Performance of Service Function Chaining on the OpenStack Cloud Platform

Davide Borsatti, Gianluca Davoli, Walter Cerroni, Chiara Contoli, Franco Callegati
University of Bologna, Italy
Email: {davide.borsatti, gianluca.davoli, walter.cerroni, chiara.contoli, franco.callegati}@unibo.it

***Abstract*—Service Function Chaining (SFC) is considered as one of the most promising solutions to deploy future end-to-end network services, taking advantage of emerging network technologies such as Software Defined Networking and Network Function Virtualization that foster advanced communication infrastructure programmability. In this paper, we present an experimental study about the OpenStack SFC extension, which enables the well-known cloud platform as an open-source software solution for implementing SFC. We verify the correctness of the traffic steering operations put in place by the SFC extension and measure the performance in terms of network throughput and response time of the related REST API, considering an increasing length of the service chain. Reported results demonstrate that deploying a service chain takes a reasonable amount of time of a few seconds, whereas the throughput performance at the data plane can be heavily affected by packet processing overhead.**

I. Introduction

Telecommunication network infrastructures have undertaken a disruptive evolutionary path, both as whole systems and in their single components. New design, deployment, and management approaches have recently emerged, with software components playing an increasingly relevant role in networking. As a consequence, the way network services are deployed should evolve from traditional models and take advantage of the increasing flexibility and programmability offered by the emerging scenarios. This has a potentially strong impact both on the improvement of the system as a whole, and on the operational costs for network operators and service providers.

In this context, the concept of *Service Function Chaining* (SFC) has been introduced to describe the deployment of composite networked services obtained by a concatenation, i.e., a *chain*, of one or more basic services, or *Service Functions* [1]. Equivalently, a *Service Function Path* (SFP) is defined as the series of service functions that network traffic must traverse in order for a given service to be correctly delivered.

Network Function Virtualization (NFV) is a natural ally of SFC, as it proposes Virtual Network Functions (VNFs) as software-based network service building blocks [2]. The analogy between a VNF and a service function leads to considering those blocks as potential components of a SFP, which can thus be implemented by taking advantage of virtualization technologies and cloud-like scalability features. Also, Software Defined Networking (SDN) principles are typically adopted to enable dynamic traffic steering across the data plane interconnecting service functions [3]. SDN concepts can also be used to enhance efficiency and flexibility of the control and management planes of SFP deployments: in fact, the SDN architecture can be taken as a reference scenario for the definition of the SFC Control Plane, i.e., the Service Plane [4].

A key aspect that must be carefully pondered from both service provider's and Telco operator's perspective is the level of performance offered by SFC deployments, which in most cases are implemented on top of cloud computing software platforms. As a remarkable example, OpenStack is widely considered as one of the most relevant open-source frameworks that could accelerate the adoption of NFV by Telco operators [5]. Although OpenStack was originally conceived as a software platform for Infrastructure-as-a-Service (IaaS) cloud deployments, the evolution of its networking components showed a progressive integration of NFV/SDN-based solutions [6], to the point that it is often adopted as *the* open-source implementation of the ETSI NFV MANO Virtualized Infrastructure Manager (VIM) component [7].

In this paper, we present an experimental study on the performance of OpenStack as a platform for implementing SFC. In particular, we: (i) describe the behavior of the OpenStack SFC extension [8], (ii) verify the correctness of the related traffic steering operations, and (iii) assess the performance in terms of response time and throughput. To the best of our knowledge, this is the first time that a detailed experimental evaluation of the OpenStack SFC extension is reported.

The remainder of the paper is structured as follows. In Section II we mention some significant related contributions. Then, in Section III we describe the OpenStack SFC extension. In Section IV we present the experimental setup we used to carry out the performance evaluation tests, whose results are discussed in Section V. We then present our conclusions in Section VI.

II. Related work

The main idea behind SFC has been conceived as a possible answer to the need for improved and flexible management of middleboxes and service deployment that network operators and service providers have been facing in the past decade [9], [10]. Recently, SFC has become a hot topic in the research community [11], and is part of a standardization initiative by the IETF [1]. SFC makes use of a service-specific overlay that creates the required service topology, possibly spanning multiple technological or administrative domains [12].

Among the several research issues concerning SFC currently under investigation, it is worth to mention service orchestration

[13], physical resources allocation to data plane components [14], trade-off between optimized performance and resource cost in SFP deployments [15], service function overload and failure management [16]. From a theoretical perspective, recent studies include the analysis of the algorithmic complexity of traffic steering in SFC, which is a particular case of the more general waypoint routing problem [17], as well as performance modeling using network calculus concepts [18].

Despite the huge amount of literature available on SFC, to the best of our knowledge there is no previous attempt to evaluate the performance of the OpenStack SFC extension.

III. The OpenStack SFC extension

OpenStack is an open-source software platform consisting in a rich set of services allowing the deployment and management of public and private cloud infrastructures [19]. Taking advantage of different kinds of virtualization technologies, ICT resources such as storage, CPU, RAM, and network are decoupled from a variety of vendor-specific implementations and exposed as abstractions to multiple tenants. OpenStack defines a consistent set of Application Programming Interfaces (APIs) to manage those virtual resources as discrete pools, with which administrators and users can interact directly by means of standard cloud management tools, e.g., RESTful HTTP clients. The OpenStack platform can rely on a quite large set of extensions, including one dedicated to SFC [8], which provides an API to support SFP creation and deployment in Neutron, the well-known OpenStack's network virtualization component.

The documentation of the OpenStack SFC extension defines a service function as a virtual or physical machine that perform a specific network function such as firewall, content cache, packet inspection, or any other function that requires processing of packets in a flow exchanged between two endpoints in the network. The extension allows for the creation of SFPs, it natively supports interaction with Open vSwitch (OvS), it implements a flow classification mechanism (i.e., the ability to classify traffic based on service-level characteristics), and it provides a vendor-neutral API.

The SFC extension defines and deploy a SFP according to a four-step approach. The first element to be instantiated is the *Flow Classifier* (FC), which is needed to classify the incoming traffic based on predefined policies (e.g., header matching rules), in order for the flow to be properly steered through the required set of service functions. In other words, the FC contains the set of matching criteria that will be used to determine whether a specific traffic flow must traverse the associated SFP or not. Those criteria can be specified by various parameters, spanning from data link to transport layer header fields, as well as OpenStack metadata, such as the port ID assigned by Neutron to the source and destination ports.

As a second step, *Port Pairs* (PPs) must be created. A port pair represents a service function instance that includes an ingress and egress port. In other words, it represents a single hop of the SFP, specifying the network ports, as defined by Neutron, attached to a given service function instance. A port pair can be either uni- or bi-directional, depending on how the associated service function can be traversed by the flow. Also, a port pair may have a *weight* associated to it, to be used to perform load balancing over the SFP.

A *Port Pair Group* (PPG), whose definition is the third step in the creation of a SFP, is a collection of one or more port pairs. If a port pair represents entry and exit points of a particular service function instance, a port pair group can be considered as a list of different instances implementing the same service function. The definition of a port pair group enables load balancing for the corresponding service function: in fact, each new traffic flow traversing a given chain will be forwarded to one of the port pairs belonging to the same group according to a sort of weighted round-robin policy, based on the specific weight that was assigned to each port pair at creation time. Another interesting feature is that a port pair group can be configured with a *tap* attribute. The port pairs belonging to this kind of group will not play an active part in the chain, as they will simply receive a copy of the flow traversing the SFP, without the need of forwarding it to the next hop. This functionality can be useful for those VNFs, such as a packet analyzer, that do not perform any action on the traffic passing though the chain but that just need to receive information about the incoming flows.

Finally, a *Port Chain* (PC) is instantiated as a binding between one or more flow classifiers and an ordered list of port pair groups, thus defining and implementing the actual SFP. All incoming traffic flows matching the rules of the flow classifier(s) specified in the port chain will have to traverse a port pair of each port pair group associated to the port chain, according to the specified order. A port chain may be uni- or bi-directional as well. In the former case, the chain will only be traversed by flows matching the criteria specified in the flow classifiers, whereas, in the latter case, the chain will be traversed also in the opposite direction by flows matching the "symmetric" version of those matching criteria. Moreover, the SFC extension allows linking together different port chains, creating the so called *Service Function Graph*.

After a SFP (i.e., a port chain) has been defined and deployed, the relevant traffic steering policies are installed in the OpenStack network subsystem by adding suitable OpenFlow rules to the OvS-based virtual switches found in each compute node. In particular, the integration bridge (`br-int`), to which all instances running in a given physical node are connected, and the tunneling bridge (`br-tun`), from which tunnels depart toward other physical nodes, are configured with internal and external traffic steering rules, respectively. In particular, when the SFP involves instances running on multiple compute nodes, correct forwarding operations on the physical network infrastructure require to identify which hop of which SFP each packet is currently traversing. To this purpose, each packet must carry a *Service Path Identifier* (SPI) and a *Service Index* (SI), and then be encapsulated using a suitable data plane transport technology. The current implementation of the OpenStack SFC extension supports either Multi-Protocol Label Switching (MPLS) [20] or Network Service Header

(NSH) [21] encapsulation.

Two different kinds of traffic steering rules are added to the virtual switches by the SFC extension when a new port chain is deployed. First, each compute node includes matching rules that follow the criteria as defined by the corresponding flow classifier. These rules are used to intercept packets transmitted by source nodes or service functions, which are typically unaware of the underlying traffic steering technology being used, and to push the additional headers required by the chosen encapsulation method. Second, additional steering rules are included, with matching conditions that depend on the encapsulation technology and refer to the specific switch ports to which the relevant service functions are attached.

All the interactions with the OpenStack SFC extension can be carried out either by using the native command line utilities, or through its REST API. The latter is a more general approach that enables service chain management and orchestration from third-party applications through a standardized interface. Therefore, in this paper we choose to evaluate the SFC extension response time via the REST API, testing the creation of increasingly long SFPs. We also show that, after the deployment of a SFP, traffic is correctly steered through specific instances that would not otherwise be crossed with traditional forwarding.

IV. Experimental setup

To measure the performance of the SFC extension, we installed OpenStack (Rocky release via Devstack) on a test bed consisting of 7 bare-metal servers from the CloudLab facilities [22], as shown in Figure 1. Out of those servers, 4 are used as OpenStack compute nodes (`contr01`, `comp02`, `comp03`, and `comp04`), including one acting as a controller (`contr01`). These nodes are physically connected through separate management and data networks. Out of the remaining servers, 2 of them (`ovs01` and `ovs02`) host instances of OvS, thus implementing an SDN-enabled data network[1], while the last one (`ext`) is used as access gateway to reach the instances. Multi-tenant traffic isolation over the data network is operated via VXLAN tunneling. Since the operating system kernel does not support NSH, we adopted MPLS for SFC encapsulation [20]. The packet overhead introduced by VXLAN tunnels plus MPLS encapsulation is such that we had to reduce the MTU of all instance interfaces to 1418 bytes. All physical servers are equipped with 10 Gigabit Ethernet interfaces.

Virtual machines and networks were instantiated in the OpenStack cluster, yielding the logical network topology shown in Figure 2. With the chosen topology, instances representing endpoints (i.e., customer nodes) reside on the `customerVxlan` network, which is also shared with service function instances. The latter are also connected to a dedicated `internalVxlan` network, used for intermediate

[1]Although in this paper we do not take advantage of the SDN capabilities of the physical data network and rely only on the SFC extension to the OpenStack networking service, we plan to use the same test bed setup in the near future to investigate a possible integration of the SFC extension with an external SDN controller in charge of programming the physical network

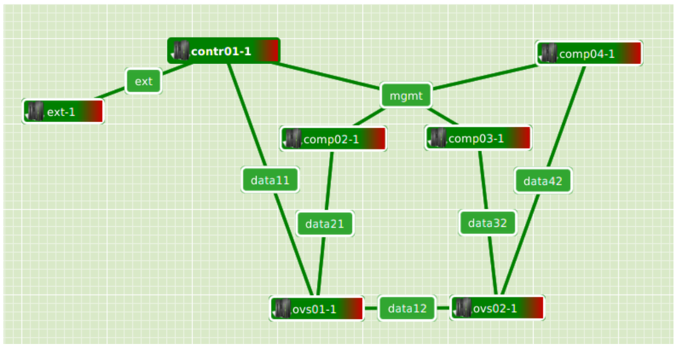

Fig. 1. Physical setup of the OpenStack test bed deployed in CloudLab, as displayed by the jFed experiment management tool.

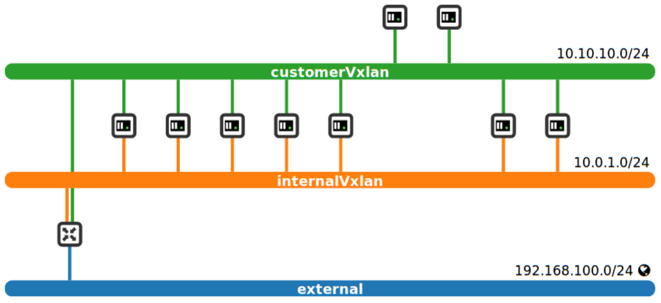

Fig. 2. Virtual network topology and deployed instances.

communication along the SFP. Having two distinct networks in the SFC test bed enables the emulation of realistic scenarios where a single service function may need to have interfaces connected to two separate networks, e.g., as in the case of WAN Accelerator or NATs. While this topology opens the possibility to create SFPs with asymmetric connectivity, the actual direction of the traffic crossing a given service functions can still be configured when creating the corresponding port pair. The placement of the VNF instances over the four OpenStack compute nodes is depicted in Figure 3, where each block represents a physical node and the circles inside it symbolize the virtual machines (i.e., VNFs, source and destination) running on it. Since we are interested in evaluating the performance of the traffic steering mechanism implemented by the SFC extension, the deployed VNFs do not apply any specific processing or conditioning action to the traffic flows traversing them, apart from simple packet forwarding.

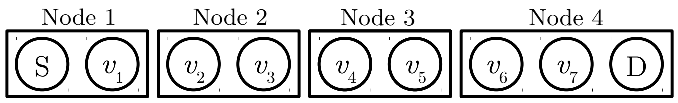

Fig. 3. Instance placement over the physical nodes of the test bed.

In order to carry out the performance assessment tests, a REST client for the SFC extension API was developed, in the form of a Python script. The client operates calls to the mentioned REST API in order to define and deploy a predefined service chain. The details of the requested SFP are provided to the client by a user-defined JSON file, in which the user can specify the endpoints of the SFP as well as flow classifying parameters and the ordered sequence of service functions to be traversed. The structure of the JSON file is as follows:

```
{
  "source": "node_name",
  "destination": "node_name",
  "match_fields": {
          "match_field": "field_value",
          ...
  },
  "chain": [
    {
      "name": "vnf_name",
      "direction": "direction_value"
    },
    ...
  ]
}
```

where: `source` and `destination` represent the endpoint nodes of the service chain, expressed as names of instances known to OpenStack; `match_fields` is a list of key-value pairs identifying matching fields (e.g., `protocol`) and values (e.g., `udp`) to be specified in the flow classifier criteria; `chain` is the ordered list of service functions to be traversed, each specified by the `name` of the corresponding VNF instance; for each service function, the `direction` attribute is used to specify whether the service function must forcibly eject the output traffic on the internal network (`in`), on the customer one (`out`), on the same one it received it from (`same`), or whether the VNF instance must simply receive a copy of the flow (`tap`).

After parsing the JSON file and retrieving the service chain specifications, the SFC client executes the four steps of definition and deployment of a SFP that were described in Section III. The sequence of operations carried out by the client and the corresponding timings are sketched in Figure 4, considering the general case of the complete deployment of a single port chain, including m flow classifiers, M port pairs, and $N \leq M$ port pair groups.

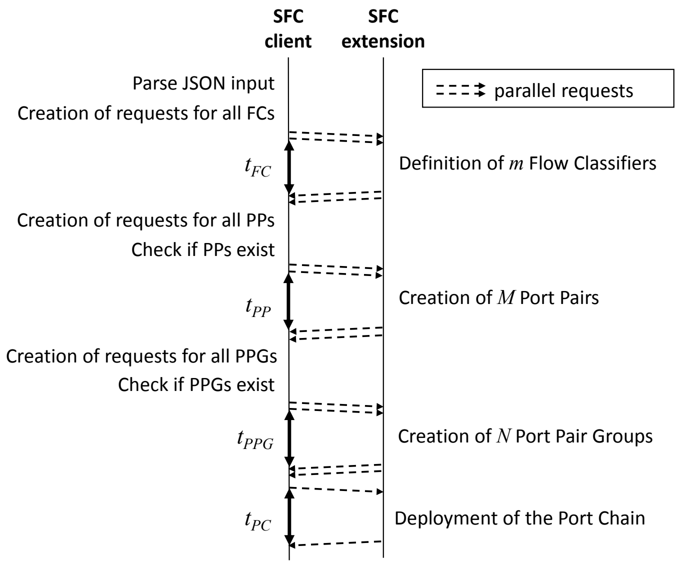
SFC client
SFC extension
Parse JSON input
Creation of requests for all FCs
parallel requests
t_{FC}
Definition of m Flow Classifiers
Creation of requests for all PPs
Check if PPs exist
t_{PP}
Creation of M Port Pairs
Creation of requests for all PPGs
Check if PPGs exist
t_{PPG}
Creation of N Port Pair Groups
t_{PC}
Deployment of the Port Chain

Fig. 4. Execution flow and related timings of the SFC REST client application for a single port chain deployment.

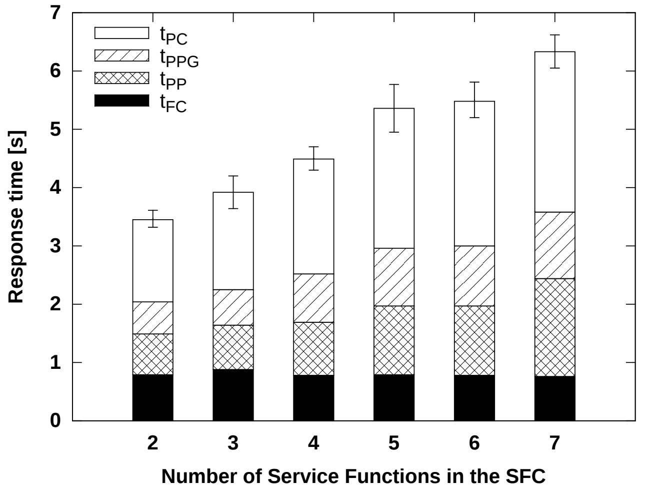

Fig. 5. Average response time (with 95% confidence intervals) of the SFC extension REST API as a function of the length of the SFP. The histogram shows the contribution of each of the four steps required to deploy a SFP.

V. Experimental results

To evaluate the response time of the SFC extension REST API, we timed the execution of each of the four requests listed in Figure 4 needed to deploy a single SFP. We measured the response time at the client side for an increasing number N of traversed service functions, with a single flow classifier ($m = 1$) and a single port pair per port pair group ($M = N$). We considered each of the four steps separately, so as to be able to determine how each operation scales with the chain length. We repeated the tests ten times for statistical significance.

Figure 5 shows the average response time for each of the four REST endpoints, as well as the average total time needed to define and deploy a SFP via the REST interface provided by the SFC extension. Our experiments show that a few seconds are needed to deploy a SFP, with a linear increase of approximately half a second for each service function added to the chain. The timing of client-side operations performed before and after each request are not shown here because their duration is negligible with respect to the SFC extension response time. We verified experimentally that the REST API of the SFC extension responds only after executing the request, and not immediately after receiving it as an acknowledgement, therefore the measured time includes the time needed for the SFP to be actually deployed.

The time needed to define the flow classifier, t_{FC}, appears to be unaffected by the increase in SFP length. This is reasonable, not only because we assumed $m = 1$, but also because to define a flow classifying policy the OpenStack controller does not need to interact with the compute nodes involved in the SFP yet. More precisely, for the first three steps the controller must only update its internal database with the information received from the SFC client. In fact, it is possible to see that t_{PP} and t_{PPG} do increase with N, but not as much as t_{PC}. This can be explained by the increasing number of entries that the controller has to introduce inside the database when creating port pairs and port pair groups. On the other hand, the step which is most affected by the increase in SFP length is the actual deployment of the port chain (t_{PC}). This is reasonable too, as the deployment of the port chain requires the controller to install flow rules on virtual switches residing in all physical compute nodes involved in the SFP.

In order to prove the correctness of the traffic steering operated by the SFC extension, a proof-of-concept test was carried out. We defined the following 3-hop SFPs and deployed them:

$$\text{SFP}_1 : S \rightarrow v_1 \rightarrow v_4 \rightarrow v_7 \rightarrow D \quad (1)$$

$$\text{SFP}_2 : S \rightarrow v_2 \rightarrow v_4 \rightarrow v_6 \rightarrow D \quad (2)$$

$$\text{SFP}_3 : S \rightarrow v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow D \quad (3)$$

An `iperf3` session was launched between the two endpoints for each SFP using different UDP port numbers. The first session (1 Mbps, over SFP_3) lasted for 30 seconds, the second one (2 Mbps, over SFP_2) for 60 seconds, and the third one (3 Mbps, over SFP_1) for 90 seconds.

The throughput measured at the egress port of some of the involved service functions is shown in Figure 6. v_4, which is crossed by all chains, sustains the combined traffic of the three SFPs. When the `iperf3` session over SFP_3 ends, i.e. when the throughput measured at v_3 goes to zero, the remaining active traffic keeps traversing v_4 as well as the other service functions of SFP_1 and SFP_2. In the last 30 seconds, only traffic over SFP_1 is still active, therefore the overall traffic crossing v_4 is coincident with the traffic crossing v_1. This outcome proves the correct deployment of the requested SFPs in our test bed.

We then carried out another set of experiments to evaluate the impact of the introduction of a service chain of increasing length on the TCP throughput measured between two endpoints. Table I reports the list of SFPs that were deployed to perform the TCP throughput test, where chain C_i includes i service functions. The throughput measured for the first chain C_0 is used as a reference, as it is the maximum achievable value obtained from source S to destination D without any VNF in between. Then the order of the VNFs chosen for the other chains is not random. From C_1 to C_4 the chain length is increased by adding one VNF from each physical node according to the placement shown in Figure 3, starting from the nodes where S and D reside. For the last two chains, the order of the two additional VNFs is chosen as to maximize the number of transitions through different compute nodes over the physical network. This can be considered as the worst-case scenario in terms of throughput.

The throughput values of TCP sessions generated with `iperf3` over the chains in Table I are reported in Figure 7.

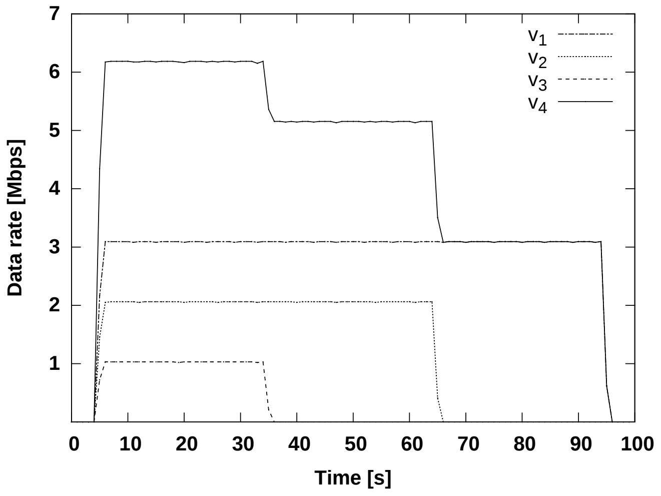

Fig. 6. Throughput measured at the egress interface of some relevant service function instances.

TABLE I
LIST OF SERVICE CHAINS USED FOR TCP THROUGHPUT MEASUREMENTS

Chain hop sequence
$C_0 = S \rightarrow D$
$C_1 = S \rightarrow v_1 \rightarrow D$
$C_2 = S \rightarrow v_1 \rightarrow v_6 \rightarrow D$
$C_3 = S \rightarrow v_1 \rightarrow v_2 \rightarrow v_6 \rightarrow D$
$C_4 = S \rightarrow v_1 \rightarrow v_2 \rightarrow v_4 \rightarrow v_6 \rightarrow D$
$C_5 = S \rightarrow v_1 \rightarrow v_2 \rightarrow v_4 \rightarrow v_3 \rightarrow v_6 \rightarrow D$
$C_6 = S \rightarrow v_1 \rightarrow v_2 \rightarrow v_4 \rightarrow v_3 \rightarrow v_5 \rightarrow v_6 \rightarrow D$

While the maximum achievable throughput for C_0 is above 8 Gbps, as expected the use of SFC affects the performance of the network, reducing the throughput for each VNF added to the chain. Interestingly, a significant penalty is caused by the transition through different physical nodes, as in the case of chains C_2 to C_6. So the location of a VNF can drastically change the throughput achievable between source and destination. This is also easy to see from the results presented in Table II, which reports the throughput measured with just one VNF between S and D and placed in three different positions: co-located with S, co-located with D, or in a separate physical node. From these results it is clear that the transit through the physical network, which involves MPLS labelling and VXLAN encapsulation, as well as forwarding through the OvS switches implementing the SDN-enabled data network, introduces a significant processing overhead that limits the throughput between the endpoints of the chain. The adoption of packet processing acceleration technologies could provide a possible solution to this performance problem [23].

Lastly, a brief consideration is due on the port pair group tap attribute mentioned in section III, as implemented by the SFC extension. After some tests we found that this functionality actually works only if the VNF that is supposed to act as a tap is running on the same physical node as the previous hop of the chain. In other words, if the mirrored flow has to

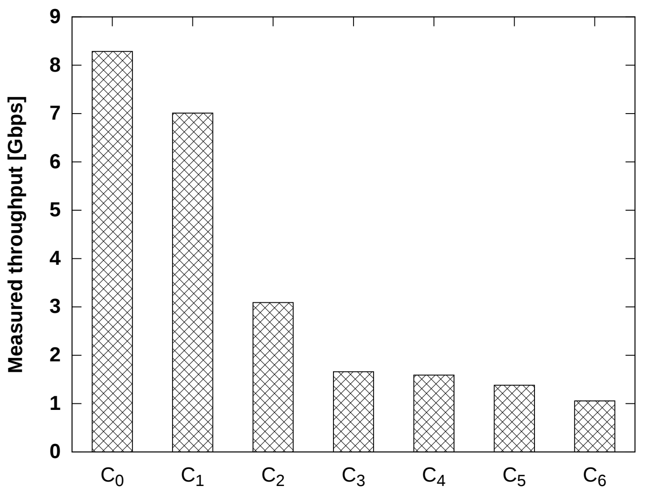

Fig. 7. TCP throughput measured with increasing chain length (95% confidence intervals are not shown due to their negligible width).

TABLE II
SINGLE VNF PERFORMANCE FOR DIFFERENT LOCATIONS

VNF location	Same as S	Same as D	Other node
Throughput (Gbps)	7.01	3.39	1.99

go through the OpenStack tunneling bridge, then it will not reach the tap VNF. This is probably caused by a bug in the SFC extension version we tested, which does not install the necessary OpenFlow rules in the tunneling bridge to properly mirror the flow.

VI. CONCLUSION

In this paper we presented the OpenStack SFC extension, which enables the well-known IaaS cloud platform as an open-source software solution for implementing the service function chaining approach. We carried out an experimental study on the CloudLab platform to verify the correctness of the traffic steering operations put in place by the SFC extension, and reported a first attempt to measure the performance in terms of network throughput and response time of the related REST API, considering an increasing length of the service chain. While the time required to deploy a SFP at the management/control plane is reasonably kept in the order of a few seconds, the throughput performance at the data plane is heavily affected by the overhead caused on packet processing by the tunneling and encapsulation technologies needed to properly steer the traffic along the service chain. In the future, we plan to test how packet processing acceleration techniques can improve the data plane performance, as well as investigate a possible integration of the SFC extension with external SDN controllers for enhanced physical network programmability.

REFERENCES

[1] J. M. Halpern and C. Pignataro, "Service Function Chaining (SFC) Architecture," RFC 7665, Oct. 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7665.txt

[2] "Network Functions Virtualisation (NFV); Architectural Framework," The European Telecommunications Standards Institute (ETSI), October 2013. [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/nfv

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

[4] G. Davoli, W. Cerroni, C. Contoli, F. Foresta, and F. Callegati, "Implementation of service function chaining control plane through openflow," in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov. 2017.

[5] Accelerating NFV delivery with OpenStack. [Online]. Available: https://www.openstack.org/assets/telecoms-and-nfv/OpenStack-Foundation-NFV-Report.pdf

[6] F. Foresta, W. Cerroni, L. Foschini, G. Davoli, C. Contoli, A. Corradi, and F. Callegati, "Improving OpenStack networking: Advantages and performance of native SDN integration," in *2018 IEEE International Conference on Communications (ICC)*, May 2018.

[7] "Network Functions Virtualisation (NFV); Management and Orchestration," The European Telecommunications Standards Institute (ETSI), December 2014. [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/nfv

[8] Service function chaining extension for openstack networking. [Online]. Available: https://docs.openstack.org/networking-sfc/latest/

[9] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," in *ACM SIGCOMM 2012 Conference*. New York, NY, USA: ACM, 2012.

[10] W. John, K. Pentikousis, G. Agapiou, E. Jacob, M. Kind, A. Manzalini, F. Risso, D. Staessens, R. Steinert, and C. Meirosu, "Research directions in network service chaining," in *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, Nov. 2013.

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

[12] H. Chen, S. Xu, X. Wang, Y. Zhao, K. Li, Y. Wang, W. Wang, and L. M. Li, "Towards optimal outsourcing of service function chain across multiple clouds," in *2016 IEEE International Conference on Communications (ICC)*, May 2016.

[13] A. M. Medhat, G. A. Carella, M. Pauls, M. Monachesi, M. Corici, and T. Magedanz, "Resilient orchestration of Service Functions Chains in a NFV environment," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov. 2016.

[14] M. T. Beck, J. F. Botero, and K. Samelin, "Resilient allocation of Service Function Chains," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov. 2016.

[15] T. Soenen, S. Sahhaf, W. Tavernier, P. Skldstrm, D. Colle, and M. Pickavet, "A model to select the right infrastructure abstraction for Service Function Chaining," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov. 2016.

[16] J. Lee, H. Ko, D. Suh, S. Jang, and S. Pack, "Overload and failure management in service function chaining," in *2017 IEEE Conference on Network Softwarization (NetSoft)*, July 2017.

[17] S. A. Amiri, K.-T. Foerster, R. Jacob, and S. Schmid, "Charting the algorithmic complexity of waypoint routing," *SIGCOMM Comput. Commun. Rev.*, vol. 48, no. 1, pp. 42–48, Apr. 2018.

[18] Q. Duan, "Modeling and performance analysis for service function chaining in the sdn/nfv architecture," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, June 2018.

[19] OpenStack. [Online]. Available: https://www.openstack.org

[20] A. Farrel, S. Bryant, and J. Drake, "An MPLS-based forwarding plane for Service Function Chaining," IETF, Internet-Draft, Aug. 2018. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-mpls-sfc-02

[21] P. Quinn, U. Elzur, and C. Pignataro, "Network Service Header (NSH)," RFC 8300, Jan. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8300.txt

[22] Cloudlab. [Online]. Available: https://www.cloudlab.us

[23] M. Kourtis, G. Xilouris, V. Riccobene, M. J. McGrath, G. Petralia, H. Koumaras, G. Gardikis, and F. Liberal, "Enhancing VNF performance by exploiting SR-IOV and DPDK packet processing acceleration," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, Nov. 2015.