Demonstrating Multi-Domain Orchestration through Open Source MANO, OpenStack and OpenDaylight

Panagiotis Karamichailidis, Kostas Choumas and Thanasis Korakis
Dept. of ECE, University of Thessaly, Volos, Greece
Email: kohoumas, karamiha, korakis@uth.gr

***Abstract*—In recent years, the rise of Network Function Virtualization (NFV) makes the Network Service (NS) deployment much agile and flexible. The proprietary and custom-made hardware is replaced by a virtual and software-based infrastructure, that is easily exploited in a common way for the NS deployment. One of the most challenging problems in this environment is deploying and organizing in a large-scale and multi-domain infrastructure, which contains geographically distributed but interconnected data-centers. The proposed solution focuses on the extra networking operations required in a NFV infrastructure, managed by Open Source MANO, OpenStack and OpenDaylight. We develop software proxies that collaborate with the aforementioned tools and enhance their functionality. Finally, we implement and evaluate the proposed architecture, using the NITOS experimentation testbed.**

I. Introduction

As virtualization dominates in every aspect of networking and computing, Network Services (NSs) could not be unaffected by this. For example, the Evolved Packet Core (EPC) of LTE is a NS that previously was exclusively supported by proprietary and hardware implemented Network Functions (NFs) and now is not. The proliferation of Network Functions Virtualization (NFV) enables the software implementation of these NFs, that is decoupled from the utilized computation, storage and network resources. In this way, NFV exposes a new set of entities, named Virtualized NFs (VNFs) and NFV Infrastructure (NFVI). The latter entity includes all the resources used for the VNF deployment, management and execution.

According to the ETSI Standardization group, the NFV Management and Orchestration (NFV-MANO) [1] is a challenging task that requires the synergy of several functional blocks, organized in an architectural framework and collaborating through specified reference points. Except for the VNF Manager (VNFM), that is in charge of the VNF lifecycle, the other two functional blocks of our interest are the NFV Orchestrator (NFVO) and the Virtualized Infrastructure Manager (VIM). The NFVO has two main functions, the NS Orchestration (NSO) and the Resource Orchestration (RO), which implement the lifecycle management of the NSs and the orchestration of the NFVI resources across multiple VIMs respectively. The VIM is responsible for controlling and managing the NFVI resources within a single domain(1), leveraging on hypervisors and SDN controllers for the control of the computation/storage and network resources respectively. There are also specialized VIMs, such as the WAN Infrastructure Manager (WIM) that controls only network resources. In this paper, we focus on network and compute domains (or data-centers), controlled by WIMs and VIMs respectively, and we investigate the role of SDN in their operation. The interaction and functionality of the SDN controller within the NFV-MANO architecture is not clearly defined, even in the report on the SDN usage in the NFV-MANO [2], since the SDN controller could either be part of the VIM or the NFVI, among other options. *This work sheds some light on the functionalities that could be offered by the SDN controller, either as a part of the VIM or constituting the WIM, especially for enabling the multi-domain orchestration.*

For our deployment, we exploit and extend the state-of-the-art open-source software tools, named Open Source MANO (OSM) [3], OpenStack [4] and OpenDaylight [5], which are for the NFVO/VNFM, the VIM and the SDN control respectively. This triangle is one of the most widely-used options for implementing the NFV-MANO, and from now on is denoted as *Open*^{3}. Although *Open*^{3} enables a remarkable set of operations, it still does not support multi-domain NS deployment. We present a solution that enables the VNF deployment of a NS over multiple compute domains, as well as their interconnection through a network domain, relying on the *Open*^{3} tools and our custom-made and open-source software. Our software leverages on the functionalities offered by the OpenDaylight SDN controller. We evaluate our implementation and we showcase the results of our experimentation in the NITOS testbed [6].

II. Extending *Open*^{3} Networking

We present *Open*^{3}++, which extends the networking functionality of *Open*^{3} by enabling in more scenarios the interconnection of VNFs belonging to the same NS, either if they are collocated inside a single domain or spread to multiple domains.

Before proceeding, it will be helpful to clarify the terminology and the classification of the networks followed by OpenStack, since our software extending *Open*^{3} to *Open*^{3}++ is mainly handling these networks with use of OpenDaylight. In a single compute domain managed by OpenStack, the VNF interconnection is built either through tunnels over the OpenStack *management* network, which are named *tenant* or *project* networks, or using overlay networks on top of the OpenStack *guest* network, which are called *provider* networks. Our focus is on the latter case, since the guest network can

(1)A domain could be either administrative, which exists within a single organization/service provider, or technology oriented, based on the type of technology in scope.

978-1-5386-9376-6/19/$31.00

be physically connected to other network domains and the provider networks may be used for interconnecting VNFs of different compute domains. In *Open*^{3}, the provider networks exist "out there" and OpenStack simply interfaces them. But in *Open*^{3}++, they are deployed on demand by exploiting the SDN-LAN and SDN-WAN controllers with use of our extensions. The following sections give more details on the SDN-LAN and SDN-WAN controllers, as well as their usage for the creation of *flat* or *vlan* provider networks[(2)] connecting the compute nodes of a single or multiple domains.

A. SDN-LAN & SDN-proxy

In *Open*^{3}, OpenStack implements the VIM and optionally leverages on an OpenDaylight instance, named SDN-OvS in our examples, for configuring the OvS bridges running in the compute nodes (usually named as *br-int*). When OpenStack receives from the OSM's RO a request for three VNFs of a new NS, connected through a provider network, it chooses the compute nodes that these VNFs will be deployed according to its scheduling policy. Then, OpenStack deploys the VNFs and informs SDN-OvS to configure the bridges of the chosen compute nodes. The bridge configuration is sufficient for connecting the VNFs located in the same compute node (e.g. VNF 1 and VNF 2), or to export the traffic to the physical NIC, when it is directed from a VNF to another non-collocated one (e.g. from VNF 2 to VNF 3). However, the traffic forwarding between the NICs of the compute nodes requires the appropriate configuration of the OpenStack guest network connecting them. This is the task of the extra OpenDaylight instance, named SDN-LAN.

Open^3 does not include any interface to SDN-LAN, assuming that it is standalone and proactively builds and keeps active the provider networks, even in time periods that they are not used. On the other hand, *Open*^{3}++ relies on a software daemon, named SDN-proxy[(3)], which repetitively checks SDN-OvS and prompts dynamically SDN-LAN to deploy and keep only the needed provider networks on top of the domain's underlying network. More specifically, SDN-proxy checks SDN-OvS for the required provider networks and forces SDN-LAN to form an isolated overlay network for each provider network, which functions as an abstract layer-2 switch. This function is completed with the assistance of the *Virtual Tenant Network (VTN) Manager* plugin, which is used by both OpenDaylight instances implementing SDN-OvS and SDN-LAN. VTN-Manager enables the creation of a virtual bridge (*vBridge*) for each provider network and *VLAN mapping* is used for assigning the VLAN traffic of the provider network to the respective vBridge (VLAN 0 corresponds to the untagged flat network).

B. SDN-WAN & Orchestration-proxy

In a multi-domain NS deployment, there are multiple VNFs belonging again to the same NS, but they are deployed to different compute domains. In *Open*^{3}++, we developed a daemon called Orchestration-proxy[(4)], that is a proxy between the RO and the VIMs/WIMs. It pretends to be a VIM instance for the RO and a RO for the underlying VIMs/WIMs. In particular, Orchestration-proxy presents to the RO as a single VIM responsible for a single domain and receives the requests for the NS resources. Then, it "breaks" the set of resources and assigns the subsets to various compute domains, interacting with their VIMs. For these VIMs, it is behaving as the RO, requesting the VNF deployment to their compute domains.

Except for this task, the Orchestration-proxy is responsible to interact with the WIM(s) managing the network domain(s) interconnecting these compute domains, in order to build the VNF connections. Each WIM is actually an SDN controller, called SDN-WAN, responsible for a network domain used for the interconnection of other domains. *Open*^{3}++ uses OpenDaylight and the VTN-Manager plugin to implement SDN-WAN, which again creates one vBridge for each provider network connecting VNFs, and maps this vBridge to the VLAN traffic of the provider network. The goal of both SDN-LAN and SDN-WAN is to create overlay layer-2 networks on top of their domains, using VLAN for their isolation. The networks of the same VLAN but of different domains are stitched together.

III. Demonstration

Both SDN-LAN and SDN-WAN are OpenDaylight instances that utililize the VTN-Manager plugin, in order to build the provider networks. VTN-Manager exploits the Link Layer Discovery Protocol (LLDP) to discover the underlying network and retrieve the shortest path between each pair of switches. Each link has a configurable cost and the flow setup is reactive.

The shortest path connecting a VNF couple is chosen by VTN-Manager during the forwarding process of the very first ARP request. If the first packet is forwarded from VNF 1 to VNF 2, then the corresponding ARP request has the MAC of VNF 1 as source address and the broadcast MAC as destination address, thus, the SDN controller learns the ingress switch and the port that VNF 1 is attached to and pushes the packet to all other switches. The other switches, in turn, forward this packet to all ports except for the ones discovered by LLDP, delivering the packet only to VNFs and not to switches. Once VNF 2 responds with an ARP reply, the SDN controller learns the ingress switch and the port of VNF 2 and calculates the shortest path between the ingress switches of the two VNFs. Then, it deploys the flows to the switches participating in this path, matching only the packets sent from VNF 1 to VNF 2.

Although *Open*^{3}++ benefits from the dynamic deployment or removal of provider networks comparing to *Open*^{3}, its delay performance may be worse at the first packets exchanged between the VNF couples, due to its reactive flow configuration. This evaluation is to estimate the average time needed for the

[(2)] *flat* provider networks forward untagged traffic, while *vlan* provider networks expect for VLAN tagged traffic to forward.

[(3)] The repository of SDN-proxy is given in this URL: http://repo.nitlab.inf.uth.gr/karamiha/odl-project/tree/test

[(4)] The repository of the Orchestration-proxy software is given in this URL: http://repo.nitlab.inf.uth.gr/karamiha/orchestrator

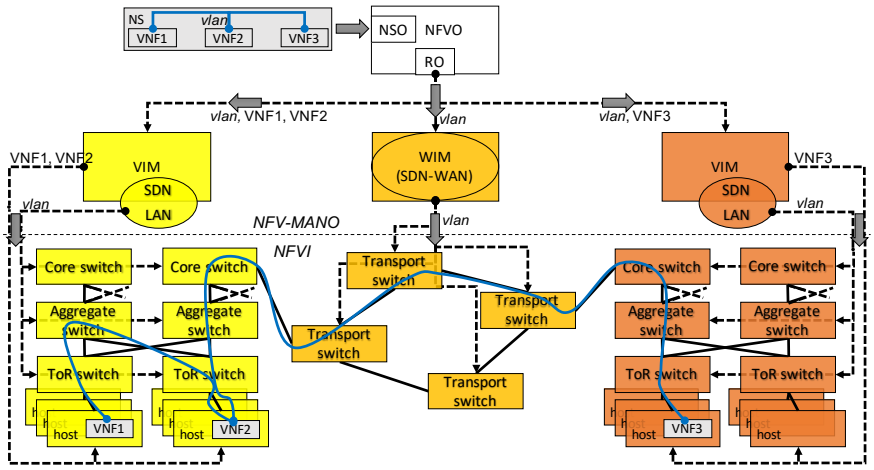

Fig. 1: Simple scenario illustrating the deployment of a NS with 3 VNFs over two compute domains (or data-centers), the yellow and the red one, interconnected through a network domain, the orange one.

flow configuration connecting a new VNF couple, when these VNFs are i) in the same compute domain or ii) in different compute domains.

A. End-to-end Delay

Let's assume an SDN network with multiple switches connected in a three-tier fat-tree network topology. This is a typical scenario in a compute domain or data-center with multiple interconnected compute nodes, using access or Top of Rack (ToR), aggregate and core layer switches, as it is depicted in Figure 1. The compute nodes are NITOS nodes, featuring Intel Core i7-2600 CPU at 3.40 GHz and 8M Cache, while the switches are emulated by executing Mininet in a separate NITOS node. All NITOS nodes are interconnected through an OpenFlow HP 3800 switch.

In the example of Figure 1, there is a NS with three sequentially connected VNFs, named VNF 1, 2 and 3. VNFs 1 and 2 are deployed in the yellow compute domain, in two different compute nodes belonging to different racks. In general, two VNFs of the same compute domain are either directly connected, if they are hosted on the same compute node, or connected through a single ToR switch, or two ToR and an aggregate switch or two ToR, two aggregate and a core switch. From our experimentation in NITOS, we checked that the ping delay for the first packet sent from VNF 1 to VNF 2 is between $8 - 12$ msec, including the delay of the ARP and ICMP forwarding, as well as the delay of the reactive flow configuration. The upper and lower delay limits are not significantly affected by the path length, except for the case that the VNFs are directly connected.

VNFs 2 and 3 are deployed in different compute domains, connected through a network domain. When VNF 2 pings VNF 3, the yellow SDN-LAN pushes the right yellow core switch to forward the ARP request of VNF 2 to the left orange switch. For the orange SDN-WAN, the left orange switch is the ingress switch of this ARP request, thus repeating the same process with before, the packet is forwarded from the right orange switch to the left red core switch. Finally, the red SDN-LAN receives this ARP request and similarly pushes it to the left red ToR switch to be forwarded to VNF 3. The duration of the ARP request forwarding from VNF 2 to VNF 3 is expected to be almost three times higher comparing to the previous case, between VNF 1 and VNF 2. This is also verified by our experimentation, since the duration of the ping between VNF 2 and VNF 3 is between $20 - 33$ msec.

IV. Conclusion

This paper presents an implementation of multi-domain NS deployment, which is based on OSM, OpenStack and OpenDaylight, as well as our SDN-proxy and Orchestration-proxy. There are many challenges, especially in programming Orchestation-proxy, which can be modeled as NP-hard optimization problems, such as Location-Routing Problem and Virtual Network Embedding. The focus of this work is to present the framework for applying the solutions, relying on widely-used software tools.

Acknowledgment

This work has been financially supported by the EU Horizon 2020 project 5G-PICTURE under grant agreement No 762057. The European Union and its agencies are not liable or otherwise responsible for the contents of this document; its content reflects the view of its authors only.

References

[1] ETSI GS NFV-MAN 001 v1.1.1 (2014-12), "Network Functions Virtualisation (NFV); Management and Orchestration".
[2] ETSI GS NFV-EVE 005 v1.1.1 (2015-12), "Network Functions Virtualisation (NFV); Ecosystem; Report on SDN Usage in NFV Architectural Framework".
[3] ETSI OSM, "Open Source MANO", https://osm.etsi.org.
[4] O. Sefraoui, M. Aissaoui and M. Eleuldj, "OpenStack: toward an open-source solution for cloud computing", International Journal of Computer Applications, vol. 55, no. 3, pp. 38-42, 2012.
[5] J. Medved, et al, "Opendaylight: Towards a Model-Driven SDN Controller architecture", IEEE WoWMoM, 2014.
[6] "Network Implementation Testbed using Open-Source platforms", https://nitlab.inf.uth.gr.