Orchestrating Service Function Chaining in Cloud Environments

Ahmed M. Medhat, Tran Quang Thanh, Stefan Covaci
Technical University Berlin
Berlin, Germany

Giuseppe Carella, Thomas Magedanz
Fraunhofer FOKUS
Berlin, Germany

***Abstract*—The rapid emergence of Software Defined Networks (SDN) and Network Function Virtualization (NFV) concepts is enabling the innovation of cloud infrastructures for supporting real time services and applications. Interconnecting Service Functions (SFs) in a specific ordered way to support applications' requirements is defining the concept of Service Function Chaining (SFC). SFC uses those evolved technologies, SDN and NFV, to provide SFCs in cloud environments in a rapid and cost-effective way. This paper proposes an extension to the current ETSI NFV Management and Orchestration (MANO) Architecture in order to improve the overall service quality of SFC in cloud environments and securely control their lifecycle. Specifically, are proposed a SFC Orchestrator and a token-based Access Control for providing management and provisioning of SFCs, as well as efficient usage of resources, enhancing the overall network Quality of Service (QoS). In addition, it is presented an open source implementation based on the the Open Baton project, and its integration with existing open source tools. Finally, an evaluation use case is introduced to validate the proposed work.**

Keywords—Service Function Chaining; Orchestration; NFV; SDN; MANO

I. INTRODUCTION

Management of network resources and services based on user and network requirements are critical items into operators' business and operational system. Increasing the number of services/applications with the always growing users' number led to excess traffic rendering network resources management and services even more challenging. Service Function Chaining (SFC) is a necessary solution to classify flows, manage network traffic and provision appropriate policies on the network path between users and services. SFC is defined as an ordered series of service functions (SFs) chained to support the traffic of a specific service/application. SFC benefits from the principles of Software Defined Networking (SDN) [1]. SDN isolates the control plane from the data plane and provides suitable programming abstractions which are needed in SFC to provide dynamic control chains by simplifying traffic steering across SFs. Network Function Virtualization (NFV) [2] is a new telco initiative enabling the virtualization of network functions. It is adopted in SFC for simplifying the deployment and orchestration of SFs.

SFC is addressed by the IETF SFC working group [3] and the Open Network Foundation (ONF) [4], determining the specifications for the SFC architecture. SFC has attracted a lot of attention within the community of researchers as well as among operators and network equipment vendors (e.g., Juniper [5], QOSMOS [6] and Huawei [7]). A large number of open source tools enabling SFC are also available for building SFC prototypes. Notable examples are OpenDaylight (ODL) [8], OPNFV [9], OpenContrail [10] and OpenStack's Neutron/Service Insertion and Chaining [11]. The previous research works show a lot of progresses in providing implementations and architectures applying the concept of SFC in its new paradigm based only on SDN technology (such as STEERING [12], SIMPLIFYING [13], and Context-aware SFC [14]). Moreover, there are other research works making use of NFV technologies together with SDN presenting implementations and architectures of SDN/NFV-based SFC which this paper is concerning about.

The previous research works related to cloud-based SFC architectures have some limitations in terms of orchestration, Quality of Service (QoS) provisioning, scalability, flexibility and traffic engineering. MIDAS architecture [15] is not using an NFV orchestrator, does not support QoS enforcement and has low scalability. ESCAPE framework [16] does not support QoS enforcement and traffic engineered SFC system. Policy-based SFC architecture model [17] does not support a load balancing system and has not a SF placement engine. Cloud-based SFC framework [18] is suffering from low scalability and flexibility and it does not support QoS enforcement, SFC traffic engineering and SF placement engine. Cloud4NFV platform [19] does not concern about SF load balancing and does not support QoS enforcement and SF placement engine.

In order to solve these limitations, this paper is introducing an ETSI NFV [28] based SFC architecture extended with a proposed SFC Orchestrator (SFCO) module as a main contribution of this work. Moreover, this paper proposes a token-based access control to securely control the NFV environment. The proposed ETSI NFV-based SFC framework abstracts the low level details of SFC management and deployment allowing adaption of SFC paths to the QoS requirements, while maintaining the traffic loads and transparently scale if required. The proposed SFC Orchestrator component enables SFC provisioning in cloud environments using the capabilities of the NFV Orchestrator and the SDN Controller which is assumed to support SFC data path instantiation. Furthermore, the proposed SFC Orchestrator is filling the gaps existing in previous research works. The SFCO enhances the overall network QoS performance by

providing optimized traffic paths for the predefined SFCs and uses the available resources efficiently.

The rest of paper is organized in the following pattern. Section II highlights the typical SFC architecture as defined by the IETF SFC working group [3]. Section III illustrates the proposed ETSI NFV-based SFC architecture including the proposed SFCO module and the proposed security control module as well. Section IV introduces an implementation setup for the proposed architecture to provide its validity and presents a use case for evaluating the proposed work. Finally, the paper concludes in Section V.

II. Typical SFC Architecture

As stated in the IETF SFC specifications [3], a typical SFC architecture is composed of three main layers, namely the management plane, control plane and data plane. The management plane is known as the SFCO which is responsible for the SFC management and SF instances management. The control plane layer is responsible for mapping SFC to a specific Service Function Path (SFP), installing and governing forwarding rules on the data plane and adjusting the SFP based on SF instances status and overlay links [20].

The main functional components of the data plane, as shown in Fig. 1, are classifier, SF, SF Proxy and Service Function Forwarder (SFF). The classifier distinguishes the traffic based on the target application or service (web-browsing, video streaming, voice call…etc) and other predefined conditions. It then forwards the traffic to the determined SFC by adding an SFC header containing a path ID into the flow packet headers. SFC is an ordered set of abstract SFs which must be applied to the traffic's packets chosen as a result of classification. A SF, which is known also as Virtual Network Function (VNF) in the NFV terminology, is a function responsible for applying specific operations on received packets and can be shared among multiple SFPs. An example of a SF could be the Deep Packet Inspection or a Firewall. A single SF can be instantiated by multiple instances for scalability reasons, and those instances could also be placed in different locations.

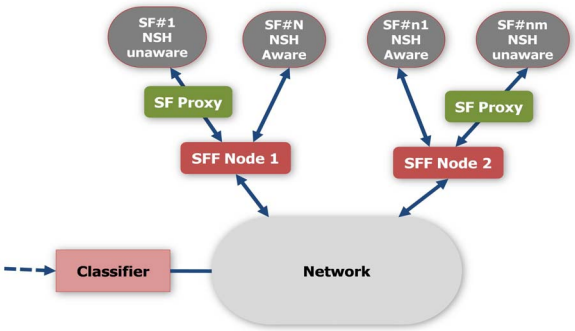

Figure 1. A typical SFC architecture

A SFF is responsible for forwarding the traffic to SFs or other SFFs according to the information carried in the packet's encapsulation based on the assigned SFP. The IETF SFC working group does not impose a specific SFF, but the SFF must be able to insert SFC special header, which is called Network Service Header (NSH) [21], into the traffic packets. SFP is the actual path (the exact SFFs/SFs) that packets traverse. A SF proxy may become required between SFF and SFs as most SFs do not understand the SFC packet headers. This SF proxy de-encapsulates the SFC headers from the packets when the traffic goes into the SF and encapsulates it again when it comes out from the SF to the SFF [3].

ONF proposed also another architecture model for L4-L7 SFC architecture [4]. The ONF SFC system is mainly based on the IETF SFC specification [3] incorporating an extended OpenFlow switch version supporting NSH to be the SFF in the proposed SFC architecture. The ONF SFC also provides a component called SFC Orchestrator but it has only the basic functionalities of SFC management such as creating/deleting SFC and configuring the classification rules based on the user requirements.

III. ETSI NFV-based SFC Architecture

This section presents the proposed extension to the ETSI NFV architecture in order to support SFC in cloud environments and discusses the proposed SFCO functionalities in more details.

A. *Proposed SFC Architecture*

The proposed SFC Architecture, shown in Fig. 2, is based on the ETSI NFV MANO Architecture [29]. The main components of the ETSI NFV architecture are NFV Orchestrator (NFVO), VNF Manager (VNFM), Virtualized Infrastructure Manager (VIM), and the NFV Infrastructure (NFVI) including the SDN Controller. It is assumed that the SDN Controller supports SFC data plane functionality.

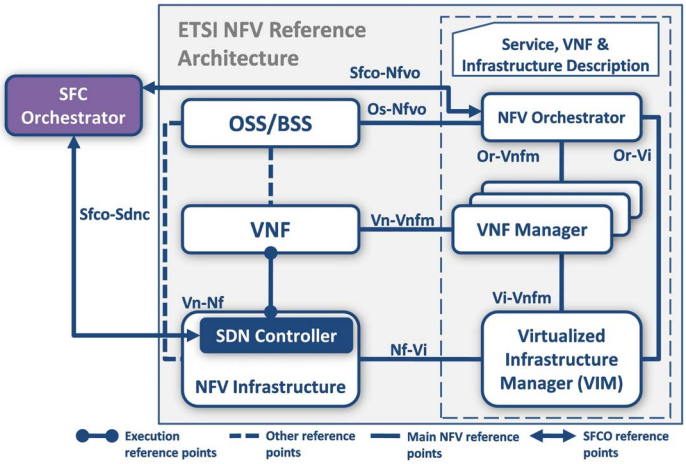

Figure 2. ETSI NFV-based SFC Architecture

The NFVO, in general, is responsible for the Network Service lifecycle, instantiation, scaling, updating and terminating. In the SFC scenario, the NFVO is responsible for the SFC instance orchestration across multiple Point of Presence (PoP) and its lifecycle management. The VNFM is responsible for each VNF instance (SF in SFC scenario) lifecycle management. The VIM is responsible for controlling and managing the NFVI compute, storage and network resources. The SDN controller is used to manage the traffic steering by deploying the flow rules onto the switches. Finally, the SFCO component, the main contribution of this work, is

responsible for translating the abstract high level user requests into flow classification rules, and sequences of SF/SFF instances that construct chain paths correlated with the flows. SFCO has three main interfaces with the NFVO, the SDN Controller and the monitoring system.

The proposed SFCO is composed of five main functional blocks and three interfaces as shown in Fig. 3. The main functional blocks are SFC Manager, QoS Performance Monitoring, SFC Catalogue, QoS Manager and SF Instance Placement.

The *SFC Manager* block is the primary functional component inside the SFCO. It provides management and deployment of SFC using the SDN Controller and NFVO. Its functionality includes create/update/delete SFC classifier rules, through the SFC data plane functionality inside the SDN Controller based on the requests coming from the client. Also, it creates/updates/deletes SFC, and SFP through the SFC Service inside the SDN Controller based on requests coming from the user and the information inside the QoS Performance Monitoring block.

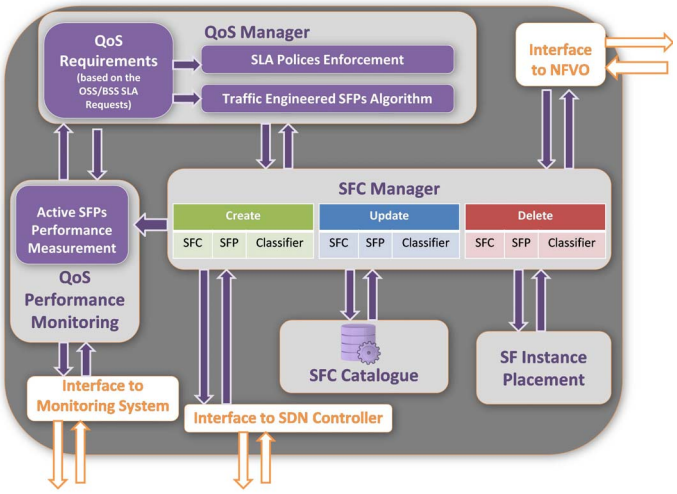

Figure 3. SFCO Functional Blocks Architecture

The *QoS Performance Monitoring* block is responsible for collecting the SFs instances' statistics (CPU & Memory usage for the VM containing this SF) using monitoring system that should have all monitoring information about the SF instances. This functional block is also in charge of collecting the SFFs' statistics (Ports' Transmitted and received bytes) using Open vSwitch database (OVSDB) and OpenFlow plug-in inside the SDN Controller. It also provides the QoS performance measurements of the active SFPs based on calculating the delay and the throughput for each active SFP. This block is considered as a vital component to the network's performance, since it sends an alert to SFC Manager in case of a path collision, an overloaded SF instance or not meeting the QoS requirements for specified application or user. Moreover, this block requests scaling of SF instances or migrating SF instance in case of the reaching network's performance bottlenecks.

The *SFC Catalogue* block represents the database for SFC. Its function is to store all the information related to the configured SFCs, SFPs, SFC Classifier rules, SFFs and SF in a database.

The *QoS Manager* block is responsible for enhancing the network QoS performance using the QoS Requirements list which is based on the OSS/BSS SLA requests for the users and applications) and two main functional sub-blocks: Traffic Engineered SFPs Algorithm and SLA Policies Enforcement.

The *Traffic Engineered SFP Algorithm* sub-block is providing a traffic engineered SFPs based on the network services provider's QoS requirements. This sub-block includes two modules; the first module collects all information related to the active SFPs who do not meet the required QoS and the second module is providing an algorithm to instantiate the optimized SFPs based on the required QoS for each SFC.

The *SLA Manager* functional sub-block is important to enhance the QoS for users and required applications/services through providing a special SLA for each application/service and each user. This SLA is provisioned in the SFFs (OpenVSwitch - OVSs) using the OVSDB Plug-in in SDN Controller through creating queues. Each queue has its minimum and maximum packet rates, and each queue is mapped to special matching traffic conditions assigned by the SLA Manager block.

The *SF Instance Placement* block is responsible for locating the SF instances in the NFVI. Its functionality is to provide an optimized placement algorithm for the new instantiated or migrated SF instances, and then it sends the placement information request to the NFVO for doing the next actions. It utilizes the QoS Performance Monitoring functional block in providing the overloaded SFs instances and SFFs information. Also it has all information about the network topology that is used as an input for the SF instances placement algorithm.

The three interfaces in the proposed SFCO are the NFVO interface, the Monitoring System interface and the SDN Controller interface. The NFVO interface is used by the SFC Manager functional block to receive the SFC requests sent by the NFVO using the messaging queue system. This interface is also used by the SF Instance Placement functional block for sending information to the NFVO about the placement and scaling of SFs instances. The Monitoring System interface is utilized by the QoS Performance Monitoring block for requesting and receiving the monitoring information about the available SFs instances. It also communicates with the OVSDB plug-in of the SDN controller in order to receive periodic statistics about the transmitted and received packets per each switch's port. The SDN Controller interface is used by SFC Manager functional block for sending the functional block's requests and receiving information about all configured SFC components, chains and paths. The second functional sub-block which uses this interface is the SLA Policies Enforcement / QoS Manager that sends its queues and QoS requests to the OVSDB plug-in which applies these requests on the switches' ports.

B. SFC Deployment and Configuration

SFC is deployed through the SFCO and its interaction with the NFVO and SDN Controller. Fig. 4 shows the

workflow diagram illustrating how the SFC instantiation is done. Creating the Network Service Descriptor (NSD) at the NFVO is based on VNF Descriptors (VNFDs) which are used for instantiating the SFs instances on the NFVI. The NSD contains multiple VNF Forwarding Graphs (VNFFGs) which represent the SFCs in the ETSI NFV MANO specification [29]. After creation and instantiation of NSD at the NFVO (step 1a), the NS Record (NSR) is sent to the SFCO at step 1b). At step 2a), SFCO receives the NSR and translates the included VNFFGs into Chains. At step 2b), the SFCO sends SFs, SFFs, SFC, SFPs and SFC Classifier requests to the SDN Controller in order to deploy the NFVO's SFC requests. After processing the SFC requests at the SDN controller (step 3), the SDN controller sends an acknowledgment (ACK) to the SFCO at step 4) to insure that the SFC is deployed successfully.

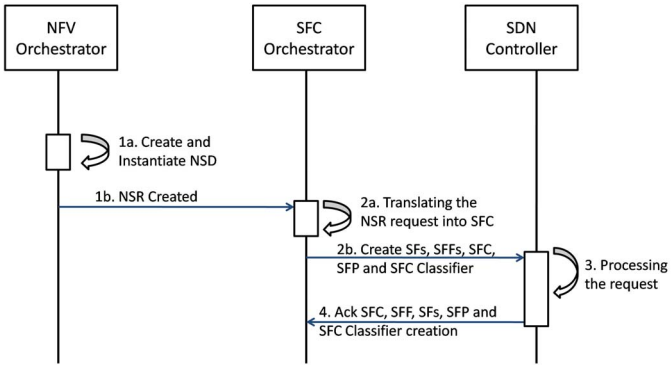

Figure 4. SFC Deployment

C. *Secure NFV Orchestrator*

The NFVO represents the central component in the ETSI MANO domain, therefore it is needed to be protected from malicious or unauthorized accesses. In this work, the token-based access control model is proposed to secure access to NFVO APIs. As a result, consumer applications/services must present an access token when accessing such APIs. The token will be checked and access will be granted or denied by a given access control service. OAuth2-based [26] and Attribute Based Access Control models [27] are getting a lot of interest recently and are adopted in our approach. The former is the evolving standard solution to secure API access. OAuth allows users (resource owner) to grant third-party applications (client) accessing user data (resource server) without sharing credential (e.g. password). The latter is developed by OASIS to standardize the authorization decisions in enterprise applications. It defines XACML (a XML-based policy language), a request / response scheme and access control architecture. Depending on security requirements, either "coarse-grained" or "fine-grained" access control layer of protection is taken into account.

- OAuth2 (Coarse-grained): Only the OAuth2 access token is checked and if it is valid the authorized decision is granted.
- OAuth2 + ABAC (Fine-grained): Both the token and other attributes of the token owners are checked.

Fig. 5 describes a common scenario that using the proposed access control model. First, an end user/consumer application needs to authenticate with the Access Control Service to get an access token and then includes this token in the API request.

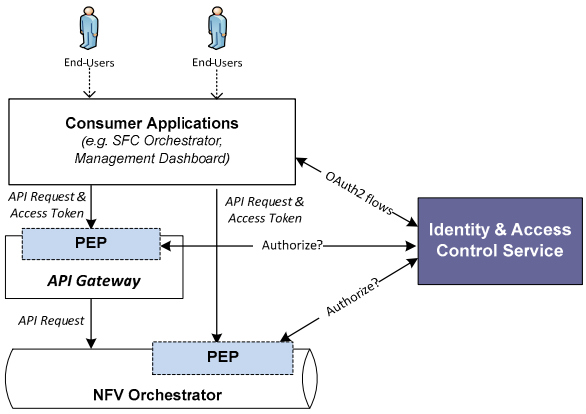

Figure 5. Token-based Access Control

A specific module, PEP (Policy Enforcement Point), which can be part of the NFVO or located in another intermediate security component (e.g. security gateway/proxy), will intercept the API request and check with the access control service to give authorized decision (grant or deny the access) by applying the coarse-grained or fine-grained approach. Flexibility and reusability are several foreseeable benefits when applying such token-based access control approach as access policies are specified and validated separately from where they are enforced.

IV. Validation and Evaluation

This section provides an implementation setup using NFV/SDN Open Source tools to validate the proposed architecture model. Moreover, it presents an evaluation use case in order to show the efficiency of the SFCO.

A. *Implementation Setup*

Fig. 6 depicts the setup of the framework. It is composed of NFVO, VNFM, VIM, SDN Controller and the proposed SFCO. Open Baton [22], developed by Fraunhofer FOKUS and TU Berlin, provides a open source standard-compliant implementation of the ETSI MANO specification [29]. It is implemented in Java, taking the advantage of the Spring Framework [25], and it is used in our implementation setup as implementation of the NFVO and a Generic VNFM. OpenStack [23] implements the VIM and it includes a modified Open vSwitch (OvS) supporting Network Service Header (NSH). This modified OvS correspond to the SFF and the SFC classifier rules are applied on it. ODL [8] is used as SDN Controller in the implementation setup since it performs the SFC data path programmability. The proposed SFCO component is also implemented in Java on top of the Spring Framework.

All these components are integrated together to provide the ETSI NFV based SFC. Open Baton NFVO communicates with the Generic VNFM, SFCO and other components using a messaging queue system. This messaging system is based on RabbitMQ (open source message broker software) that implements the Advanced Message Queuing Protocol

(AMQP). The Generic VNFM makes use of the Element Management System (EMS), which is responsible for executing the VNFM's commands locally in the VNF Container.

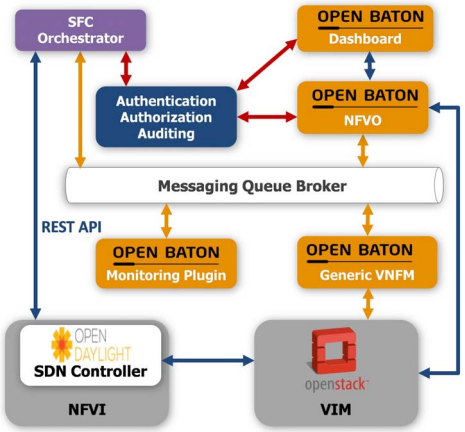

Figure 6. ETSI NFV-based SFC Implementation Setup

The NFVO integrates with a VNFs monitoring system, Zabbix [24]. This integration is done through Open Baton monitoring plug-in which interacts directly with Zabbix and communicates with other architecture's components through the messaging queue broker as discussed previously. The SFC Orchestartor is interacting with the SDN Controller through the Northbound REST API of ODL Controller. The proposed Authentication Authorization Auditing (AAA) module adopted the aforementioned token-based access control approach to provide securely control for the NFVO. Components such as SFCO or the Management Dashboard is required to authenticate first with the AAA to get access tokens and the NFVO authorize its API requests (to grant or deny access) based on such token. This component has been implemented also using the Java Spring Framework and several open source solutions such as WSO2 Identity Server, MySQL and Syslog.

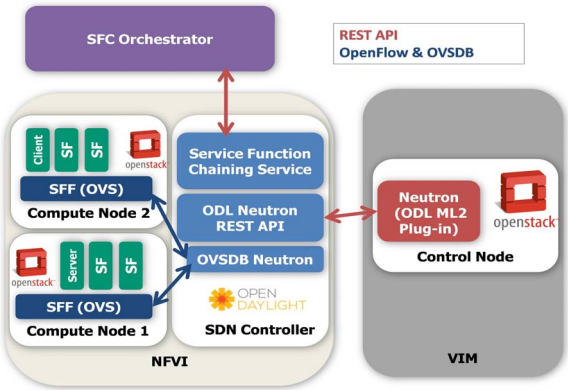

Figure 7. ODL-Controller and Openstack Integration

ODL SDN Controller has an interaction with OpenStack through its Neutron ML2 Plug-in in the northbound of ODL Controller using the REST API protocol as shown in Fig. 7. ODL ML2 plug-in is available in the recent OpenStack's releases that support VxLAN and GRE tunneling protocols between OvSs. These tunneling protocols are used for overlay network creation in order to achieve network virtualization. The OpenStack OvSs (or SFFs in case of SFC) are programmable using OVSDB Neutron application also available in ODL. This OVSDB Neutron application uses OVSDB and OpenFlow for configuration and data path respectively. The OVSDB protocol uses JSON/RPC calls to manipulate a physical or virtual switch that has OVSDB attached to it.

B. Evaluation Use Case

For the evaluation of the proposed framework we selected the Remote Patient Monitoring (RPM) use case where patients' vital health parameters (e.g. heart rate) are collected, stored in the cloud and given access through several sets of APIs. This use case adopted multi-tiers architecture including the user interface (*RPM UI*) and service APIs (*RPM API)* such as Patient Identity and Access Management API (*IDM*) and Electronic Health Record Management API (*EHR*). In addition to that, an API Gateway (API GW) is placed in between to secure access from UI tier to API tier. The SFC is applied to compose the use case architecture: *RPM UI →API GW→RPM API (EHR and IDM)*. These service functions are implemented and deployed using Java programming language and Spring boot framework [25]. As the EHR service is used more frequent than the IDM service, require for low latency in getting the patient monitoring information (e.g. patients heart rate), we assign QoS policies using the SLA Manager block in the SFCO, to use 70% of bandwidth rate as a guaranteed rate for the EHR traffic. On the other side, the IDM service is configured to use 30% of bandwidth rate as a guaranteed rate. As shown in Fig. 8, when both traffics use the same SFF port, they share the bandwidth rate as the specified guaranteed rate. The traffic rates for both services are forced to use the specified bandwidth rate in the SLA (70% and 30%). That results positively in terms of latency of the real time services.

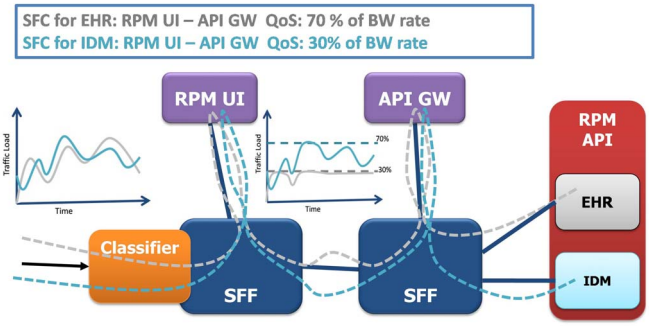

Figure 8: SLA enforcement for SFCs using SLA Manager

Another scenario, when the network is stressed with high traffic, which blocks the ports of SFFs or overloads the SFs' containers (such as Virtual Machines). Fig. 9 shows that both SFCs traffic (EHR and IDM) are sharing RPM UI 1 and API GW 1, which results in overloading of these SFs' containers and congestion of the correlated SFFs' ports and connected links which will consequently affect negatively on the QoS in terms of high latency.

The usage of the traffic engineered SFP Algorithm agent that is inside the QoS Manager block of SFCO, enables

dynamic update of the SFP for the IDM chain to use RPM UI 2 and API GW 2 in order to optimize the SFPs for the two chains. As shown in Fig. 10, updating the SFP for the SFC dedicated to the IDM service decreases the load on RPM UI 1 and API GW 1. That also will result in providing lower delay since the links between the SFFs are unblocked.

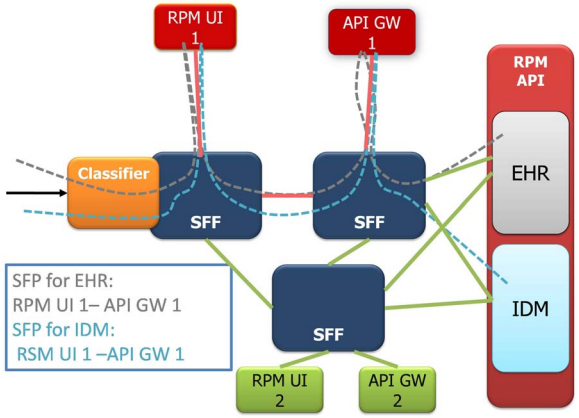

Figure 9. Overloaded SFs and Congested SFFs' ports

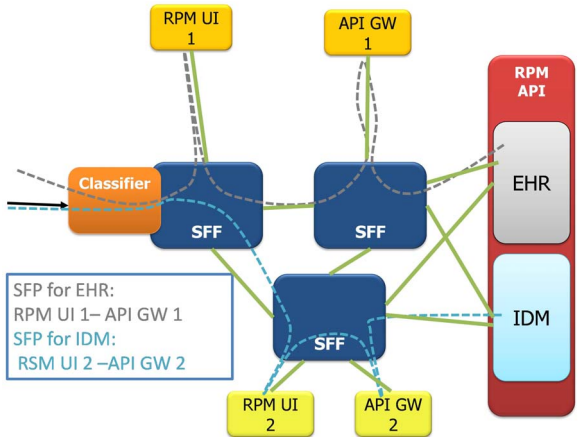

Figure 10. Optimized SFP using the Traffic Engineered SFP Agent

V. Summary

The delivery of end-to-end services requires various SFs to be provisioned in a SFC. Provisioning SFC in virtual environments is addressed through this paper by extending the ETSI NFV MANO architecture with a SFCO component has the capability of provision SFC in cloud environment and providing optimized paths for the chains with an efficient usage of resources. The paper also proposed a token-based access control to secure the NFV Orchestrator. At the end the paper presents an implementation setup for the proposed framework using open source solutions. Furthermore, several our developed open source components are introduced and an evaluation use case is discussed in order to show how efficient is the SFCO proposed solution.

Acknowledgment

This Project has been funded with support of the European Commission. This publication reflects the view only of the authors, and the Commission can't be held responsible for any use which may be made of the information contained therein.

References

[1] D. Kreutz, F. Ramos, et. al., "Software-Defined Networking: A Comprehensive Survey", in Proceedings of the IEEE, vol. 103, no. 1,p. 63, 2015.

[2] NFV White Paper, "Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action.", Issue 1, Oct 2012.

[3] J. Halpern, and C. Pignataro, " Service Function Chaining (SFC) Architecture", IETF SFC WG, March, 2015.

[4] ONF White Paper, " L4-L7 Service Function Chaining Solution Architecture".https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/L4-L7_Service_Function_Chaining_Solution_Architecture.pdf

[5] Juniper Networks, "Decoding Software Defined Networking (SDN)," 2013. http://www.juniper.net/us/en/local/pdf/whitepapers/2000510-en.pdf

[6] QOSMOS, "Enhanced Service Chaining for Service Providers". http://www.qosmos.com/sdn/service-chaining/

[7] Huawei, "Enabling Agile Service Chaining with Service Based Routing," 2013.http://www.huawei.com/ilink/en/download/HW_308622

[8] OpenDaylight.https://opendaylight.org/

[9] OPNFV.https://opnfv.org/

[10] OpenContrail.www.opencontrail.com

[11] OpenStack."Neutron/ServiceInsertionAndChaining", https://wiki.openstack.org/wiki/Neutron/ServiceInsertionAndChaining

[12] Y. Zhang, et al. "Steering: A software-defined networking for inline service chaining." *21st IEEE International Conference on Network Protocols (ICNP)*, 2013.

[13] Z. Qazi, et al. "SIMPLE-fying middlebox policy enforcement using SDN," *ACM SIGCOMM Computer Communication Review*. Vol. 43. No. 4, 2013.

[14] B. Martini, et al. "SDN controller for context-aware data delivery in dynamic service chaining," *1st IEEE Conference on Network Softwarization (NetSoft)*, 2015.

[15] A. Abujoda, and P. Papadimitriou. "MIDAS: Middlebox Discovery and Selection for On-Path Flow Processing," *IEEE COMSNETS, Bangalore,* 2015.

[16] A. Csoma, et al. "ESCAPE: Extensible service chain prototyping environment using mininet, click, netconf and pox." *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014.

[17] K. Giotis, Y. Kryftis, and V. Maglaris. "Policy-based orchestration of NFV services in Software-Defined Networks." *1st IEEE Conference on Network Softwarization (NetSoft)*, 2015.

[18] F. Callegati, et al. "Dynamic chaining of Virtual Network Functions in cloud-based edge networks." *1st IEEE Conference on Network Softwarization (NetSoft),* 2015.

[19] J. Soares, et al. "Toward a telco cloud environment for service functions." *IEEE Communications Magazine,* 53.2 (2015): 98-106.

[20] H. Li, Q. Wu, et. al., "Service Function Chaining (SFC) control plane components and requirements", *IETF SFC WG*, June 2015.

[21] P. Quinn and J. Guichard, "Service Function Chaining: Creating a Service Plane via Network Service Headers", in *IEEE Computer Journal, vol.47, issue 11, pp 38-44,* Nov. 2014.

[22] OpenBaton. http://openbaton.github.io/

[23] OpenStack. http://www.openstack.org/

[24] Zabbix. http://www.zabbix.com/

[25] Spring Boot framework, http://projects.spring.io/spring-boot/

[26] OAuth 2.0 Authorization Framework, http://tools.ietf.org/html/rfc6749

[27] OASIS eXtensible Access Control Markup Language, https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

[28] ETSI NFV, http://www.etsi.org/technologies-clusters/technologies/nfv

[29] Network Function Virtualization Management and Orchestration. http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf