFs instances with minimum processing latency are located in order to setup the part of chain they are responsible for and properly enforce traffic steering by consistently configuring the flow rules. According to the JSON format of the forwarding instructions provided by the CO, each VIM selects the VNF instances of the specified types (e.g., VNF-1 and VNF-2 in DC-1) with minimum processing latency and installs traffic steering rules for a specific flow (e.g., based on the specified src and dst fields) in the specified order (e.g., first VNF-1, then VNF-2 in DC-1).

The CO periodically retrieves latency measurements in terms of inter-DC network delays and VNF instances processing delays to update its topology view and to recompute the estimated end-to-end latency of deployed chains. The polling period is configurable through a yaml configuration file and has been set to 5s in the experiment setup described in Section 6. The CO polls directly the REST APIs exposed by the WIM to retrieve the latest inter-DC latency measurements. Analogously, the CO retrieves intra-DC latency measurements by polling the URI provided by each DC. As detailed in the experiment setup description in Section6, we used the implementation of a time series database provided by Gnocchi. In the current version of the CO possible latency violations of deployed chains are detected and notified in the CO-Client GUI. While the end user may trigger update or delete actions through the GUI, the implementation of automated adaptation policies in the CO will be investigated in future research.

### *5.2. Service chain update*

This operation regards the updating of an established service chain upon a request to face with changes in operator needs or user demands. In this workflow, the case of a request for adding a new VNF to a previously established chain is considered, while keeping the rest of the chain unchanged.

In order to trigger such a chain update, the CO-client sends an update request by specifying the id of the chain, the new VNF type and its ordered position in the chain. The CO handles the request by invoking the optimization algorithm to select the VNF instance to be added to the chain, taking into account how the pre-existing chain has been deployed. The CO then processes the algorithm output to provide appropriate instructions to WIM and VIMs for updating the chain accordingly. Specifically, the WIM is updated whenever the updated chain needs to traverse an edge cloud data center that was not involved in the original chain deployment, while the VIMs process update requests by comparing the new chain against the old one: parts of the chain that remain unchanged are kept as they are, parts of the chain that are no longer needed are removed, while new parts of the chain are added.

### *5.3. Service chain network adaptation*

This operation regards the adaptation capability of the orchestration system with respect to the network status in the SDN WAN. The service chain network adaptation is performed using an OpenFlow-based load balancing algorithm. As input, this algorithm periodically receives monitoring traffic statistics of the OpenFlow switches in the WAN collected by the SDN controller. The WIM then comes into play by adapting the network paths connecting edge cloud domains and underpinning the VNF chain path segments with respect to the load status information of switches/links.

With respect to static load balancing where flows are allocated with calculated routes before data transmission, dynamic load balancing offers a more flexible way to handle network devices load using updated traffic statistics. In this work, as shown in Fig. 5, we consider a WAN topology

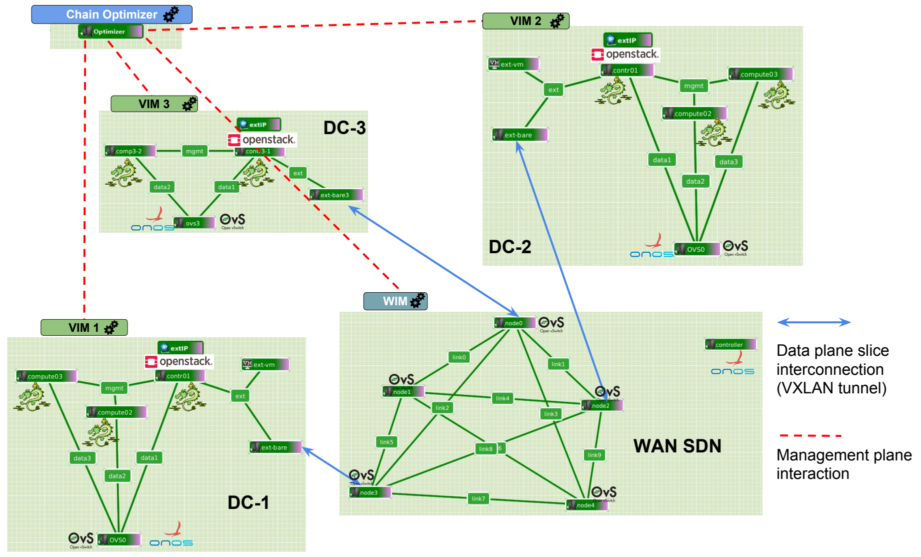

Chain Optimizer
Optimizer
VIM 3
DC-3
VIM 2
DC-2
VIM 1
DC-1
WIM
WAN SDN
Data plane slice interconnection (VXLAN tunnel)
Management plane interaction

Figure 5: Experiment Deployment on the Virtual Wall testbed from the Fed4FIRE experimentation platform

composed of redundant links providing alternative paths for data transmission. The WIM is then responsible for steering the traffic from one data center to the other over the OpenFlow switches by applying the load balancing algorithm. In particular, the status of the network is continuously monitored through the collection of real-time statistics to derive switches load status. A threshold is also fixed as an upper bound for the switches load. If a switch (or more switches) load exceeds this value, a redirection mechanism is automatically triggered according to the following steps. First, the service chains paths traversing the overloaded switch are identified. Then with the help of the SDN controller, the WIM exploits the Dijkstra shortest path algorithm to calculate new routes for every identified service chain involving only the subset of switches still available (i.e., not overloaded) in the WAN. After that, new flow entries are set-up in the flow tables of the available switches thus allowing to steer the traffic again across the WAN, while all the flow rules belonging to the old paths are deleted from the overloaded switches. In this way, our algorithm re-balances the load and helps in eliminating the congestion in the network.

During the experiment we show an example in which, after the deployment of a service chain request, a subset of the switches in the WAN SDN domain becomes overloaded, which triggers the dynamic adaptation capability, thus redirecting the traffic through other available switches.

In Fig. 4 we illustrate an example of interworking between the CO and the VIMs/WIM. More specifically, a service chain request is shown requiring the setup of a chain between "Node-B" and "Node-E" composed of $5$ VNFs. The CO handles this request by first selecting the instances of the VNFs available in a distributed multi-DC environment (e.g. VNF-1a instance in DC-1 and VNF-6d instance in DC-3) and then sends appropriate forwarding instructions (see bottom of Fig. 4) to concerned VIMs and WIM so that the flow is actually steered through these instances.

## 6. Integration of Orchestration Subsystems and Experiment set-up

Fig. 5 shows the SDN/NFV deployment we setup on top of the Fed4FIRE+ platform to reproduce an integrated SDN- and Openstack-based NFVI and to perform experiments with the proposed service chaining orchestration system. The experimental setup was deployed on the Virtual Wall testbed [62] through the establishment of five experiment containers, i.e., slices of resources from the testbed facility involving as many as 28 virtual machines. More specifically, we dedicated a slice to put in operation the CO, a slice to reproduce the WAN domain, and three slices to reproduce three SDN-based DC domains (named DC-1, DC-2 and DC-3). The slices are briefly described hereafter.

The ***Chain Optimizer experimental slice*** includes a physical node running Ubuntu Server 16.04 LTS. The Chain Optimizer exposes a REST API for receiving service chaining requests and handles the decision provided by the algorithm by processing and delivering appropriate service chaining intents to the WIM and VIM orchestrators through JSON messages over HTTP. The CO also gathers information about network and processing latency via REST API from the WIM and VIM orchestrators, respectively. In particular, processing latency measurements are posted by VNF instances onto a Time Series Database service (i.e., Gnocchi) provided by OpenStack on each DC slice and exposed through Gnocchi REST APIs. Similarly, network latency is gathered through the REST API of the WIM, which leverages on OpenFlow statistics provided by the SDN controller.

The ***SDN WAN experimental slice*** consists of three main components: the SDN network, the WAN SDN controller and the WIM orchestrator. All these components have been deployed within a unique experiment slice where one physical node has been allocated for the SDN controller and the WIM orchestrator, and 5 other physical nodes were allocated for the SDN topology. All the nodes run a Ubuntu 16.04 distribution. Specifically, we have installed the Open vSwitch (OvS) software on the physical nodes composing the topology in order to emulate OpenFlow switches. The OvS instances are controlled by an instance of the ONOS SDN controller running on the 6th physical node of the slice. The ONOS version used is Loon (1.11.0).

The three ***SDN DC experimental slices*** host cloud instances based on small OpenStack clusters (Pike version) running on a Linux Ubuntu 16.04

operating system [63]. Each DC slice includes two or three OpenStack compute nodes, where virtual machine instances are deployed over a QEMU-KVM hypervisor. One of the compute nodes also acts as controller and network node, respectively providing service REST API endpoints and external connectivity. All OpenStack nodes are connected to another physical node running an instance of OvS, representing the data plane SDN infrastructure of the DC, which is controlled by an instance of ONOS (Ibis version, 1.8.3) running locally. The same physical node hosts also the VIM orchestrator. A separate network is used for the OpenStack management plane. Finally, another physical node is used as the egress router of the DC slice and is connected to the SDN WAN slice. The OpenStack cluster running in each slice exposes the essential cloud services and related APIs, including compute and placement (Nova), identity (Keystone), image (Glance), and network (Neutron), as well as the time series database service (Gnocchi). On top of those cloud services, the service chains requested by the Chain Optimizer are deployed by means of the OpenStack SFC extension, an additional component providing API and mechanisms to support service function paths creation and deployment in Neutron [64]. Traffic steering across the service chain elements is implemented by means of specific OpenFlow rules installed in the virtual switches through a Ryu SDN controller internally deployed by Neutron. The problem of determining the current position of a given packet within a chain can be solved by taking advantage of different encapsulation techniques, including Multi-Protocol Label Switching (MPLS) and Network Service Header (NSH). Due to compatibility issues with the Linux kernel version adopted, only the former encapsulation is used in our setup.

We selected ONOS as the SDN controller of the WAN since it is characterized by a high modularity feature and a distributed architecture. Moreover, it has better performance in terms of jitter and packet loss with respect to OpenDayLight and Floodlight controllers [65][66]. OpenStack was chosen since it represents the most popular and mature open-source software platform for IaaS deployments, and offers the SFC extension mentioned in Section 7.2. It is also well integrated with other existing NFV platforms such as Open Source MANO.

The established DC slices and the WAN slice interact at the data plane level by exchanging packet data traffic, and at the orchestration plane level by exchanging control messages between the CO, WIM and VIM orchestrators. Each DC slice is connected to the WAN slice at the data plane level by means of VXLAN tunnels established on top of the Virtual Wall management network.

The VXLAN virtual tunnel endpoint (VTEP) located at the DC slice egress router appears as an IP-routable interface of the router itself, whereas the corresponding VTEP located at the WAN slice node is bridged to the OvS instance running in the same node. This particular setup allows the WAN slice to act as a layer-2 SDN infrastructure (orchestrated by the WIM) interconnecting the three DC slices. The Chain Optimizer slice exchanges messages with the other slices at the orchestration plane level via the Virtual Wall management network. Interactions at the network control plane level do not take place between different slices. This is in line with the envisioned architecture, where each domain is supposed to adopt its own SDN control plane solution independently of the choice made by other domains.

For the purpose of the experiments described in this paper, whose main focus is the dynamic establishment and adaptation of service chains, the full orchestration of computing and network resources and the lifecycle management of VNFs are not strictly needed. Indeed, VNF placement and deployment operations (possibly in appropriate DC locations based on specified constraints) are out of the scope of this work, which focuses on orchestrating VNFs instances so as to properly select VNFs and virtual links to establish and adapt service chains dynamically and concurrently. For this reason, the deployment set-up does not run an NFVO platform (e.g., Open Source MANO [67], OpenBaton [68]). In our current system implementation, and without lack of generality, the CO interacts directly with the VIMs/WIM northbound interfaces. In addition, a set of VNFs are instantiated at the experiment deployment time to be already running at the time the service chaining is considered. In this context, the specific function played by the VNF is irrelevant, e.g., either it runs a Firewall or NAT appliance. They are considered only for their capacity to generate and process traffic data. Thus, they have been basically instantiated as virtual machines running software modules specifically developed to carry out these experiments, or tools to generate traffic in the network (e.g., iperf [69]). The VNFs have been deployed in the DCs according to the predefined placement plan shown in Table 1. However, future evolutions of our orchestration system will require the integration of the proposed orchestration functionalities with relevant MANO components, i.e., NFVO and VNFM, to perform service chaining within the context of a Network Service deployment and lifecycle management.

Finally, the following additional software modules are used to support the experimental activity: (i) *CO-Client GUI* which is a web-based GUI that allows to generate service chain creation and deletion requests and deliver them to the CO. A service chain request contains an ordered sequence of types of VNFs that should process the traffic flow, the bandwidth demand, and some parameters identifying the traffic flow (e.g., source and destination nodes); (ii) *Workload-Processing Latency Publisher*: a custom Java application that emulates a processing latency profile proportional to the I/O workload. According to a configurable processing capacity value, this application computes a processing latency value as a function of the processing

Table 1: Placement of the VNF instances on the three edge Cloud DCs.

| | **DC-1** | **DC-2** | **DC-3** |
|---|---|---|---|
| **VNF Types** | VNF-1, VNF-2, VNF-3, VNF-4, VNF-5, VNF-6, VNF-7 | VNF-5, VNF-6, VNF-7, VNF-8, VNF-9, VNF-10 | VNF-5, VNF-6, VNF-7 |
| **VNF instances** | VNF-1a, VNF-1b, VNF-2a, VNF-2b, VNF-3a, VNF-3b, VNF-4a, VNF-4b, VNF-5a, VNF-5b, VNF-6a, VNF-6b, VNF-7a | VNF-5d, VNF-6e, VNF-7d, VNF-7e, VNF-8a, VNF-8b, VNF-9a, VNF-9b, VNF-10a, VNF-10b | VNF-5c, VNF-6c, VNF-6d, VNF-7b, VNF-7c |

capacity and the traffic input rate measured at the network interface. It also executes a script running in each VNF instance to emulate the effect of this calculated processing latency by applying this value as a delay to the output interface, using the Linux Kernel Traffic Control command [70]. The processing latency value is periodically posted as an ad-hoc defined metric maintained by the Gnocchi database in the OpenStack deployment for the benefit of CO computation.

## 7. Experimental Results

In this section, we provide a set of experimental results to evaluate the performance of the orchestration system to process service chain requests, to compute the latency-optimized VNF chains by correctly elaborating monitoring data on processing and network latency, and to set-up VNF chains across network and cloud domains by properly enforcing the interaction of the CO with the underlying VIM and WIM orchestrators.

To this purpose, we used the CO-Client GUI to deliver *create* and *delete* service chain requests to the CO with a different set of lengths and requirements. Accordingly, the CO handles the request, computes a latency-optimized solution and sends out the corresponding forwarding instructions to the affected VIM and WIM orchestrators. More specifically, a set of different service chain requests with their respective requirements (in terms of bandwidth and maximum latency) have been generated and sent one by one to the CO through the CO-Client GUI. An example of the generated chains sequences is described in Table. 2. As soon as the switches are configured and the chain is established in the edge cloud domains, the traffic (i.e., an iperf flow [69] with a bit rate equal to $1$ Mb/s) starts flowing across the network.

Table 2: Example of generated chains sequence.

| Sequence | Chain | Requirements |
|---|---|---|
| 1 | VNF1, VNF2, VNF8 | 1Mbps 500ms |
| 2 | VNF3, VNF6, VNF9, VNF10 | 1Mbps 500ms |
| 3 | VNF4, VNF7, VNF10 | 1Mbps 500ms |
| 4 | VNF4, VNF5, VNF9, VNF10 | 1Mbps |
| 5 | VNF1, VNF2, VNF7, VNF8 | 1Mbps |

## *7.1. Performance of latency-optimized service chain path computation*

We considered the overall response time which is the time measured at the CO side elapsing from the reception of a request to the delivery of a response to the client. The overall response time includes the execution of the VNF selection optimization algorithm by the CO and, if a solution is found, the time needed for sending the forwarding instructions to VIM and WIM orchestrators and receiving their reply. We also specifically measured the computation time needed by the CO to run the VNF Selection algorithm and solve the optimization problem. Table 3 reports the measured values as a function of the service chain length. As expected, the overall time required for handling a service chain request increases with the chain length since the WIM and VIM orchestrators require more time for the chain setup. Given the relatively limited scale of the experimental setup, the computation time of the optimization algorithm remains in the order of a few milliseconds (from 2 to 3.2 ms) and it is therefore not shown as a separate metric in the table. Most of the overall response time is due to the VIMs and WIM response times, which are analysed in detail in subsections 7.2 and 7.3, respectively.

Table 3: Time needed for service chain deployment

| Chain Length | Overall response Time [s] |
|---|---|
| 2 | 64.34 |
| 3 | 69.34 |
| 4 | 74.96 |
| 5 | 82.75 |

Hereafter, we present a set of tests conducted to evaluate the end-to-end latency estimation error and to perform a comparative evaluation with a greedy VNF selection strategy. The tests have been conducted by generating a set of 10 chain requests, with length 3, VNF types randomly picked up from the list of available ones and randomly selected source and destination nodes. Requests are sequentially submitted to the CO and corresponding UDP traffic flows from source to destination nodes are generated using the iperf tool. These tests have been conducted in three different scenarios: *i) no background traffic* - in this scenario the inter-DC traffic is given by the traffic flows of deployed chains; *ii) constant background traffic* - the latency between DCs is increased by a constant value (5 ms), using the traffic control command (tc) in the Linux kernel, after each chain request in order to emulate a continuous increment of background traffic in addition to traffic flows traversing deployed chains and *iii) step-increasing background traffic* - the inter-DC latency is increased by 50 ms after the third chain request and by 15 ms after the sixth chain request in order to emulate a step-wise increment of inter-DC background traffic.

Since the CO makes its decisions leveraging an estimated end-to-end latency, it is relevant evaluating the error introduced. The end-to-end latency is computed by the CO as the sum of the processing latency collected from VNFs and inter-DC latency measurements. Therefore, the error in latency estimation may be affected by inter-DC network delay variations that can be experienced in the infrastructure and by intra-DC network delays. Fig. 6 shows the relative error between measured and estimated end to end latency. In all scenarios the average relative error is below 3.9%. We consider such error acceptable.

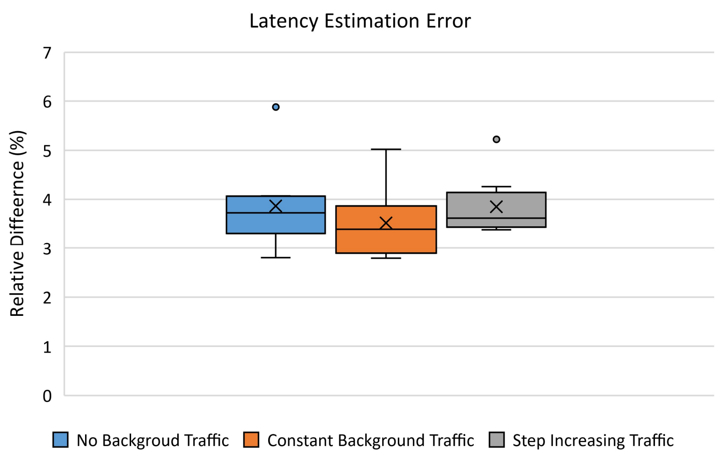


Figure 6: Estimation error in end-to-end latency computed by the Chain Optimizer

We also compared the performance of the CO with respect to a greedy strategy, which selects VNF instances with lowest processing time and does not account for inter-DC network latency. Fig. 7 shows the relative difference in measured end-to-end latency of deployed chains obtained by the VNF Selection algorithm of the CO vs. the greedy strategy. The CO outperforms the Greedy strategy on average and the relative difference increases with the background traffic, since this also affects the inter-DC latency. As shown in the figure, in some cases Greedy outperforms the CO. This is expected, since the experiments start with the same configuration but at each

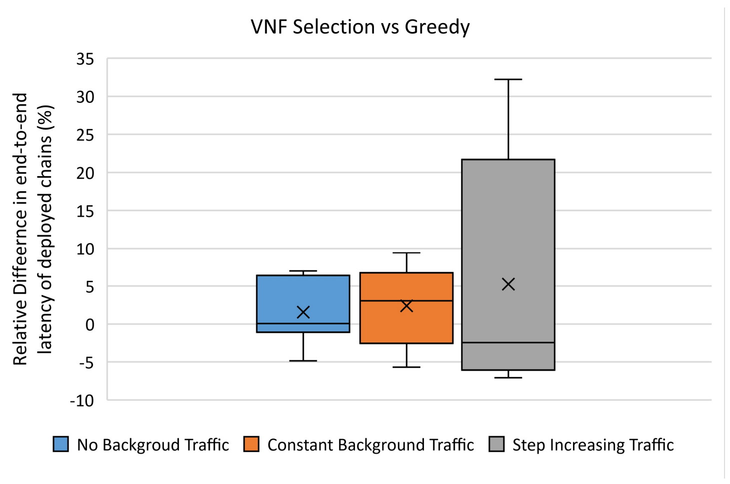


Figure 7: Distribution of relative difference in measured end-to-end latency of deployed chains between the VNF Selection algorithm at the Chain Optimizer and a greedy strategy

step the two strategies may take different decisions and therefore the infrastructure status may evolve differently.

## *7.2. Performance of service chain path deployment in the cloud*

The response time of the VIM for deploying service chains measured in one of the DC slices as a function of the chain size is shown in Fig. 8. Such response time is measured from the instant the VIM receives the forwarding instructions from the CO, to the instant the VIM replies that the chain has been correctly deployed, thus including the time required to locate the needed VNF instances (already running on the servers) and apply all the necessary traffic steering rules in the OpenStack network components. The linear growth of the VIM response time with the number of VNFs included in the service chain clearly shows the significant impact of the size of the requested chain. However, even for a service chain composed of as many as 10 elements in a single DC, the overall deployment stays below ten seconds. Considering that chain deployment can run in parallel in different DCs and that we hardly expect a service chain to include more than a few units of VNFs, we can conclude that the VIM operations are sufficiently scalable.

In order to better understand the behavior of the VIM response time and its relation with the service chain size, we decided to analyze in details the inner mechanisms of the OpenStack SFC extension, which deploys the actual service chain path in four steps [71]:

1. A *Flow Classifier* is instantiated, which classifies the incoming traffic based on header matching rules and selects the packets that must be forwarded through a given service chain.

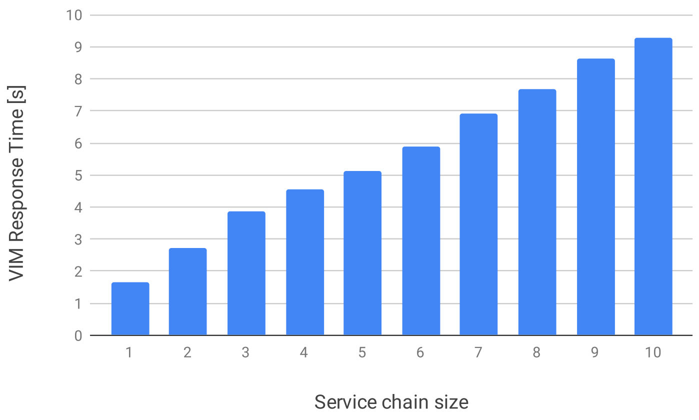


Figure 8: Response time at the VIM for service chain deployment as a function of the chain size, for the case of a single instance per VNF type.

2. A number of *Port Pairs* are then created, each specifying the ingress and egress ports of a potential element of the chain. Note that each Port Pair refers to an actual instance of a VNF, and it can be assigned a weight to be used for load balancing among multiple instances of the same VNF type. A Port Pair can be either uni- or bi-directional, depending on how the corresponding VNFs should be traversed by the packet flow.

3. As many *Port Pair Groups* are created as the number of elements in the chain. A Port Pair Group includes the Port Pairs corresponding to multiple instances of the same VNF type. Weighted load balancing is automatically applied to a given chain element when a given Port Pair Group includes more than one Port Pairs, based on the weight assigned to the latter.

4. Eventually, a *Port Chain* is created as a binding between one or more Flow Classifiers and an ordered list of Port Pair Groups, including all the required configurations in the data plane components (e.g., relevant flow matching and steering rules installed in virtual switches). The Port Chain then represents the deployment of the service chain path to be traversed by some given traffic flows, implementing load balancing if multiple instances of the VNFs are available. A Port Chain can also be asymmetrical or symmetrical, depending on whether the packets must traverse the chain only in the forward path or also in their way back to the source.

Referring to the first example of chain reported in Table 2 and consid-

Table 4: Breakdown of the VIM response time for service chain deployment in the cloud, for increasing chain size and a single instance per VNF type.

| **Service Chain Size** | **Flow Classifier [s]** | **Port Pair [s]** | **Port Pair Group [s]** | **Port Chain [s]** | **VIM Resp. Time (+/- StDev) [s]** |
|---|---|---|---|---|---|
| 1 | 0.1602 | 0.5225 | 0.0686 | 0.8524 | 1.6702 (+/- 0.1233) |
| 2 | 0.1567 | 0.5958 | 0.5768 | 1.3422 | 2.7419 (+/- 0.2015) |
| 3 | 0.1568 | 0.7155 | 1.1189 | 1.7873 | 3.8478 (+/- 0.3837) |
| 4 | 0.1587 | 0.7397 | 1.2577 | 2.3155 | 4.5583 (+/- 0.5231) |
| 5 | 0.1631 | 0.7973 | 1.2990 | 2.7990 | 5.1419 (+/- 0.4076) |
| 6 | 0.1547 | 1.1460 | 1.2622 | 3.2401 | 5.8900 (+/- 0.1669) |
| 7 | 0.1575 | 1.4964 | 1.4205 | 3.7319 | 6.9096 (+/- 0.1192) |
| 8 | 0.1602 | 1.7106 | 1.4427 | 4.2506 | 7.6638 (+/- 0.1940) |
| 9 | 0.1530 | 2.0064 | 1.5720 | 4.7808 | 8.6235 (+/- 0.2164) |
| 10 | 0.1563 | 2.0971 | 1.6619 | 5.2477 | 9.2691 (+/- 0.7349) |

ering the VNF placement in DC-1 reported in Table 1, the OpenStack SFC extension running in DC-1 created two Port Pair Groups (corresponding to VNF-1 and VNF-2), four Port Pairs (corresponding to the four instances VNF-1a, VNF-1b, VNF-2a and VNF-2b), one Flow Classifier and One Port Chain.

We measured the contributions from the four steps mentioned above to the overall VIM response time, as reported in Table 4 for a chain size ranging from 1 to 10 VNFs and a single instance per VNF type. The number of chain elements does not affect the time needed to create the Flow Classifier, as in this case we required one flow only to traverse the chain. The chain size does have an impact on the other contributions, although this is relatively limited when creating Port Pairs and Port Pair Groups, because those steps require only to update an internal database. What brings the most significant contribution to the VIM response time is the time to create the Port Chain: this is the step when all the OpenStack networking configurations are finally applied, including specific OpenFlow rule installation in the virtual switches. The results quantify how the higher the number of chain elements, the longer the time needed to perform all configurations.

The number of instances per VNF type is another parameter that affects the time needed to create a Port Chain and, as a consequence, the overall VIM response time, as shown in Fig. 9 for the case of a chain size of 4 VNFs, each with a number of instances ranging from 1 to 5. Finally, the impact of defining multiple flows (ranging from 1 to 10) on the time to create the Flow Classifier is shown in Fig. 10, for a chain size of 4 VNFs and 3 instances per VNF type. The contribution of this step to the response time does not depend on the chain size nor on the number of instances per VNF type, and it is limited to sub-second scale.

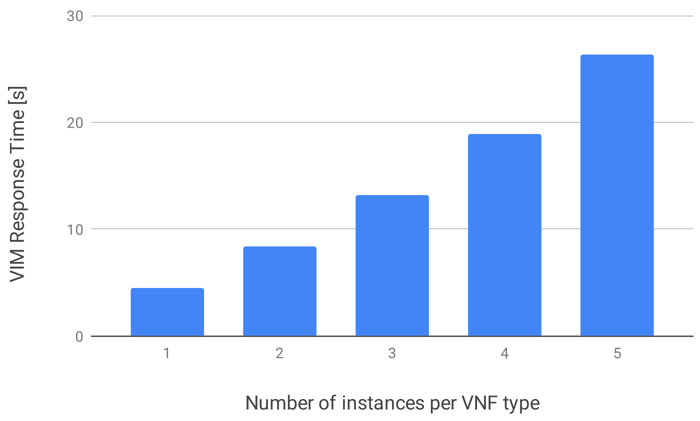


Figure 9: Response time at the VIM for service chain deployment as a function of the number of VNF instances per VNF type, for the case of a chain size of 4 VNFs.

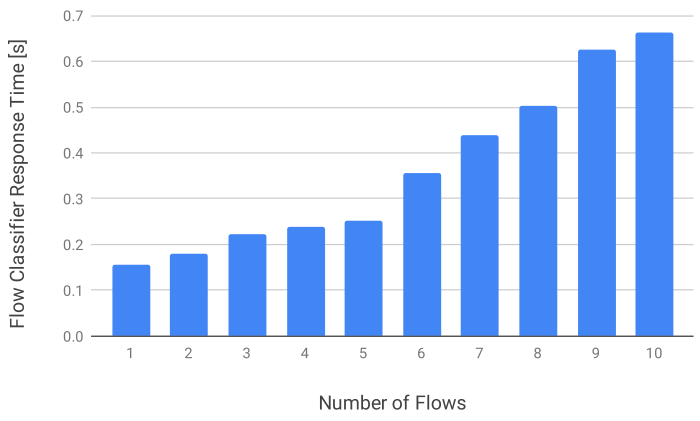


Figure 10: Response time of the Flow Classifier deployment as a function of the number of flows that must traverse the service chain being deployed, for the case of a chain size of 4 VNFs and 3 instances per VNF type.

## *7.3. Performance of service chain path deployment and redirection across WAN*

In this paragraph, we present the performance related to the deployment of the service chain paths across the WAN domain. More specifically, we report the overall time required by the WIM to handle a service chain setup request, the time required by the WAN SDN controller to install the flow rules for a service chain path, and the time required to perform redirection in case of switches overload.

Fig. 11 plots the measured overall response time for the 5 chain requests detailed in Table 2. The response time corresponds to the time elapsing between the reception of a request by the WIM and the acknowledgement sent back to the CO. It includes the identification of the switches composing the service chains paths, the check of their status evaluating their current load and the time necessary for the setup of the specific flow entries in the switches flow tables. The error bars indicate the standard deviation from the measured mean over 10 identically performed measurements for each chain sequence. The experiments gave a mean of 44.38s with a standard deviation equal to 4.23s. Results show that the WIM response's time not only depends on the length of the deployed service chain but also on the number of selected DCs where the VNFs are deployed. This is reflected in the plot where Chain 1 and 3 present a lower time with respect to the other chains since their length is equal to 3. Among those 2 chains, Chain 1 requires less time since it is deployed in only 2 DCs (DC-1 and DC-2) while the VNFs composing Chain 3 are deployed in the 3 DCs.

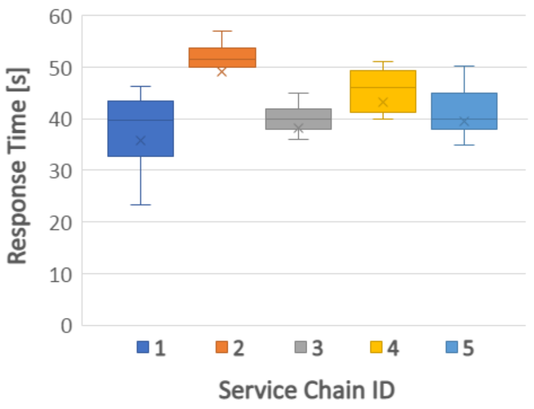


Figure 11: Response time of path service chains.

In Fig. 12, we detail the setup time considering the same 5 paths for the creation of different sequences of service chains. The setup time represents the time required by the SDN controller for the configuration of the flow entries in all the switches composing the path interconnecting the VNFs in the chain. As shown by the results, this time also mainly depends on the service chains length and has an average of $17s$ which is reasonable in our opinion. It is worth noting that the time required to setup a path for a chain spread across $2$ DCs is around $15$s while it increases up $20$s for chains spreading across $3$ DCs.

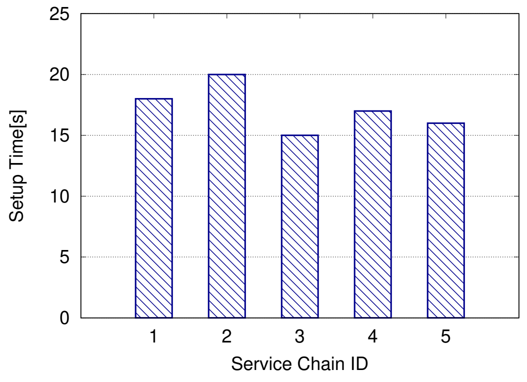


Figure 12: Setup time of path service chains.

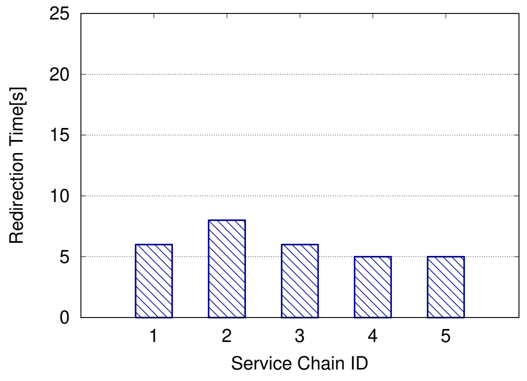


Figure 13: Redirection time of path service chains.

In Fig. 13, we show the redirection time, which represents the time necessary for the load-balancing algorithm to adapt the network paths connecting the edge cloud domains to the load status information of switches. In case one or more switches are overloaded, the WIM redirects the traffic through other available switches by first deleting the old configuration rules and then setting new ones in the selected switches. Also in this case, the redirection time depends on the length of the service chains (i.e., number of flow rules to be deleted and then reconfigured) and is relatively low (i.e.

around an average of 6s) with respect to the overall setup time.

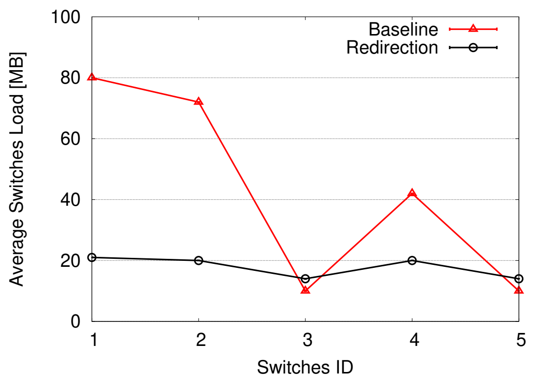


Figure 14: Impact of the load-balancing algorithm on the switches load.

Finally, to show the efficiency of the load-balancing algorithm, we evaluated the performance of the SDN network in a larger emulated environment. More specifically, we used the Abilene topology [72], which is composed of 11 nodes connected in a mesh network. In each node of the topology, we deployed an OpenFlow-enabled emulated switch using the Mininet tool [73]. A subset of those switches (i.e., 5 out of 11) were connected to emulated cloud platforms that reproduce the behavior of the data centers. Fig. 14 plots the average load of the switches connected to the cloud platforms when the load-balancing algorithm is applied with respect to the baseline. We can clearly notice that without the redirection mechanism there are high disparities among the switches in terms of average load where, for example, switches 1 and 2 are almost doubly loaded with respect to switches 3 and 5. On the contrary, the application of the load-balancing algorithm introduces an improvement in the results demonstrated by an average load almost equally distributed among the available switches.

## 8. Conclusions and Discussion

In this work, we presented the result of an integration effort to build LASH-5G, an end-to-end orchestration system comprising optimized VNF selection and dynamic traffic steering control capabilities supporting latency-aware, adaptive and reliable service chaining over geographically distributed SDN-based cloud DCs interconnected through SDN WAN. We carried out experiments to validate and evaluate the performance of the orchestration system using the Fed4FIRE platform. This federated testbed facility allowed us to set-up a composite yet integrated SDN/NFV deployment thereby reproducing a distributed yet realistic 5G infrastructure set-up.

First, we were able to deploy the OpenStack cloud platform and different SDN controllers (i.e., ONOS, Ryu) through a substantial multi-domain SDN/NFV deployment (i.e., 28 virtual machines spread across 3 cloud domains interconnected through 5-nodes WAN).

Definitely, this composite cloud deployment allowed us to develop and fine-tune the VIM software components, especially in terms of handling underlying heterogeneous network controllers. In addition, we could finely tune the VIM operation to handle all possible cases of needed configurations while enforcing dynamic service chaining rules when multiple cloud domains are involved (e.g., correctly deploying and updating service chains while handling different combinations of VNF instances and service endpoints located in different OpenStack nodes and clusters).

Second, we were able to measure communication latency in an integrated network environment while testing the VNF selection algorithm included in the CO implementation on top of a dynamic view of underlying virtual resources and network topology. According to the performance metrics measured with the experiments we observed that the computation time of the algorithm is very low with respect to the time elapsed for forwarding instructions delivery and chain installation. This is also due to the fact that the algorithm works on an abstract topology that hinders intra-DC topology details.

Finally, we used virtual testbeds and their federation features to carry out experiments using scales that are generally larger than experiments carried out in a university laboratory. With virtual testbeds, we were able to start performing significant tests of the system prototype thereby achieving evaluation process effectiveness and cost savings. As a future work, we plan to: i) evaluate the orchestration system operation in a more extensive way, ii) extend the comparative evaluation with alternative VF selection strategies and iii) adopt a container-based setup and Kubernetes orchestration capabilities.

## References

[1] B. Bloching, et al., The digital transformation of industry - how important is it? who are the winners? what must be done?, Roland Berger Strategy Consultants GmbH (2015).

[2] S. Zhang, et al., 5G: Towards energy-efficient, low-latency and high-reliable communications networks, in: IEEE International Conference on Communication Systems, 2014, pp. 197–201. doi:10.1109/ICCS.2014.7024793.

[3] H. Ji, et al., Introduction to ultra reliable and low latency communications in 5G, Computing Research Repository (CoRR) abs/1704.05565 (2017).

[4] B. A. A. Nunes, et al., A survey of software-defined networking: Past, present, and future of programmable networks, IEEE Communications Surveys & Tutorials 16 (2014) 1617–1634.

[5] B. Han, et al., Network function virtualization: Challenges and opportunities for innovations, IEEE Communications Magazine 53 (2015) 90–97.

[6] J. Soares, et al., Toward a telco cloud environment for service functions, IEEE Communications Magazine 53 (2015).

[7] Q. Duan, et al., A survey on service-oriented network virtualization toward convergence of networking and cloud computing, IEEE Transactions on Network and Service Management 9 (2012).

[8] Sánchez, et al., Softwarized 5G networks resiliency with self-healing, in: IEEE 1st International Conference on 5G for Ubiquitous Connectivity (5GU), 2014, pp. 229–233.

[9] A. Manzalini, et al., An edge operating system enabling anything-as-a-service, IEEE Communications Magazine 54 (2016) 62–67.

[10] F. Paganelli, et al., Context-aware service composition and delivery in NGSONs over SDN, IEEE Communications Magazine 52 (2014) 97–105.

[11] W. Cerroni, M. Gharbaoui, B. Martini, A. Campi, P. Castoldi, F. Callegati, Cross-layer resource orchestration for cloud service delivery: A seamless SDN approach, Computer Networks 87 (2015) 16–32.

[12] I. Parvez, et al., A survey on low latency towards 5G: RAN, core network and caching solutions, IEEE Communications Surveys & Tutorials 20 (2018) 3098–3130.

[13] S. Fichera, R. Martínez, B. Martini, M. Gharbaoui, R. Casellas, R. Vilalta, R. M. noz, P. Castoldi, Latency-aware resource orchestration in SDN-based packet over optical flexi-grid transport networks, Journal of Optical Communications and Networking 11 (2019) B83–B96.

[14] T. Wauters, et al., Federation of internet experimentation facilities: architecture and implementation, in: European Conference on Networks and Communications (EuCNC), 2014.

[15] B. Martini, et al., Latency-aware composition of virtual functions in 5G, in: Proceedings of 1st IEEE Conference on Network Softwarization (NetSoft), 2015, pp. 1–6. doi:10.1109/NETSOFT.2015.7116188.

[16] A. Mohammed, et al., SDN controller for network-aware adaptive orchestration in dynamic service chaining, in: IEEE NetSoft Conference and Workshops (NetSoft), 2016, pp. 126–130.

[17] F. Callegati, et al., Performance of intent-based virtualized network infrastructure management, in: Proceedings of IEEE International Conference on Communications (ICC), 2017, pp. 1–6.

[18] B. Martini, M. Gharbaoui, D. Adami, P. Castoldi, S. Giordano, Experimenting SDN and cloud orchestration in virtualized testing facilities: Performance results and comparison, IEEE Transactions on Network and Service Management 16 (2019) 965–979.

[19] M. Gharbaoui, et al., Experimenting latency-aware and reliable service chaining in next generation internet testbed facility, in: IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2018, pp. 1–4. doi:10.1109/NFV-SDN.2018.8725783.

[20] M. Gharbaoui, et al., Demonstration of latency-aware and self-adaptive service chaining in 5G/SDN/NFV infrastructures, in: IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2018, pp. 1–2. doi:10.1109/NFV-SDN.2018.8725645.

[21] A. M. Medhat, et al., Service function chaining in next generation networks: State of the art and research challenges, IEEE Communications Magazine 55 (2017) 216–223.

[22] B. Martini, et al., SDN controller for context-aware data delivery in dynamic service chaining, in: IEEE 1st Conference on Network Softwarization (NetSoft), 2015, pp. 1–5.

[23] Y. Zhang, et al., Steering: A software-defined networking for inline service chaining, in: 21st IEEE International Conference on Network Protocols (ICNP), 2013, pp. 1–10.

[24] Z. Qazi, et al., Practical and incremental convergence between SDN and middleboxes, in: Open Network Summit, Santa Clara, CA, 2013.

[25] A. M. Medhat, et al., Extensible framework for elastic orchestration of service function chains in 5G networks, in: IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2017, pp. 327–333. doi:10.1109/NFV-SDN.2017.8169879.

[26] F. Schardong, I. Nunes, A. Schaeffer-Filho, NFV resource allocation: a systematic review and taxonomy of VNF forwarding graph embedding, Computer Networks 185 (2021) 107726.

[27] A. M. Medhat, et al., Near optimal service function path instantiation in a multi-datacenter environment, in: 11th International Conference on Network and Service Management (CNSM), 2015, pp. 336–341. doi:10.1109/CNSM.2015.7367379.

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

[29] A. Lombardo, et al., An analytical tool for performance evaluation of software defined networking services, in: IEEE Network Operations and Management Symposium (NOMS), 2014, pp. 1–7.

[30] M. T. Thai, et al., A joint network and server load balancing algorithm for chaining virtualized network functions, in: IEEE International Conference on Communications (ICC), 2016, pp. 1–6. doi:10.1109/ICC.2016.7510712.

[31] Y.-W. Ma, et al., Adaptive service function selection for network function virtualization networking, Future Generation Computer Systems 91 (2019) 108 – 123.

[32] T.-M. Nguyen, et al., Routing via functions in virtual networks: The curse of choices, IEEE/ACM Transactions on Networking 27 (2019) 1192–1205.

[33] J. Pei, et al., Two-phase virtual network function selection and chaining algorithm based on deep learning in SDN/NFV-enabled networks, IEEE Journal on Selected Areas in Communications 38 (2020) 1102–1117.

[34] P. Kathiravelu, P. Van Roy, L. Veiga, Composing network service chains at the edge: A resilient and adaptive software-defined approach, Transactions on Emerging Telecommunications Technologies 29 (2018).

[35] R. Chaudhary, N. Kumar, S. Zeadally, Network service chaining in fog and cloud computing for the 5g environment: Data management and security challenges, IEEE Communications Magazine 55 (2017) 114–122.

[36] Y. Kim, J. Kwak, H.-W. Lee, S. Chong, Dynamic computation and network chaining in integrated sdn/nfv cloud infrastructure, IEEE Transactions on Cloud Computing (2021).

[37] D. Harutyunyan, et al., Latency-aware service function chain placement in 5G mobile networks, in: IEEE Conference on Network Softwarization (NetSoft), 2019, pp. 133–141.

[38] J. Santos, et al., Towards delay-aware container-based service function chaining in fog computing, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2020, pp. 1–9.

[39] R. Cziva, et al., Dynamic, latency-optimal VNF placement at the network edge, in: IEEE Conference on Computer Communications (INFOCOM), 2018, pp. 693–701.

[40] A. Gavras, et al., Future internet research and experimentation: The fire initiative, Comput. Commun. Rev. 37 (2007) 89–92.

[41] M. Berman, et al., Geni: A federated testbed for innovative network experiments, Computer Networks 61 (2014) 5–23.

[42] M. Gharbaoui, et al., Experiments on SDN-based network and cloud resource orchestration in Fed4FIRE, in: IEEE NetSoft Conference and Workshops (Netsoft), 2016, pp. 131–135.

[43] B. Martini, et al., Experimenting SDN and cloud orchestration in virtualized testing facilities: performance results and comparison, IEEE Transactions on Network and Service Management 16 (2019) 965–979.

[44] S.Vural, et al., Performance measurements of network service deployment on a federated and orchestrated virtualisation platform for 5G experimentation, in: IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2018, pp. 1–6.

[45] F. Silva, et al., 5ginfire: Enabling an NFV based experimentation of vertical industries in the 5G context, in: Proc. Anais do X Workshop de Pesquisa Experim. da Internet do Futuro (WPEIF), 2019, pp. 64–69.

[46] OpenBaton, https://openbaton.github.io/, 2021.

[47] G. Xilouris, et al., T-NOVA: A marketplace for virtualized network functions, in: European Conference on Networks and Communications (EuCNC), 2014, pp. 1–5. doi:`10.1109/EuCNC.2014.6882687`.

[48] B. Sonkoly, et al., Unifying cloud and carrier network resources: An architectural view, in: Proceedings of the IEEE Global Communications Conference (GLOBECOM), 2015, pp. 1–7.

[49] A. Sgambelluri, et al., Orchestration of network services across multiple operators: The 5G exchange prototype, in: European Conference on Networks and Communications (EuCNC), 2017.

[50] J. Baranda, et al., NFV service federation: enabling multi-provider ehealth emergency services, in: IEEE International Conference on Computer Communications (INFOCOM), 2020.

[51] R. Bruschi, et al., Validation of iaas-based technologies for 5g-ready applications deployment, in: European Conference on Networks and Communications (EuCNC), 2020, pp. 46–51.

[52] M. Dieye, et al., CPVNF: Cost-efficient proactive VNF placement and chaining for value-added services in content delivery networks, IEEE Transactions on Network and Service Management 15 (2018) 774–786.

[53] A. C. Baktir, A. Ozgovde, C. Ersoy, How can edge computing benefit from software-defined networking: A survey, use cases, and future directions, IEEE Communications Surveys & Tutorials 19 (2017) 2359–2391.

[54] G. N. ETSI, Network functions virtualisation (NFV): Architectural framework, ETSI GS NFV 2 (2013) V1.

[55] 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.

[56] G. N.-M. . ETSI, Network functions virtualisation (nfv); management and orchestration, ETSI GS NFV V1.1.1 (2014).

[57] G. N.-I. . ETSI, Network functions virtualisation (nfv) release 3; management and orchestration; interface and information model specification for multi-site connectivity services, ETSI GS NFV 3.4.1 (2020).

[58] A. Clemm, et al., Intent-Based Networking - Concepts and Definitions, Internet-Draft, 2021. URL: `https://tools.ietf.org/pdf/draft-irtf-nmrg-ibn-concepts-definitions-05.pdf`.

[59] Y. Boucadair, et al., Service Function Chaining Service, Subscriber and Host Identification Use Cases and Metadata, Technical Report, IETF Secretariat, 2017. URL: `https://tools.ietf.org/html/draft-sarikaya-sfc-hostid-serviceheader-04`.

[60] B. Martini, et al., A service-oriented approach for dynamic chaining of virtual network functions over multi-provider software-defined networks, Future Internet 8 (2016) 24.

[61] F. Callegati, et al., SDN for dynamic NFV deployment, IEEE Communications Magazine 54 (2016) 89–95.

[62] Virtual Wall Web Site, https://doc.ilabt.imec.be/ilabt/virtualwall/, 2021.

[63] K. Pepple, Deploying openstack, 2nd Edition, O'Reilly Media (2013).

[64] Service Function Chaining Extension for OpenStack Networking, https://docs.openstack.org/networking-sfc/latest/, 2021.

[65] L. Zhu and others, SDN controllers: Benchmarking & performance evaluation, Available: http://arxiv.org/abs/1902.04491, 2019.

[66] L. Mamushiane, T. Shozi, A QoS-based evaluation of SDN controllers: ONOS and opendaylight, in: IST-Africa Conference (IST-Africa), 2021, pp. 1–10.

[67] M.-I. Csoma, et al., Management and orchestration for network function virtualization: An open source MANO approach, in: 19th RoEduNet Conference: Networking in Education and Research (RoEduNet), 2020, pp. 1–6.

[68] G. Carella, et al., Open baton: a framework for virtual network function management and orchestration for emerging software-based 5G networks, in: Newsletter, 2016.

[69] Iperf, https://iperf.fr/, 2021.

[70] https://wiki.linuxfoundation.org/networking/netem, 2021.

[71] D. Borsatti, et al., Performance of service function chaining on the OpenStack cloud platform, in: 1st Workshop on Segment Routing and Service Function Chaining (SR+SFC), 14th International Conference on Network and Service Management (CNSM), 2018, pp. 432–437.

[72] Abilene topology, [online] https://web.archive.org/web/20080113120821 /http://abilene.internet2.edu/, 2021.

[73] K. Karamjeet, et al., Mininet as software defined networking testing platform, in: International Conference on Communication, Computing & Systems (ICCCS), 2014.