Building Multi-domain Service Function Chains Based on Multiple NFV Orchestrators

Alexandre Huff*†, Giovanni Venâncio†, Vinícius Fulber Garcia† and Elias P. Duarte Jr.†
*Federal Technological University of Paraná, UTFPR, Toledo, Brazil
†Federal University of Paraná, UFPR, Curitiba, Brazil
Email: alexandrehuff@utfpr.edu.br, {ahuff, gvsouza, vfgarcia, elias}@inf.ufpr.br

***Abstract*—Service Function Chains (SFCs) are compositions of Virtual Network Functions (VNFs) designed to provide complex network services. In this work, we propose a strategy to build an SFC across multiple domains and multiple clouds using multiple NFV platforms, which we call a Multi-SFC. To the best of our knowledge, this is the first solution to allow an SFC to be built across multiple different orchestrators – although there are other solutions for multiple domains and clouds. The basic building block of the proposed strategy is the SFC segment, in which all VNFs are connected within a single cloud/domain/platform. A pair of different segments is interconnected through a VNF tunnel that consists of a pair of VNFs, each interfacing one of the connected segments. A tunnel can be implemented with different technologies such as a VPN or VXLAN. The main advantage of the Multi-SFC strategy is that it is a holistic approach that allows operators to deploy SFCs on multiple clouds/domains/platforms without having to deal with a myriad of minute details required to configure and interconnect the different underlying technologies. A prototype was implemented as a proof of concept and experimental results are presented.**

***Index Terms*—NFV, SFC, Multi-domain, Multi-SFC**

I. Introduction

Network Function Virtualization (NFV) allows the implementation in software of network services which traditionally run on hardware middleboxes. A Virtualized Network Function (VNF) is executed on commercial off-the-shelf hardware, improving the flexibility and decreasing costs [1], [2]. The European Telecommunication Standards Institute (ETSI) has proposed the NFV-MANO standard architecture for NFV Management and Orchestration [3]. NFV-MANO specifies the functionalities required for VNF provisioning and related operations.

One of the main goals of NFV-MANO is to specify standard ways to coordinate the composition of VNFs to form Service Function Chains (SFCs) that provide complex network services. An SFC consists of a composition of VNFs on a topology through which traffic is steered in a predefined order [4], [5]. Usually, a flow identifier is employed to steer traffic from a function to next along the SFC – this contrasts with conventional routing in which all decisions are taken based on the destination IP address.

Current systems usually allow the instantiation and orchestration of all VNFs of an SFC composition to be done on a single NFV platform [6]–[11]. Although in some cases multiple instances of the same orchestrator are permitted, to the best of our knowledge no system allows multiple different orchestrators to be used. As multiple different platforms have become available [2], [12] it is just natural to allow an SFC to be built on several clouds/platforms. The need to compose SFCs using VNFs running on multiple domains also arises when network services are composed of VNFs that natively run on specific domains. Another reason is to allow VNFs to access resources available at specific domains.

In this work, we propose a strategy that allows the execution of an SFC across multiple clouds of multiple administrative domains orchestrated by multiple NFV platforms. We call this strategy Multi-SFC. In practice, SFC composition using different NFV platforms requires specific, detailed knowledge of the NFV orchestrators, becoming a very complex task to the network operators. This is the case even for NFV platforms such as Tacker [6], Open Source MANO (OSM) [7], and Open Baton [13] all of which implement the standard NFV-MANO NFV Orchestrator (NFVO). In addition, the global configuration of an SFC running across different administrative domains (i.e. steering traffic through all segments of each cloud/domain/platform) involves coordination efforts from network operators of all domains. Although the ETSI has discussed strategies for the communication of NFV orchestrators on different administrative domains [14], the problem is still far from solved, as one has to deal with specific features and different NFVO data models.

The Multi-SFC architecture proposed in this work relies on a holistic approach and defines a framework which provides high-level abstractions for the management and composition of Multi-SFCs. The configuration of the NFV infrastructure is taken to a higher level of abstraction by leveraging traffic steering over multiple clouds/domains/platforms. The basic building block of the proposed strategy is the SFC segment, in which all VNFs are connected within a single cloud/domain/platform. A pair of different segments is interconnected through a VNF tunnel. Tunnels can be based on different technologies, such as VPN (Virtual Private Network) or VXLAN (Virtual eXtensible LAN) which are instantiated as VNFs at the incoming and outgoing points of the SFC segments being connected. Overall, the main advantage of this holistic approach is that it abstracts the myriad of low-level minute configurations required to compose and manage SFC lifecycle over multiple clouds/domains/platforms. Thus,

This work was partially supported by CAPES Finance Code 001 and CNPq grant 311451/2016-0.

978-1-7281-8159-2/20/$31.00 ©2020 IEEE

the user with the permissions to set up a virtual network in each domain can execute a Multi-SFC without the manual intervention of network operators.

A proof-of-concept prototype was implemented based on two NFV platforms (Tacker and Open Source MANO) along with two different versions of OpenStack on two different administrative domains. Experiments are presented that were executed to evaluate the Multi-SFC strategy in terms of interoperability and overhead. Results allow the conclusion that the Multi-SFC strategy is an effective solution, which to best of our knowledge is the first NFV-MANO-compliant strategy to build SFCs across multiple clouds, multiple domains, *and* multiple different NFV orchestrators.

The rest of this paper is organized as follows. Section II presents related work. The Multi-SFC architecture is described in Section III. The prototype and experiments are in Section IV. Section V concludes the paper and presents future work.

II. Related Work

Francescon et al. proposed the X-MANO framework [15] which enables the orchestration of network services across multiple domains. X-MANO only allows the composition of previously created network services through specific NFVO interfaces at each domain. X-MANO also requires manual configuration and instantiation of inter-domain links.

A mechanism for the automated establishment of dynamic Virtual Private Networks (VPN) in the NFV context has been recently proposed [16]. The purpose is to provide encryption and security to connect the functions of an SFC.

The TeNOR NFV orchestrator [17] allows management and control of the network services on distributed virtualized infrastructures. TeNOR automates SFC configuration and instantiation. Although a mapping-based solution is provided to instantiate an SFC over multiple Points of Presence (PoPs), TeNOR does not allow to create SFCs across multiple different NFV orchestrators.

OPNFV (Open Platform for NFV) [18] from the Linux Foundation aims at simplifying the development and deployment of NFV components. OPNFV enables the interoperability of NFV solutions of different developers, but does not allow SFCs across multiple NFV orchestrators.

Blue Planet MDSO (Multi-Domain Service Orchestration) [8] is a framework for composing and managing network services over multiple domains and NFV infrastructures. Despite of simplifying SFC configuration and instantiation, this framework requires the usage of its own NFV orchestrator.

Cloudify [9] is a project that allows the integration of virtual and physical network functions and provides an NFVO and a generic VNF Manager (VNFM) to orchestrate several clouds. The system employs pre-configured virtual routers to interconnect different clouds through tunnels. Although Cloudify orchestrates several clouds, it does not allow SFCs to be constructed on multiple different NFV orchestrators.

pSmart [19] allows SFCs on multiple domains to provide cost-effective resource utilization. pSMART aims at reducing privacy and security risks, and employs a learning based decision process for SFC mapping. Another related work investigates strategies to deploy SFC across multiple datacenters [20] which takes into consideration not only cost, but also the usage of backup functions to improve SFC reliability.

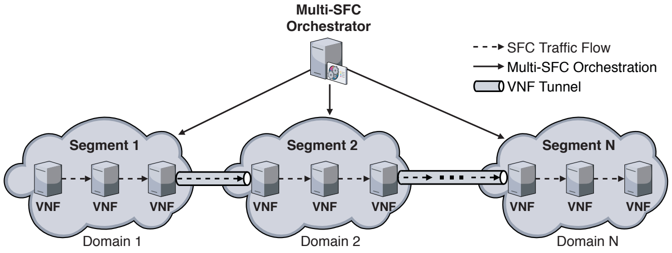

Fig. 1. Multi-SFC: Segmentation.

The 5G Exchange (5GEx) project [21] was defined to coordinate the allocation and efficient usage of compute, storage, and networking resources to deploy services in 5G networks. The 5GEx project relies on SDN and NFV techonologies for provisioning services over multiple-technologies and spanning across multiple 5G operators. 5GEx relies on a single NFV orchestrator. Finally, in [22] the authors propose a framework for the orchestration of 5G network services across multiple domains. The objective is to optimize both resource utilization and revenue, while matching service requirements.

III. The Multi-SFC

In this section we describe the Multi-SFC solution for the composition and management of SFCs distributed across multiple clouds, domains, and NFV platforms. We assume that a domain is formed by a collection of systems and networks operated by a single organization or administrative authority [3]. One or more clouds are hosted at each domain; each cloud runs an NFV platform which corresponds to a set of systems running the NFV-MANO stack.

The basic building block of a Multi-SFC is the *segment*, which consists of VNFs running on a single cloud/domain/platform as shown in Fig. 1. A pair of different segments is interconnected through a VNF tunnel. Tunnels can be based for instance on VPN or VXLAN technologies, and are instantiated as VNFs at the incoming and outgoing points of the SFC segments being connected. After a segment is instantiated on a specific domain, it is connected to segments running on the other domains.

The *Multi-SFC Orchestrator* shown in Fig. 1 is responsible for managing the Multi-SFC lifecycle, which consists of the composition, instantiation, execution and destruction of a Multi-SFC on multiple domains/clouds/platforms. The *Multi-SFC Orchestrator* provides a high-level and generic API (Application Programming Interface) to allow Multi-SFC composition and management. By using this interface the user specifies the SFC as sequence of VNFs, and maps on which cloud/domain each VNF is to be instantiated by the *Multi-SFC Orchestrator*.

Fig. 2 shows the architecture of a Multi-SFC with two segments split on a pair of domains. The *Multi-SFC Or-*

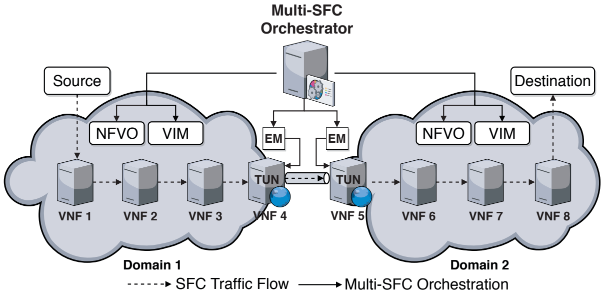

Fig. 2. A Multi-SFC split on a pair of domains.

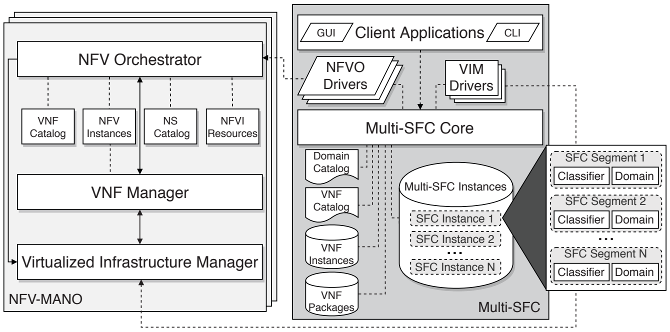

Fig. 3. The *Multi-SFC Orchestrator* architecture.

chestrator does all the configuration required to steer traffic across these segments. Tunnels implemented as VNFs are employed to connect Multi-SFC segments across multiple domains becoming part of the SFC composition. This strategy allows transparent inter-domain communication, with a VNF implementing a tunnel attached to the end of a segment and to the beginning of the next. According to the IETF SFC architecture [4] these tunnels correspond to Service Function Forwarders (SFFs). In this context the Multi-SFC introduces the support for the orchestration of multi-domain SFFs.

Fig. 3 shows the *Multi-SFC Orchestrator* architecture which is proposed as a generic and extensible solution aligned to the definitions of the NFV-MANO architecture. The NFV-MANO is also shown in the figure, illustrating its interaction with the *Multi-SFC Orchestrator* modules. The proposed architecture allows the integration of multiple NFV platforms and several client applications.

The *Multi-SFC Core* is the main module of the *Multi-SFC Orchestrator*. This module is in charge of coordinating SFC composition on multiple NFV orchestrators, managing the Multi-SFC tunnels, as well as validating requests executed by client applications. A centralized and generic communication API is provided by the *Multi-SFC Core* presenting a REST interface for SFC composition and management. The *Multi-SFC Core* leverages the VNF Tunnel Element Management (EM) to allow tunnel configuration. The main operations of the API provided by the *Multi-SFC Core* to the *Client Applications* to compose a Multi-SFC are described next.

`GET /msfc/uuid`: generates and retrieves a unique identifier (uuid) in order to compose a new Multi-SFC (msfc). The unique identifier is employed by several operations to identify the different segments that form a Multi-SFC.

`GET /catalog/domains`: retrieves information of all domains stored in the *Domain Catalog*. Information regarding all NVFO and VIM platforms as well as all VNF tunnel technologies available on that domain is returned by this operation. This operation is employed by client applications for instance, to check whether two different domains match in terms of tunnel technologies available, and internally by the *SFC-Core* to gather information about endpoints and authentication methods available on each NFV platform.

`GET /catalog/vnfs/<dom-id>/<plat-id>`: lists all VNF packages stored in the *Domain Catalog* repository belonging to a specific domain `<dom-id>`. Since multiple NFV platforms are allowed, only compatible VNF packages (`<plat-id>`) are returned by this operation.

`POST /msfc/sfp/compose`: operation for the composition of a segment, which chains its VNFs. This operation receives as input the domain, segment, and the VNF Package ID stored in the *VNF Catalog*. A single VNF is chained per call of the operation, the input network interface of this VNF is configured to be the previous VNF already in the chain. Unless there are several alternatives, the output network interface is configured automatically, otherwise the user is given a set of choices. Note that this approach allows chaining VNFs that act as branches allowing traffic to be sent on different Service Function Paths (SFPs).

`POST /msfc/source`: after the chain has been composed, the next task is to configure the incoming traffic. The input of this operation indicates whether the traffic source of the first segment of the Multi-SFC is internal or external. In case of internal traffic, the *Multi-SFC Core* allows the user to chose as traffic source either a running VNF or a VNF Package stored in the *VNF Catalog*. In case of external traffic, the network including the router are configured to allow the incoming traffic to be received.

`GET /msfc/acl/<sfc-id>`: returns all classifier policies of the NFV platform of the first Multi-SFC segment. A classifier policy is defined as an Access Control List (`acl`) that specifies attributes of the incoming traffic, such as source and destination addresses, ports, protocols, flow labels, etc.

`POST /msfc/acl`: receives as input the policies specified by the user for the incoming traffic and configures the corresponding SFC classifiers of the NFV platforms being used along the Multi-SFC. Classifier policies of the next segments are mapped and configured taking into consideration the configuration of the first segment classifier and the different NFV platforms being used.

`GET /tunnel/em`: returns the Element Management (EM) of the VNF tunnels. The EM is used to configure the tunnel. After it is instantiated, a VNF tunnel gets the corresponding EM to configure itself.

`POST /msfc/start`: this operation instantiates all segments of a Multi-SFC descriptor on their corresponding NFV domains and orchestrators. The Multi-SFC identifier is received as argument. The operation triggers the required operations to instantiate and configure the Multi-SFC, including sending VNF Packages and SFC descriptors to the corresponding NFV platforms, instantiating VNFs and Multi-SFC segments, configuring tunnels, configuring segment routing, and configuring security policies on each NFV platform. This operation can only be executed to effectively instantiate and configure the whole Multi-SFC after the previous operations have successfully created all the corresponding network service descriptors.

The description of the *Multi-SFC Core* API above is non-exhaustive. Other operations include VNF Package management, VNF Descriptor management as well as operations to instantiate, access, and destroy VNFs on the multiple clouds/domains/platforms.

In addition to the *Multi-SFC Core*, Fig. 3 also shows both *NFVO Drivers* and *VIM Drivers*. *NFVO drivers* are responsible for the abstraction of and communication with different NFV orchestrators that can be employed by the *Multi-SFC Core*. Generic operations of the *Multi-SFC Core* are translated by the *NFVO Drivers* to the specific operations and features of their corresponding NFV orchestrators. Each driver implements a set of functionalities that allow the composition and orchestration of SFCs, ranging from the management of VNFs and SFC descriptors to their instantiation and destruction.

The main operations that a *NFVO Driver* must include are those for the instantiation, monitoring and destruction of VNFs and SFCs and those to manage the Service Function Path (SFP), such as a `compose_sfp` operation that connects VNFs along a Service Function Path of a particular Multi-SFC segment using information available at the corresponding *VNF Packages*. Other operations include: `get_sfc_traffic_src` retrieves VNFs eligible to be configured as traffic source of the first Multi-SFC segment. `configure_traffic_src_policy` configures the SFC classifier to encapsulate and forward the incoming traffic which can be internal or external. This operation selects in the cloud infrastructure the most appropriate network interfaces both for internal (VNFs/VNF packages) and external (virtual routers) traffic. Finally, there are also operations to manage classifier policies. `get_available_policies` returns a list of policies and constraints which can be applied on a given SFC segment classifier given the corresponding NFV platform. `configure_policies` configures the SFC classifier by defining constraints for the input traffic of each SFC segment. This operation is in charge of setting up all the Multi-SFC user policies related to the incoming traffic. `get_configured_policies`: returns the list of policies configured by the Multi-SFC classifier. The *Multi-SFC Core* leverages this operation to configure all classifiers of the Multi-SFC segments, all VNF tunnels, and firewalls rules on the VIM network nodes.

The *VIM Drivers* module is responsible for the transparent configuration of multi-domain and multi-platform interoperability. Each *VIM Driver* is employed to configure networks, manage inter-domain routing rules, and maintain the required security policies for each corresponding Multi-SFC segment. For each VIM, the correspondent *VIM Driver* must be available on the *Multi-SFC Orchestrator*.

The *VNF Catalog* also in Fig. 3 is used to manage metadata of *VNF Packages* which are stored in the *Multi-SFC Core* repository. The *Domain Catalog* provides meta-data required for inter-domain communication, such as end-points, NFV orchestrator types, VIMs, as well as authentication data. The *VNF Instances* keeps track of the different VNF instances running on the multiple NFV platforms. Finally, the *Multi-SFC Instances* stores and maps information related to each SFC instance. Each stored instance keeps information about its Multi-SFC segments and their VNFs, information about remote SFC segment classifiers as well as information about the domains and platforms hosting the segments. This allows for instance, to identify which particular SFC segment is running on a specific NFV platform, and also allows to release all cloud resources when destroying a service chain.

IV. Implementation & Experiments

A Multi-SFC prototype was implemented as proof of concept[1]. The implementation leverages several NFV enablers, in particular: OpenStack [23], Tacker [6], and OSM [7]. In the SFC context, the Multi-SFC itself can be regarded as an SFC enabler, since it abstracts and supports the composition and lifecycle management of distributed SFC segments. The OpenStack is employed as the NFV-MANO VIM while Tacker and OSM are the NFVOs. The Multi-SFC prototype was implemented in Python. The *Multi-SFC Core* API exports a REST interface which was implemented using the Python *Flask* library. Both *NFVO Drivers* were implemented using the Python *Requests* library to communicate with their corresponding NFVO northbound interfaces. We also implemented a *VIM Driver* for the OpenStack platform using the Python *Requests* library. Both orchestrators (Tacker and OSM) employ OpenStack to manage the lifecycle of VNFs and SFCs.

While the *NFVO Driver* API abstracts the instantiation, query, and destruction of the Multi-SFC segments distributed on different domains and NFV orchestrators, the *VIM Driver* API abstracts the configuration required to interconnect those Multi-SFC segments across different domains. An EM was implemented to manage tunnel configuration and to enable traffic steering across Multi-SFC segments. We used the Python *Flask* library to implement the EM and to provide a REST API for the IP tunnel lifecycle management. This lifecycle management is performed by the *Multi-SFC Core*. Each Multi-SFC tunnel is established on the specific endpoints of the segments after all VNFs have been instantiated. IPSec, VXLAN and GRE were employed to configure and instantiate the VPN tunnels. The *VIM Driver* is used to configure routes and the security restrictions on the incoming traffic for each segment. Finally, Multi-SFC composition and classifier policy configurations were set up through a REST client application. Multi-SFC composition is based on the holistic workflow [24].

[1] The source code is available at https://github.com/alexandre-huff/multi-sfc

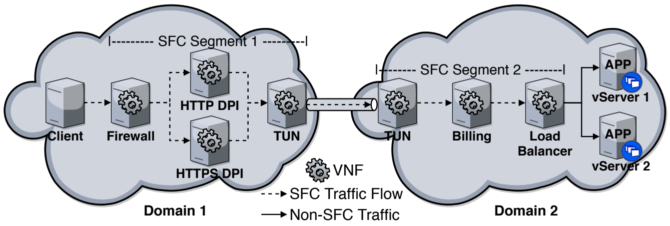

Fig. 4. Multi-SFC evaluation scenario.

Experiments were executed on two physical machines and other three virtual machines (VMs) running on a KVM virtualization system. Each of the two machines run a different OpenStack version, representing two different domains. One of these machines is based on an Intel(R) Core(TM) i7-6700HQ @ 2.6 GHz CPU with 4 cores; 6144K of L3 cache; 12 GiB of RAM; Ubuntu 18.04. The other machine is an Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz with 4 cores; 8192K of L3 cache; 8 GiB RAM; Ubuntu 18.04. The three VMs run on another machine based on the AMD Opteron(tm) Processor 6136 @ 2.4 GHz with 24 cores; 96 GiB of RAM; Ubuntu 20.04. Two different NFV orchestrators were used: Tacker and OSM each running on a separate VM. A third VM with the same configuration was employed as the OpenStack controller for OSM, while the Tacker VM runs both the Openstack controller and the NVFO. All VMs were set up with 8 vCPUs, 8 GiB of RAM, and Ubuntu 18.04. Each of the VNFs in this experiment runs Ubuntu Cloud 18.04 virtual machines with 1 vCPU and 256 MiB of RAM. Physical machines were interconnected on a Gigabit Ethernet network.

Fig. 4 shows the scenario used to evaluate the TCP goodput and latency of a Multi-SFC. We used *iPerf3* to generate traffic from the client to the servers on another domain and to measure the corresponding TCP goodput. ICMP (*ping*) was used to measure the latency. We implemented the *Firewall* VNF with *iptables* to filter and mark packets and *iproute* to forward marked packets to different network interfaces (branches). Both DPI VNFs implement packet forwarding. The VNF tunnels were implemented with IPSec, VXLAN, and GRE. The *Billing* VNF was also implemented as a packet forwarder. The *Load Balancer* VNF was implemented with *iptables* and distributes the traffic to the servers based on a hash, which is computed from the source IP address. Thus, all connections of the same client are delivered to the same server in the pool. Actually, this is an *iPerf3* requirement to measure the TCP goodput. *iPerf3* requires the establishment of at least two connections between the client and the server. Thus, all traffic of the experiment has to be sent to the same server even though there are other servers in the pool.

Initially we measured the TCP goodput between the *Client* and the *TUN* VNF in *Domain 1*, as well as between the *TUN* VNF and one *vServer* in *Domain 2*. This is a baseline measurement executed before the SFC was created. We ran this experiment for 30 times of 60 seconds each. The goodput was close to 37 Gbps in *Domain 1* and 26 Gbps in *Domain 2*. Since the *Client*, *vServer*, and both *TUN* VNFs ran on identical virtual machines, the difference of the TCP goodput between both domains is related to the difference of the CPUs of the physical machines.

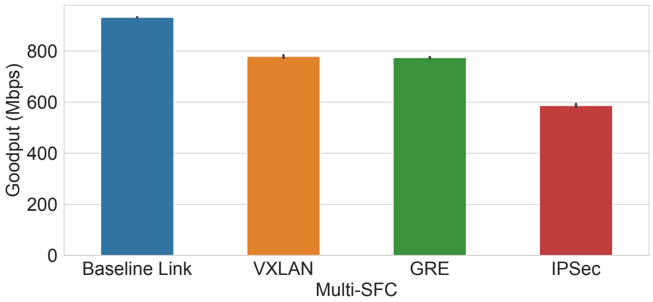

Fig. 5. Inter-domain TCP goodput using different VNF tunnels.

Next, we created the two SFC segments and measured the impact of the TCP goodput on each independently. Note that traffic was not forwarded from one domain to the other domain in this experiment. The measured TCP goodput for *Segment 1* was on average 16.679 Gbps, while for *Segment 2* it was 9.648 Gbps. The reduction of the goodput is due to fact that additional VNFs have been instantiated on each segment compute node. The SFC traffic is forwarded to and is processed by each VNF. Traffic classifiers for each segment also impose an overhead. Furthermore, the OpenStack virtual switch uses *veth* pairs (virtual Ethernet devices that connect through the kernel). These devices are known to present poor performance [25] but are used by OpenStack.

We also executed an experiment to evaluate the impact of the TCP goodput running a complete Multi-SFC consisting of the two segments shown in Fig. 4. This experiment provided end-to-end measurements, traffic was steered through all VNFs and between the two domains. As *Segment 1* has a branch, traffic is steered to one of the DPI functions. IPSec, VXLAN, and GRE VNF tunnels were employed to implement the tunnels. Fig. 5 shows the average for 30 executions of 60 seconds each with 99% confidence intervals. Before we set up the Multi-SFC we measured as a baseline the TCP goodput between the *Client* and one of the *vServers* without using VNFs, SFCs and the IP tunnel. The average baseline goodput was 932.1 Mbps. This results from the fact that the two VMs running the *Client* and the *vServer* are connected on a Gigabit Ethernet. When we executed the same measurement from the OpenStack compute nodes we got roughly the same results.

We then executed an experiment for a complete Multi-SFC using the VXLAN VNF tunnel. This experiment reached on average a goodput of 779.3 Mbps, as shown in Fig. 5. This result was expected, since the VXLAN tunnel has an overhead (packet header sizes alone are increased by 50 bytes each). We also evaluated the TCP goodput for a complete Multi-SFC running a GRE VNF tunnel. GRE encapsulation adds at least 24 extra bytes per packet. The Multi-SFC using GRE tunnel reached on average of 774.5 Mbps. We can conclude that GRE and VXLAN tunnels had roughly the same performance.

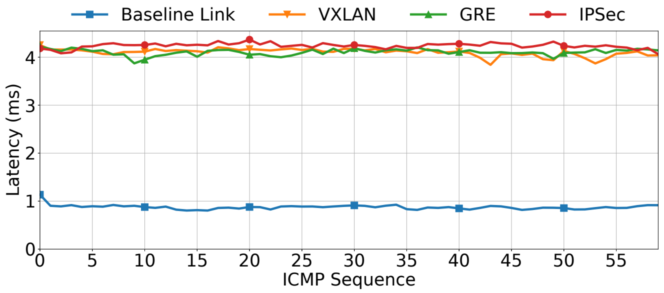

Fig. 6. Latency measured using different VNF tunnels.

Next we measured the TCP goodput for the IPSec VNF tunnel, which resulted on average in 586.9 Mpbs. In this case the lower TCP goodput is mainly due to the fact that IPSec VNFs make calls to system libraries to perform encryption (which is computationally expensive). This and all the tasks related to traffic encapsulation raised the *qemu-system* CPU utilization close to 100%. We also note that a single vCPU was allocated to the IPSec VNFs; the hardware of each compute node in which experiments were run was also a factor, as the 4 physical CPU cores are shared between the corresponding VNFs and the OpenStack compute node processes. We also note that one of the physical machines has more CPU cores but lower performance for I/O tasks (including networking).

We also executed an experiment to measure the latency. This experiment evaluated the latency of a complete Multi-SFC by steering the traffic through all VNFs on the two domains using IPSec, VXLAN and GRE VNF tunnels. Results are shown in Fig. 6. Samples were obtained one per second during 60 seconds, each experiment was repeated 30 times. As a baseline, we measured the latency from the *Client* to a *vServer* bypassing all VNFs and the IP tunnel, getting on average 0.859 ms with very small variation. The latency for the complete Multi-SFC using either VXLAN or GRE VNF tunnels reached on average 4.1 ms, whereas using the IPSec VNF tunnel the average of the latency reached 4.23 ms – as IPSec encrypts packets the higher latency was expected. Overall, the increase of the latency is caused by messages passing through all the Multi-SFC VNFs, classifiers, and tunnels (all running on OpenStack).

V. Conclusion

In this work we proposed the Multi-SFC architecture for the composition of SFCs on multiple clouds of multiple administrative domains orchestrated by multiple NFV platforms. The Multi-SFC architecture is compliant with the ETSI NFV-MANO standard. The Multi-SFC connects segments each on a specific cloud/domain/platform using tunnels, which are implemented as VNFs. A proof-of-concept prototype was implemented, experimental results show that the Multi-SFC presents low latency and sustains a satisfactory goodput.

Future work includes the design of strategies to allow efficient resource allocation and Multi-SFC placement, elasticity and migration. The use of NSH for steering traffic between multiple administrative domains should also be investigated. The interconnection of multiple domains using federations is also a promising future work.

References

[1] B. Yi, X. Wang, K. Li, S. k. Das, and M. Huang, "A comprehensive survey of network function virtualization," *Computer Networks*, vol. 133, pp. 212–262, 2018.

[2] N. F. S. de Sousa, D. A. L. Perez, R. V. Rosa, M. A. Santos, and C. E. Rothenberg, "Network service orchestration: A survey," *Computer Communications*, vol. 142-143, pp. 69–94, 2019.

[3] J. Quittek, P. Bauskar, T. BenMeriem, A. Bennett, M. Besson, and et al, "Network functions virtualisation (NFV); management and orchestration. GS NFV-MAN 001 v1.1.1," ETSI, Tech. Rep., 2014.

[4] J. Halpern and C. Pignataro, "Service function chaining (SFC) architecture," IETF, RFC 7665, October 2015.

[5] V. F. Garcia, E. P. Duarte, A. Huff, and C. R. dos Santos, "Network service topology: Formalization, taxonomy and the custom specification model," *Computer Networks*, vol. 178, p. 107337, 2020.

[6] Tacker, "Tacker - openstack NFV orchestration," 2020. [Online]. Available: https://wiki.openstack.org/wiki/Tacker

[7] ETSI, "Open source MANO," 2020. [Online]. Available: https://osm.etsi.org/

[8] Ciena, "Blue planet multi-domain service orchestration (MDSO)," 2020. [Online]. Available: https://www.blueplanet.com/products/multi-domain-service-orchestration.html

[9] Cloudify, "Network orchestration & edge networking," 2020. [Online]. Available: https://cloudify.co/

[10] B. Sonkoly, J. Czentye, R. Szabo *et al.*, "Multi-domain service orchestration over networks and clouds: a unified approach," *ACM SIGCOMM Comp. Comm. Review*, vol. 45, no. 4, pp. 377–378, 2015.

[11] ONAP, "Open network automation platform," 2020. [Online]. Available: https://www.onap.org/

[12] M. Ghaznavi, N. Shahriar, S. Kamali, R. Ahmed, and R. Boutaba, "Distributed service function chaining," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, 2017.

[13] F. Fokus and B. Tu, "Open Baton," 2020. [Online]. Available: http://openbaton.github.io/

[14] X. Haitao and A. Mann, "NFV: Report on architecture options to support multiple administrative domains. GR NFV-IFA 028 v3.1.1," ETSI, Tech. Rep., Jan. 2018.

[15] A. Francescon, G. Baggio *et al.*, "X-MANO: An open-source platform for cross–domain management and orchestration," in *2017 IEEE Conference on Network Softwarization (NetSoft)*, Bologna, Italy, Jul. 2017.

[16] H. Gunleifsen, T. Kemmerich, and V. Gkioulos, "Dynamic setup of ipsec vpns in service function chaining," *Computer Networks*, vol. 160, pp. 77 – 91, 2019.

[17] J. F. Riera, J. Batallé, J. Bonnet, M. Días, M. McGrath *et al.*, "Tenor: Steps towards an orchestration platform for multi-PoP NFV deployment," in *NetSoft Conf. and Workshops (NetSoft)*, 2016, pp. 243–250.

[18] Linux Foundation, "OPNFV," 2020. [Online]. Available: https://www.opnfv.org/

[19] K. D. Joshi and K. Kataoka, "psmart: A lightweight, privacy-aware service function chain orchestration in multi-domain nfv/sdn," *Computer Networks*, vol. 178, p. 107295, 2020.

[20] X. Zhong *et al.*, "Cost-aware service function chaining with reliability guarantees in nfv-enabled inter-dc network," in *IFIP/IEEE Symp. Integrated Network and Service Management (IM)*, 2019, pp. 304–311.

[21] A. Sgambelluri, F. Tusa *et al.*, "Orchestration of network services across multiple operators: The 5G exchange prototype," in *European Conf. Networks and Communications (EuCNC)*, 2017, pp. 1–5.

[22] R. Guerzoni, D. Perez-Caparros *et al.*, "Multi-domain orchestration and management of software defined infrastructures: A bottom-up approach," in *European Conf. Netw. and Comm. (EuCNC)*, Jun. 2016.

[23] OpenStack, "OpenStack - open source software for creating private and public clouds," 2020. [Online]. Available: https://www.openstack.org/

[24] A. Huff, G. Venancio *et al.*, "A holistic approach to define service chains using Click-on-OSv on different NFV platforms," in *IEEE Global Communications Conference (GLOBECOM)*, Dec. 2018.

[25] A. Panda, S. Han *et al.*, "Netbricks: Taking the V out of NFV," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016, pp. 203–216.