Mimicking a compute domain orchestrator with the ONOS SDN Controller

Mimicking a Compute Domain Orchestrator with the ONOS SDN Controller

Gabriele Castellano*, Ivano Cerrato*, Fulvio Risso*, Davide Pezzolla*, Antonio Manzalini†
*Politecnico di Torino, Dept. of Computer and Control Engineering, Torino, Italy
† TIM, Torino, Italy

***Abstract*—With the NFV paradigm, network services are usually instantiated in datacenters (e.g., as VMs), while software-defined networks provide just plain connectivity. However, common SDN controllers can do much more than just traffic steering; particularly they can execute network applications such as NAT, DHCP, and more. This paper presents a software architecture that can advertise an SDN domain as having compute capabilities, hence enabling an overarching multi-domain orchestrator to instantiate a network function either in a cloud or in an SDN domain. This allows an overarching orchestrator to fully exploit the processing capability of an SDN infrastructure and potentially enabling more aggressive optimization strategies across domains.**

I. Introduction

Current Network Functions Virtualization (NFV) services are mostly deployed as virtual machines (VMs) or lightweight containers in cloud computing domains (e.g., data centers), while the Software-Defined Network (SDN) infrastructure in between is only used to provide connectivity.

However, common SDN controllers such as *ONOS* and *OpenDayLight* can do much more than just traffic steering. In fact, acting as a middle-layer between the control and the data plane, they interact with the underlying switches (i.e., data plane) through a southbound interface (e.g., to program their forwarding tables), and expose a northbound interface that allows specific software bundles (a.k.a. *SDN applications*) to be dynamically deployed and executed on top of the controllers themselves (i.e., control plane). These applications, which contain the high level logic that decides which rules (e.g., OpenFlow) to deploy in the switches, can implement a set of the network functions (NFs) that are instead usually instantiated as VMs (e.g., NAT, stateless firewall) in data centers.

To the best of our knowledge, only the XoS orchestrator [1] exploits SDN controllers to execute NFs. However, XoS has well-defined requirements, such as it can control only an OpenStack datacenter in which the network infrastructure is managed by an ONOS controller. Moreover, XoS is not able to dynamically discover which NFs the domain is able to execute. For instance, this is a necessity arisen in a geographical SDN testbed we have access to, where ONOS is being populated by additional software (i.e., SDN applications), which happens outside our control. In this case, an overarching *multi-domain orchestrator* (MDO) (Figure 1) cannot take advantage from those new bundles, because it even does not know they are available.

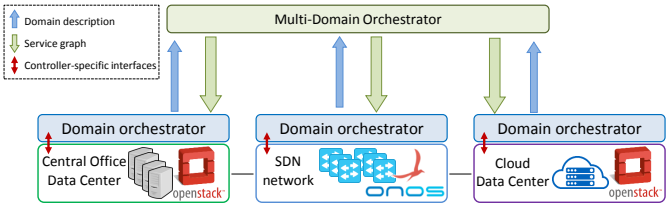

Fig. 1. Overall view of the multi-domain orchestration architecture.

Based on these considerations, this paper presents a software architecture that enables an MDO to retrieve the features of each domain, as well as to understand when a domain gets new features (e.g., a new bundle available/installed in an SDN domain). Moreover, based on such information, the MDO is able to transparently instantiate NFs in the most appropriate domain, regardless of whether it is a data center or an SDN network, thus allowing better optimization strategies when selecting the best location on which a network service must be executed. Particularly, we extend the ONOS controller with a software component, called *domain orchestrator* (DO), which: *(i)* enables the SDN domain to be exploited for the deployment of NFs (and not only for the creation of network paths); *(ii)* exports (to the MDO) a set of information (e.g., applications installed in the SDN controller) that may change from one SDN domain to another, and which may be updated over time.

This paper is structured as follows: Section II presents an overview of the orchestration framework, while Section III details the SDN domain orchestrator that we built on top of the ONOS controller. Finally, Section IV provides some preliminary evaluation results.

II. Multi-Domain Orchestration Overview

Figure 1 provides an overview of our architecture, where the MDO[1] deploys network services on heterogeneous domains, which include both data centers (e.g., under the control of the OpenStack cloud toolkit), and SDN networks (under the control of ONOS). The integration of heterogeneous domains requires a DO per domain, which *(i)* exports a domain description that can be exploited by the MDO, e.g., to select the best domain to execute a service, and *(ii)* is able to instantiate the required NFs on the resources available in the domain itself.

Domain description. DOs export the domain description according to a specific YANG data model, which includes

[1]We use the FROG orchestrator: https://github.com/netgroup-polito/frog4.

both networking and computing aspects. Particularly, from the point of view of networking, each domain is described as a "big switch" with a set of boundary interfaces, each one characterized by a set of information that allows the MDO to properly reconstruct the whole multi-domain topology. Instead, from the point of view of computing, the domain is associated with a set of *capabilities*, each one representing a specific NF (e.g., *firewall*, *NAT*) that the domain is able to implement, e.g., because it is available as software/hardware component within the domain itself (such as a specific bundle in case of ONOS, or a VM image in case of OpenStack).

Service deployment. The MDO receives service requests on its northbound interface, which come as graphs made with *(i)* NFs, *(ii)* service access points (SAPs), and *(iii)* logical links that connect the above entities. A SAP represents an entry/exit point of traffic into/from the service graph. It may be associated with a traffic classifier, which indicates that packets leaving the graph from such a SAP must be encapsulated, e.g., in a specific GRE tunnel, as well as it indicates which traffic can enter in the service graph through the SAP itself. Based on the description of each domain, the MDO selects the best target for each NF and generates a new sub-graph per domain. The full detail of the graph splitting process, as well as the inter-domain traffic steering set up algorithm, are presented in [2].

III. SDN Domain Orchestrator Prototype

The SDN Domain Orchestrator (SDN-DO) orchestrates a network infrastructure in which a set of devices are managed by a vanilla SDN controller, thus enabling the MDO to deploy NFs in existing SDN networks.

A. Exploiting SDN applications in NFs graphs: considerations

Although a NF (e.g., firewall) implemented as an SDN application and the same NF deployed as a VM-based middlebox look the same from an high-level perspective, there are several differences that have an important impact on how (and which) service graphs are supported in the different domains.

A first difference is that, while VMs reside on the data plane, SDN applications are executed in the control plane. Therefore, unlike the former, SDN applications: *(i)* do not explicitly process all packets, but usually only the first ones of a flow, and then install rules in the underlying (e.g., OpenFlow) switches so that the next packets are processed in their hardware pipeline; *(ii)* do not have network interfaces that can be connected to each other, in order to implement the chain described in the service graph. As a consequence, SDN applications cannot be easily chained, as it is not trivial to guarantee that the rules instantiated by a NF B operate only on traffic already processed by rules installed by another NF A, with B following A in the service chain. Hence, our current SDN-DO supports only graphs with up to one chained NF, while support for complex graphs is left as a future work[2].

[2]However, a similar problem has been solved in current implementation between flow rules instantiated by NFs and those directly installed by the SDN-DO itself.

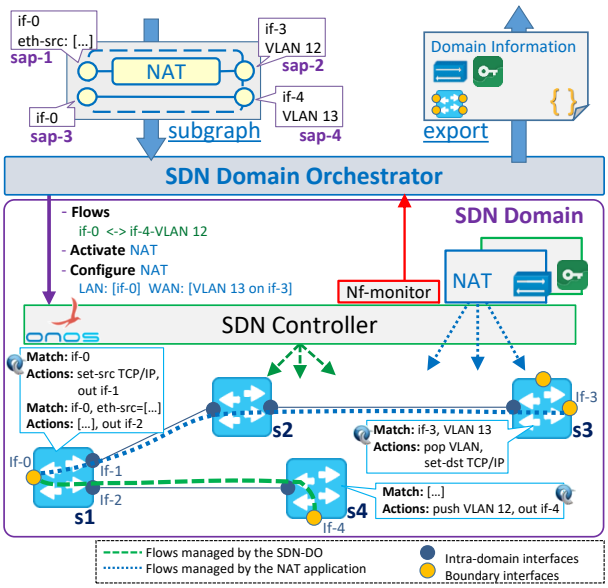

Fig. 2. Service graph deployment in an SDN domain.

A second important consideration is that, unlike VMs listed by a *VNFs image service* in a datacenter domain (e.g., Glance in OpenStack), not all the bundles available in an SDN controller may be exploited to implement NFs. For example, some of them may not actually implement any NF (e.g., the utility ONOS bundle that provides the API to configure other applications), while others, although implementing NFs, may not allow to setup specific parameters (e.g., the subset of traffic on which they have to operate), thus limiting the number of sub-graphs in which they can be deployed. Therefore, the SDN-DO (details in III-B) checks, for any bundle available in ONOS, whether the above requirements are satisfied or not.

Finally, the ONOS controller does not support the activation of multiple instances of the same bundle, preventing the SDN-DO to deploy the same NF in multiple graphs; then, if desired, this feature has to be implemented by the bundles themselves.

B. Discovering and exporting domain information

The SDN-DO exports information about virtual/physical interfaces of the switches that are responsible for connecting the domain to the rest of the world, hence that handle incoming/outgoing traffic. The list of the domain boundary interfaces and the associated parameters is exported at bootstrap and then again each time a change is detected.

Furthermore, the SDN-DO has to dynamically discover the NFs that are available (as software bundles) in the ONOS controller; to do this, we designed a particular ONOS bundle called *"nf-monitor"* (Figure 2), which identifies those applications that can be exploited to implement NFs and uses a specific ONOS northbound API in order to intercept the following events: *(i) bundle installed* - a new application is available, which may be now used to implement a NF; *(ii) bundle removed* - the application is no longer available and cannot be used anymore to implement NFs; *(iii) bundle*

activated - the application is running, then, given that ONOS does not support multiple instances of the same application, that bundle is (temporarily) no longer available for future services (hence, the SDN-DO will no longer advertise that capability), unless it explicitly supports multi-tenancy; *(iv) bundle deactivated* - the application is no longer used, hence it is available again for a new service graph. Each time the status of a NF bundle changes, the SDN-DO gets notified by the `nf-monitor` bundle, then updates the exported information and notifies the MDO accordingly.

C. Deploying service graphs

Given the considerations discussed in III-A, when the SDN-DO receives a service (sub-)graph it immediately checks the availability of the required NFs in the ONOS controller and the validity of the parameters associated with each SAP (e.g., VLAN IDs, GRE keys). If the service graph is valid, the SDN-DO interacts with ONOS in order to activate the proper NFs.

Links of the graph are handled in different ways depending on the elements they connect to. If the link connects two SAPs (e.g., the connection between `SAP-3` and `SAP-4` in Figure 2), the SDN-DO, through the SDN controller, directly instantiates the flow rules to setup the connection between the SAPs. In addition, based on information associated with each SAP, it instantiates the flow rules needed to properly tag/encapsulate the packets before sending them outside the domain, and to classify incoming packets and recognize the SAP they refer to. Instead, if a link connects a SAP to a NF (e.g., the connection between `SAP-2` and the NAT in Figure 2), the SDN-DO configures the NF software bundle with information about the traffic on which it has to operate, which is derived from the parameters associated with the SAP. For instance, the NAT of Figure 2 is configured to operate on specific traffic coming from interfaces `if-0` and `if-3` (i.e., with a specific source MAC address in the former case, with VLAN_ID 12 in the latter), to tag traffic transmitted on `if-3` with the VLAN_ID 12, and to untag traffic tagged with VLAN_12 and received from such an interface. Hence, in this case incoming/outgoing traffic is handled directly by the NF, which has then to be aware of these parameters.

IV. Validation

To validate our proposal we deployed some graphs on the scenario depicted in Figure 3. During each deployment the MDO dynamically decides whether to instantiate NFs in the SDN domain or not, based on the information exported by the domain itself, which can change over the time[3]. Then we evaluated the advantages, in terms of performance, brought by exploiting the SDN domain for the execution of NFs.

Particularly, we instantiated the service graph shown in Figure 3 in which SAPs are connected to an host and a server directly attached to the SDN domain. In a first phase, the SDN domain is configured without the possibility of running NFs, therefore the requested NFs are instantiated through VMs

[3]A video is available at: https://www.youtube.com/watch?v=N6SBo2f6Lyc

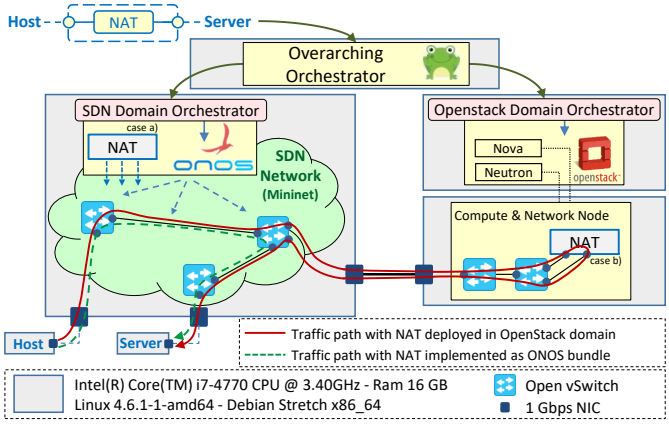

Fig. 3. Validation scenario.

TABLE I
End-to-end performance with different service implementations.

	Avg Latency [ms]	Throughput [Mbit/s]
vm-nat	1.041	838
sdn-nat	0.465	962

on the cloud computing domain, while the SDN domain is exploited by the MDO just for traffic steering purposes (traffic follows the solid line in figure). In a second phase, the SDN domain is configured with the possibility of executing some NFs (e.g., NAT, DHCP) by installing the proper ONOS applications; thus, once a new deployment request is performed to the MDO, these NFs are instantiated in the SDN domain as ONOS bundles. This way, traffic exchanged by the host and the server only traverses the SDN network (dashed line in Figure 3), and the data center is not involved at all in the service deployment.

In both cases, we measured the end-to-end latency introduced by the service and the throughput between the host and the server; the obtained the results are shown in Table I[4]. As expected, performance are better in case the NAT is executed as a bundle running in the ONOS controller, since in this case *(i)* traffic is kept local to the SDN domain, and *(ii)* the NAT is actually implemented as a set of OpenFlow rules installed by the NAT bundle in the switches (only the first packet of a flow is then processed by the bundle, while the following packets are directly processed within the underlying devices).

References

[1] L. Peterson, S. Baker, M. De Leenheer, A. Bavier, S. Bhatia, M. Wawrzoniak, J. Nelson, and J. Hartman, "Xos: An extensible cloud operating system," in *Proceedings of the 2Nd International Workshop on Software-Defined Ecosystems*, ser. BigSystem '15. New York, NY, USA: ACM, 2015, pp. 23–30.

[2] R. Bonafiglia, G. Castellano, I. Cerrato, and F. Risso, "End-to-end service orchestration across sdn and cloud computing domains," in *IEEE Conference and Workshops on Network Softwarization (NetSoft 2017)*, Bologna, Italy, 2017.

[4]For the latency we used the `ping` tool, while `iperf3` generating TCP traffic has been used for the throughput measurements.