A Holistic Approach to Define Service Chains Using Click-on-OSv on Different NFV Platforms

Alexandre Huff*†, Giovanni Venâncio†, Leonardo da C. Marcuzzo‡,
Vinícius F. Garcia‡, Carlos R. P. dos Santos‡ and Elias P. Duarte Jr.†
*Federal Technological University of Paraná, UTFPR, Toledo, Brazil
Email: alexandrehuff@utfpr.edu.br
†Department of Informatics
Federal University of Paraná, UFPR, Curitiba, Brazil
Email: {ahuff, gvsouza, elias}@inf.ufpr.br
‡Federal University of Santa Maria, UFSM, Santa Maria, Brazil
Email: {lmarcuzzo, vfulber, csantos}@inf.ufsm.br

***Abstract*—The deployment of services in virtualized networks can be done by composing multiple Virtualized Network Functions (VNFs). A Service Function Chain (SFC) consists of a predefined sequence of VNFs which are virtually connected and through which traffic is processed. This work proposes a framework for composing and managing the lifecycle of SFCs formed by VNFs built with Click-on-OSv. The proposal allows the execution of SFCs on different NFV orchestrators. We call the proposed approach "holistic" as it defines a generic API for the composition of SFCs that leverages particular details of different NFV orchestrators. A prototype of the framework was implemented to allow the composition and lifecycle management of SFCs formed by VNFs built with Click-on-OSv on the OpenStack Tacker NFV orchestrator. Results show that the framework is scalable and efficient.**

I. Introduction

Network Function Virtualization (NFV) enables the implementation in software of diverse network services which are traditionally provided as *middleboxes*. Virtual functions can then be executed on Commercial-Off-The-Shelf (COTS) hardware. Middleboxes are thus replaced by Virtualized Network Functions (VNFs) which can be managed by an NFV orchestrator [1], [2]. The European Telecommunications Standards Institute (ETSI) has proposed a standard architecture for NFV Management & Orchestration called as NFV-MANO [3] which provides specifications of multiple tasks related to VNF lifecycle management.

A VNF instance is responsible to process traffic and can act on several layers of the protocol stack. Routing traffic through a sequence of the VNFs according to a predefined order is known as Service Function Chaining (SFC), or simply "service chain" [4]. Usually a flow identifier is employed to route traffic to the next VNF in a SFC. Note that this is different from conventional routing, where decisions are taken based on the destination IP address.

The deployment of static service chains, which usually consist of multiple network functions composed in a certain way, can be complex, expensive and time-consuming [5]. The network operator has to execute a large number of tasks and services to accomplish service chain composition. This can be a laborious procedure that requires, in addition to experience, knowledge on specific modeling languages and their requirements for describing the service chain composition. In this way, network operators can get overwhelmed with details of the NFV platform being used, instead of focusing on the service composition logic itself. Furthermore, it is also necessary to ensure efficient resource usage so that the NFV infrastructure can be cost-effective.

Although the composition of network services has received much attention in recent years, there are still many challenges [6]. This work presents a solution to one of those challenges: we propose a simple and effective framework for the composition and lifecycle management of SFCs. An architecture is proposed for the framework as an extensible solution which can be used with different NFV platforms and applications. We call the proposed approach "holistic" as it defines a generic API for the composition of SFCs that leverages particular details of different NFV orchestrators. The "lifecycle management" terminology [7] that we follow represents a set of functions in charge of instantiating, accessing and destroying SFCs and their required resources.

A prototype of the proposed framework was implemented, allowing the composition and lifecycle management of service chains formed by VNFs built with Click-on-OSv [8] on the OpenStack Tacker NFV orchestration platform [9]. The system provides a REST API that abstracts a myriad of details of the NFV orchestrator, providing generic operations to compose and manage the SFC lifecycle. An application was also implemented to run performance tests on the proposed framework. Results show that it was about 4.145 times faster to compose 128 SFCs concurrently instead of doing that sequentially.

The remainder of this paper is organized as follows. Section II introduces basic concepts of Network Function Virtualization and Service Function Chaining. In Section III the proposed architecture is presented. The NFV service chain enablers, the prototype implementation, and the evaluation are described in Section IV. Section V points to related work. Finally, the conclusion is presented in Section VI.

978-1-5386-4727-1/18/$31.00 ©2018 IEEE

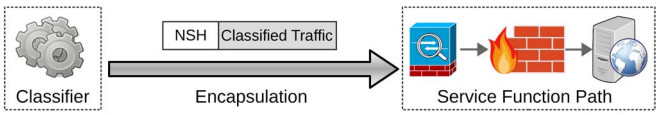

Fig. 1. The Classifier and SFP components are interconnected.

II. Service Function Chaining

Network Function Virtualization allows the implementation as software of several functionalities traditionally based on *middleboxes* [10]. The virtual functions can then be executed on general purpose hardware [1]. Software Defined Networking (SDN) technology both complement and make feasible the use of NFV by allowing the deployment of VNFs along with existing services and by providing the required network operation support. Furthermore, implementing network functions using NFV on software-defined networks makes it possible to manage each virtual network as a whole with orchestration mechanisms [11]. The NFV Orchestrator (NFVO) proposed by the ETSI NFV-MANO specification [3] employs NFV Infrastructure (NFVI) management and orchestration functions provided by the Virtualized Infrastructure Manager (VIM) to coordinate VNF composition to form network services.

In the context of NFV technology, instantiating a set of these network functions and, subsequently, steering the traffic orderly through a sequence of functions is known as Service Function Chaining. SFC allows the creation of network services consisting of multiple network functions in a given order [4]. In this paper we use the term "service chain" as a synonym to SFC. Several efforts have been recently made to standardize network service composition [1], [3], [6]. The IETF SFC Architecture [4] specifies that the decision of directing a given network flow to the next network function is made based on a flow identifier, instead of using destination IP address as is done in conventional routing.

The IETF SFC Architecture includes several logical components, such as Classifiers, Service Function Forwarders (SFFs), SFC Proxies, and the network functions themselves. The instantiation of a SFC is performed by selecting specific functions which are executed on certain network nodes, forming a service graph. A path of this graph is called Service Function Path (SFP). The Classifier deals with policies and restrictions associated with a SFP by intercepting and analyzing all traffic based on defined constraints for each instantiated network service. Network header information is configured so that the traffic flows through the appropriate SFPs [4].

Fig. 1 shows the Classifier adding a header to the intercepted traffic with the purpose of steering a flow to the corresponding Service Function Path. The SFP in this case consists of three network functions. A particular SFP is selected from classification policies associated with the flow identifiers, which can be those employed by MPLS (MultiProtocol Label Switching), Generic Routing Encapsulation (GRE), or Virtual eXtensible Local Area Network (VXLAN) [12].

The IETF SFC WG [6] has been making efforts to standardize the traffic encapsulation and information exchange between the participants of an SFP by defining the Network Service Header (NSH). The NSH contains metadata that indicates to which SFC a particular flow belongs to [13]. In this sense, SFC nodes need to be aware of the NSH since header information must be correctly parsed and interpreted. On the other hand, there are network elements unable to interpret the NSH and they must employ a SFC Proxy that does the job [5].

The SFC Proxy enables the usage of legacy network functions that are unaware of the SFC, and consequently of the NSH. These legacy functions can then be used in a service chain. The SFC Proxy encapsulates/decapsulates packets with the proper headers from both legacy network functions and SFF components, acting as a gateway between the SFC encapsulation and the legacy network functions. The SFF is responsible for forwarding flows received from the network to one or more SFCs and is also responsible for returning the processed traffic received from the SFC back to the network. SFFs maintain forwarding information that allows identifying paths between the SFPs [4].

Reclassification operations can be executed on the traffic that traverses a SFC and indicate a change of the original SFC path. The node where reclassification occurs is usually called "Branching node", suggesting the possibility to follow more than a single path (SFP) from a given VNF [4]. Moreover, a SFC can have symmetrical or asymmetrical paths. A symmetrical path indicates that the return flow must follow exactly the reverse path of the sending flow (*i.e.*, VNFs); asymmetrical paths do not pose any restriction between sending and returning flows [12].

Defining and redefining SFCs are tasks that are still hard to accomplish. The efficient composition of network services lacks high-level procedures and easy understanding. Tools that are easy to use and understand and that build efficient SFCs are still not available. This is still an open and relevant problem, and effective and simple SFC tools are essential to encourage and popularize the NFV paradigm.

III. The Proposed Framework

This work proposes a framework for the definition and lifecycle management of VNF service chains. The main goal is to provide a framework that is simple enough so that users (*e.g.*, network operators) do not have to execute a large number of configuration steps and grasp details of the multiple subsystems that form the NFV infrastructure to do service chain composition. Another important requirement is that the framework is aligned with the ETSI NFV-MANO specification, thus allowing its use on different NFV platforms that are MANO-compliant. We call the proposed approach "Holistic-Composer" as it defines a generic API for the composition and lifecycle management of SFCs that leverages particular details of different NFV orchestrators. The ultimate goal of the proposed framework is to allow the composition and lifecycle management of service chains in the simplest possible way,

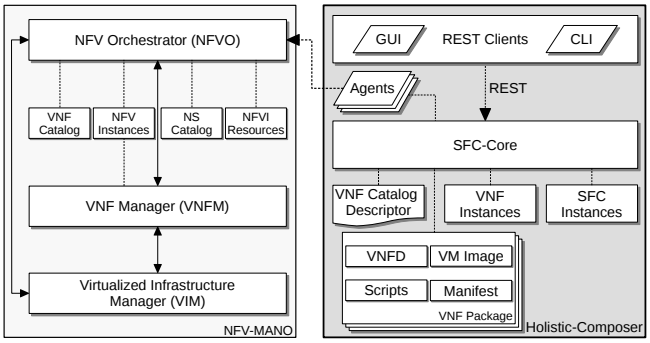

Fig. 2. Proposed architecture for the Holistic-Composer framework.

in which the network operator specifies the service chain in terms of which VNFs are to be used and the path connecting those VNFs.

Fig. 2 shows the Holistic-Composer along with the ETSI NFV-MANO framework. The proposed architecture is compatible with and complements NFV-MANO. We use communication agents to provide a generic solution that can be used on different NFV platforms. These communication agents are responsible for interacting with the NFV orchestrators. Hence, for each NFV orchestrator, there must be a communication agent responsible for abstracting those interactions.

The *SFC-Core* is the main component of the proposed framework and was developed with the purpose to reduce the complexity of composing and enabling lifecycle management of service chains in different NFV orchestrators. For this reason, the *SFC-Core* provides a standardized communication interface with its client applications using the REpresentational State Transfer (REST) model. Client applications range from end-user applications to other NFV frameworks. These applications are illustrated in Fig. 2 as *REST Clients* and may include applications that use a Graphical User Interface (GUI) as well as those that use a Command Line Interface (CLI).

In addition, the *SFC-Core* component is in charge of conducting and validating operations requested from client applications. One of these operations refers to the management of the local VNF repository, represented by the *VNF Package* element in Fig. 2. The *VNF Package* format employed by Holistic-Composer follows the ETSI model [14] in order to allow the interoperability of the proposed framework with other standardized VNF-based systems. Moreover, each *VNF Package* has a VNF Descriptor (VNFD) and the specific scripts to be performed after instantiating the corresponding VNF on the NFV Infrastructure.

Furthermore, the *VM Image* element of Fig. 2 represents the VNF image that a *VNF Package* may contain or not, since the *VM Image* may be fetched directly from the NFVO repository. The *Manifest* element represents the *VNF Package* descriptor and includes information such as the VNF name, developer, version, and release date.

The *SFC-Core* also maintains the *VNF Catalog Descriptor* element to handle metadata information, such as unique identifiers, names, storage location, descriptions, categories, and the function types of *VNF Packages* stored in the local repository. For instance, the function type shows whether a given VNF runs on Click-on-OSv. Since Click-on-OSv requires extra steps for its configuration these are on a specific Element Management (EM). The category (*e.g.,* firewall or load balancer) specifies the given functionality, and can be used so that the *SFC-Core* can suggest VNFs for a specific composition or even compose network services in an automated way.

In order to allow the interoperability of the proposed framework and different NFV platforms and SFC solutions, a catalog keeps information about VNFs instantiated by the Holistic-Composer, which are depicted in Fig. 2 as *VNF Instances*. Consequently, it is possible to distinguish VNFs instantiated by other tools of those instantiated by the Holistic-Composer.

The *SFC Instances* component maps active service chain instances and the corresponding VNFs running on the NFV orchestrator. The *SFC Instances* component also obtains information about the instantiated SFCs from the local and NFV orchestrator repositories. This allows, for example, releasing unused resources after destroying a service chain, and also identify which particular SFCs are running on different NFV orchestrators.

IV. Implementation of the Proposed Architecture

A prototype of the proposed framework was implemented with the NFV Enablers OpenStack, Click-on-OSv, and Tacker to compose and manage the lifecycle of SFCs as described below. According to the ETSI, NFV Enablers are tools that contribute to the development and deployment of NFV [15].

OpenStack [16] is a cloud computing platform that provides virtual physical resources used by VNFs, thus acting as a VIM in the NFV environment. Click-on-OSv [8] allows the creation and execution of high-performance and minimalist VNFs built with the Click Modular Router which run on the OSv unikernel. In addition, Click-on-OSv supports multiple hypervisors (*e.g.*, Xen, KVM, VMware, VirtualBox) and provides FCAPS (*Fault, Configuration, Accounting, Performance and Security*) functionalities through a REST API. Tacker [9] is an official OpenStack platform project that implements capabilities of both VNF Manager and NFV Orchestrator, as well as allows orchestrating network services using end-to-end VNFs.

The Holistic-Composer was implemented in Python. We used the *Flask* library to implement specific *SFC-Core* features that provide a standardized REST interface. REST clients can interact with the *SFC-Core* to compose and manage the lifecycle of service chains. All the information used on REST Clients remains stored in a database which is managed by the *SFC-Core*. This database stores information from *VNF Packages*, *VNF Catalog Descriptor*, *VNF Instances*, and *SFC Instances*. Directories generated by *SFC-Core* store the raw *VNF Packages* content.

The *SFC-Core* implementation also enables JSON and TOSCA/YAML formats for VNF descriptors, as required by NFV orchestrators such as Tacker and the Open Baton. Furthermore, the *VNF Instances* element stores information

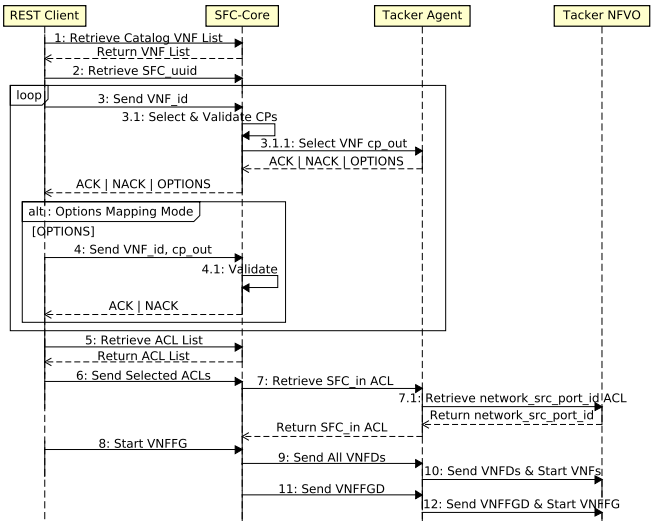

Fig. 3. Messages exchanged between the framework and the Tacker NFVO.

such as the unique identifier of the *VNF Package* being used, and also the VNFD and VNF instance identifiers generated by the NFV orchestrator. Storing NFVO generated records is essential to retrieve status information of VNFs and besides it is used to destroy VNFs and their descriptors when removing a service chain running on the NFV orchestrator.

Similarly, the content of the *SFC Instances* element includes information such as the SFC unique local identifier, the list of VNF instances involved in the service chain composition, as well as identifiers of SFC descriptors and their VNF Forwarding Graphs (VNFFG) running on the NFVO. A communication agent was also implemented using the *Requests* library to exchange information with the REST API provided by the Tacker. A client application was implemented to test the proposed framework. This application runs through CLI and assists the composition and management of service chains. We also implemented an Element Management (EM) of the NFV-MANO specification for Click-on-OSv to handle its FCAPS functionalities.

A. Service Chain Composition

Fig. 3 shows at a high level a typical sequence of messages exchanged between the framework components and the Tacker NFV orchestrator during the SFC composition process.

The first message "*1: Retrieve Catalog VNF List*" sent from the *REST Client* to *SFC-Core* requests a list of available VNFs stored in the local repository. As a result, the *SFC-Core* queries the *VNF Catalog Descriptor* and replies with the full list of VNFs stored in the catalog. Subsequently, the *REST Client* sends the message "*2: Retrieve SFC_uuid*" to the *SFC-Core* in order to get a universal unique identifier to be used as a label to identify the service chain composition on all next message exchanges. This allows multiple service chain compositions to be performed concurrently.

Message "*3: Send VNF_id*" is then employed to submit the unique ID of the selected VNF to *SFC-Core*. As soon as the *SFC-Core* receives that message it selects and validates the suitable *Connection Points* (*CP*) using the corresponding VNF descriptor (VNFD) which is stored in the local repository. The selection and validation operations are triggered by message "*3.1: Select & Validate CPs*" and the steps described next are executed to select the proper *CPs*. First, the *SFC-Core* searches for a *CP* which has a *Virtual Link* (*i.e.*, subnet name) that corresponds to the same *Virtual Link* of the output *CP* of the previous VNF in the SFC. If this process succeeds, then the input *CP* is selected. In case no *CP* matches the prior *CP's Virtual Link*, a *NACK* message is returned to the *Rest Client*. In case the VNF is the first of the SFC, then the first valid *CP* of that VNFD is chosen. Essentially, NFVO orchestrators communicate in a way that allows the execution of management operations on VNFs using isolated network traffic. For that reason, *CPs* attached to management *Virtual Links* are not valid for the SFC composition and thus are not used for this purpose.

On the other hand, the output *CP* selection rules depend on the NFVO. In general, if a VNF has just one *CP* and the input *CP* rule is satisfied (*i.e.*, the *Virtual Link* matches), this *CP* is selected as input and output for that VNF and a *ACK* message is returned. If the output *CP* could not be selected, the *SFC-Core* places that responsibility on the particular NFVO communication agent through message "*3.1.1: Select VNF cp_out*". We are currently using the *Tacker Agent* as the NFVO communication agent.

However, other rules could be applied by communication agents when using other NFVO orchestrators that allow SFCs to cross different subnets, or allow SFCs consisting of VNFs that use the same *Virtual Link* for various *CPs*. In this case, the communication agent searches for all valid *CPs* in the VNF descriptor and applies one of the following rules: if there is just one *CP* the process succeeds by selecting the output *CP* and returns an *ACK* status code. On the other hand, a message with the *OPTIONS* status code along with the list of suitable *CPs* is returned if there is more than one applicable *CP*. Indeed, selecting the output *CP* depends on particular NFVO features since each NFVO has a different approach to form SFCs. In addition, multiple *CPs* might share the same *Virtual Link*. This happens in VNFs acting as SFC branching nodes, which can also use more than one *Virtual Link*.

It is important to highlight that each different NFVO presents specific restrictions. For instance, at the time of this writing, the *Tacker NFVO* does not allow SFCs that use more than one subnet. It also does not allow the creation of SFPs (*Service Function Paths*) which have VNFs that use two *CPs* sharing the same *Virtual Link* in the same SFC. In addition, if neither the *SFC-Core* nor the *Tacker Agent* could select the right *CP*, a message with the *OPTIONS* status code and a list of potential *CPs* is sent to the *REST Client*. This message requires the user to select a *CP*. As a result, the alternative block "*alt: Options Mapping Mode*" takes place. Message "*4: Send VNF_id, cp_out*" carries the output *CP* selected by the network operator from the *Rest Client* to the *SFC-Core*. As soon as the *SFC-Core* receives that message, it validates the

output *CP* and sends a reply indicating whether that VNF has been successfully included in the SFC. This message exchange pattern (*i.e.*, loop block) is executed repeatedly until all VNFs have been included in the SFC composition.

After all VNFs are successfully included in the SFC, the *REST Client* application requests the Access Control List (ACL) restrictions to the *SFC-Core* using message "*5: Retrieve ACL List*". Afterwards, the operator makes the required assignments to the ACL constraints (*e.g.*, *ip_proto: 6*, *destination_port_range: 80-80* and *ip_dst_prefix: 10.10.0.5/32*). Message "*6: Send Selected ACLs*" is used to send all this information to the *SFC-Core*. Since the OpenStack Tacker also requires selecting the Open vSwitch network traffic source port identifier, the *SFC-Core* sends message "*7: Retrieve SFC_in ACL*" to the *Tacker Agent* requesting the source port identifier. As a result, the *Tacker Agent* retrieves this identifier in message "*7.1: Retrieve network_src_port_id ACL*" using the OpenStack Tacker REST API. As soon as this identifier is received by the *SFC-Core*, it is added to the ACL constraints.

In addition, the *REST Client* application sends message "*8: Start VNFFG*" to the *SFC-Core* in order to instantiate the SFC. Once the *SFC-Core* receives this message it sends all VNF descriptors to the *Tacker Agent* with message "*9: Send All VNFDs*". The *Tacker Agent* in turn, submits and instantiates all the VNF descriptors to the *Tacker NFVO*, which is represented by message "*10: Send VNFDs & Start VNFs*". Subsequently, the *SFC-Core* sends the generated SFC descriptor (*i.e.*, VNFFGD) to the *Tacker Agent* through message "*11: Send VNFFGD*". Finally, the *Tacker Agent* sends the VNFFGD to the *Tacker NFVO* and requests its instantiation with message "*12: Send VNFFGD & Start VNFFG*".

B. Evaluation: Concurrent

This section describes experiments executed to evaluate the prototype performance. We were particularly interested on the time it takes to compose SFCs in different scenarios as described below. The experiments were executed on an Intel(R) Core(TM) i7-6700HQ CPU with 4 cores up to 3.5 GHz, and 12 GiB of DDR4 RAM at 2133 MHz. We used the Linux Ubuntu 16.04.3 running kernel 4.13.0-37-generic x86_64, and the Apache HTTP Server 2.4.18 with the *mod_wsgi* adapter *libapache2-mod-wsgi-py3* to provide a standard Python WSGI (Web Server Gateway Interface) between the web server and the *SFC-Core Flask* library. Apache also was configured to have 4 processes listening for incoming requests to the *SFC-Core*. We also employed the *memcached* daemon which is a distributed memory object cache system to do inter-process communication, as the Apache HTTP Server handles each stateless request using a different process. We also employed the MongoDB database.

All experiments were performed 100 times, averages are presented. A *REST Client* was implemented in order to run SFC composition experiments. We highlight that the client application simply sends REST requests to the *SFC-Core* which composes and manages the SFC. The elapsed time of each test was recorded by the *REST Client* at the time instants the SFC composition process starts and completes.

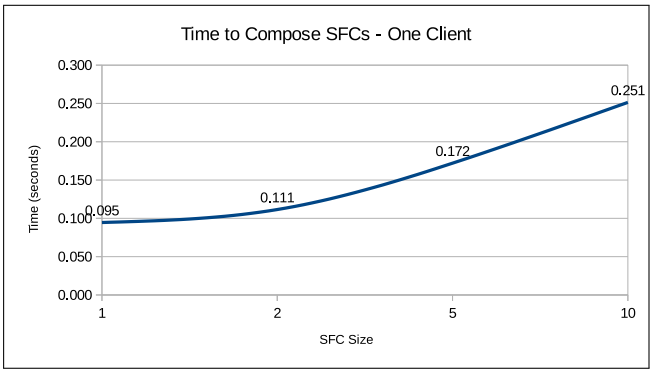

Fig. 4. Time to compose SFCs with different sizes.

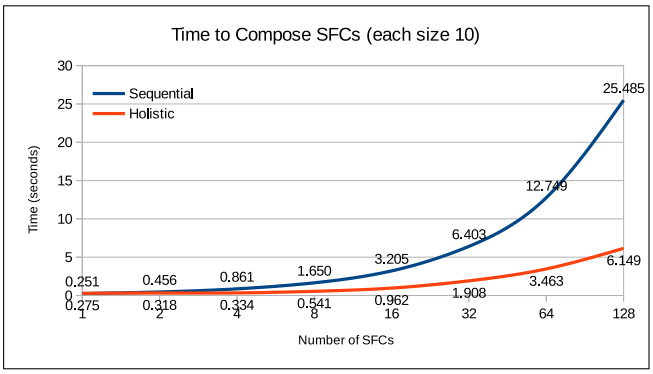

Fig. 5. Time to compose different numbers of SFCs.

Fig. 4 shows the time spent by a single client to compose SFCs that consist of 1, 2, 5, and 10 VNFs. As can be seen, after adding the second VNF, around 17.5 ms are needed on average to add each of the next VNFs, time thus follows a roughly linear pattern. Fig. 5 shows the time to compose different numbers of SFCs all consisting of 10 VNFs, using both the Sequential and the Holistic approaches. In the Sequential approach, 25.485 seconds were necessary to compose 128 SFCs sequentially. On the other hand, when using the Holistic approach all the 128 SFCs are composed concurrently, requiring in average 6.149 seconds to complete each composition, which is around 4.145 times faster than the Sequential approach.

V. Related Work

NFV-related technologies have become increasingly relevant. The Open Source MANO (OSM) project [17] implements the NFV-MANO framework and has three main components known as OpenVIM, OpenMANO and OpenMANO-GUI. In addition, the OpenMANO component implements the NFVO functional block and uses the REST model to provide VNF management functionalities, network services and their corresponding templates. Although the OSM project provides a web interface to model network services through VNF composition, users still need to deal with the complexity of

setting descriptors manually, furthermore it does not support the Click-on-OSv FCAPS operations.

Open Baton [18] also implements the ETSI NFV-MANO specification. Its main components include an NFV Orchestrator, a generic VNF Manager (VNFM), and a Software Development Kit (SDK) with libraries to implement specific VNFMs. Open Baton also uses OpenStack as the standard implementation for the VIM component of the NFV-MANO specification. Although Open Baton offers a web interface to manage VNFs and network services, the capabilities to create these services are limited to the upload of VNF Packages created manually or by third-party tools. Currently, the interface does not allow composing VNFs to create service chains. Also, Open Baton does not natively support FCAPS operations required by VNFs that run over Click-on-OSv.

Salsano et al. [19] proposed the *Reusable Functional Blocks Description and Composition Language Design, Deploy and Direct* (RDCL 3D) framework in the context of service chains. Although the RDCL 3D enables the composition of VNFs in different NFV platforms through communication plugins, it is necessary to modify OpenVIM so that it is possible to execute VNFs built with ClickOS [10]. Our proposal does not require any ETSI VIM modifications to run Click-on-OSv. Another distinction between RDCL 3D and our work refers to the use of the Click Modular Router. While RDCL 3D uses ClickOS which only runs on a Xen Server, our architecture is based on Click-on-OSv which is portable and can be executed *e.g.*, on KVM, Xen Server, VMware, or VirtualBox.

Finally, RDCL 3D uses proprietary descriptors to specify SFC projects and also implements the service chain composition logic on the client side (*i.e.*, GUI). In our proposal, the *SFC-Core* component is in charge of abstracting both the implementation and validation complexities of service chain composition on the server side through a REST API. This allows a larger number of NFV enablers to be employed by the Holistic-Composer.

VI. Conclusion

In this work we proposed the Holistic-Composer framework to simplify the composition and lifecycle management of VNF service chains on multiple NFV platforms. The framework was specified according to the ETSI NFV-MANO architecture and can be used along with different NFV platforms and virtually any end-user applications. A prototype was implemented that allows the composition and lifecycle management of SFCs built with Click-on-OSv running on the OpenStack Tacker NFV orchestration platform. Multiple SFC compositions can be done concurrently. In particular, we highlight the REST API provided by the *SFC-Core* component that frees the operator from understanding minute details to configure and operate NFV orchestrators by defining generic operations for the composition and lifecycle management of SFCs.

Experimental results are shown for the time it takes to compose a single SFC with a growing number of VNFs and the time to compose multiple SFCs each consisting of 10 VNFs. The holistic framework was able to compose concurrently 128 SFCs around 4.145 times faster than a sequential approach. As future work, we are currently implementing a communication agent for the OSM Orchestrator. Other orchestrators such as Open Baton are also planned. Future work also includes extending the architecture to allow resource management, in particular elasticity. Moreover, allowing service chains to share VNF instances can also be useful.

References

[1] M. Chiosi, S. Wright, D. Clarke, P. Willis, L. Johnson, M. Bugenhagen, J. Feger, W. Khan, C. Chunfeng, and et al, "Network Functions Virtualisation (NFV): Network Operator Perspectives on Industry Progress," ETSI, White Paper, Oct. 2013.

[2] F. Z. Yousaf, M. Bredel, S. Schaller, and F. Schneider, "NFV and SDN - Key Technology Enablers for 5G Networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2468–2478, 2017.

[3] J. Quittek, P. Bauskar, T. BenMeriem, A. Bennett, M. Besson, P. M. Bruun, J.-M. Calmel, B. Chatras, and et al, "Network Functions Virtualisation (NFV); Management and Orchestration. GS NFV-MAN 001 V1.1.1," ETSI, Group Specification, dec 2014.

[4] J. Halpern and C. Pignataro, "Service function chaining (sfc) architecture," Internet Requests for Comments, RFC Editor, RFC 7665, October 2015.

[5] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A survey on service function chaining," *J. Netw. Comput. Appl.*, vol. 75, no. C, pp. 138–155, Nov. 2016.

[6] IETF, "Service function chaining (sfc) - documents," 2018. [Online]. Available: https://datatracker.ietf.org/wg/sfc/documents/

[7] ETSI, NFVISG, "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV," ETSI, Group Specification, dec 2014.

[8] L. da Cruz Marcuzzo, V. F. Garcia, V. Cunha, D. Corujo, J. P. Barraca, R. L. Aguiar, A. E. Schaeffer-Filho, L. Z. Granville, and C. R. P. dos Santos, "Click-on-OSv: A platform for running Click-based middleboxes," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, may 2017, pp. 885–886.

[9] OpenStack Tacker, "Tacker - openstack nfv orchestration," 2018. [Online]. Available: https://wiki.openstack.org/wiki/Tacker

[10] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 459–473.

[11] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[12] P. Quinn and T. Nadeau, "Problem statement for service function chaining," Internet Requests for Comments, RFC Editor, RFC 7498, April 2015.

[13] P. Quinn, U. Elzur, and C. Pignataro, "Network service header (nsh)," Internet Requests for Comments, RFC Editor, RFC 8300, January 2018.

[14] A. Kojukhov, A. M. de Nicolas, B. Chatras, D. Druta, D. Gassanov, M. Brunner, M. Brenner, S. Li, T. Nguyenphu, U. Rauschenbach, and Z. Sacks, "Network functions virtualisation (nfv) release 2; protocols and data models; vnf package specification. GS NFV-SOL 004 V2.3.1," ETSI, Group Specification, jul 2017.

[15] M. Chiosi, D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, C. Cui, H. Deng, and U. Michel, "Network functions virtualisation: An introduction, benefits, enablers, challenges & call for action," ETSI, White Paper, Oct. 2012.

[16] OpenStack, "Openstack - open source software for creating private and public clouds," 2018. [Online]. Available: https://www.openstack.org/

[17] ETSI, "Open source mano," 2018. [Online]. Available: https://osm.etsi.org/

[18] F. Fokus and B. Tu, "Open baton: an open source reference implementation of the etsi network function virtualization mano specification," 2018. [Online]. Available: http://openbaton.github.io/

[19] S. Salsano, F. Lombardo, C. Pisa, P. Greto, and N. B. Melazzi, "RDCL 3D, a Model Agnostic Web Framework for the Design and Composition of NFV Services," in *IEEE Conference on Network Function Virtualization and Software Defined Networks*, no. 1, oct 2017.