Network Function Chaining in DCs: the Unified Recurring Control Approach

Janos Elek, David Jocha and Robert Szabo
Ericsson Research, Hungary

***Abstract*—The introduction of Network Function Virtualization (NFV) in carrier-grade networks promises improved operations in terms of flexibility, efficiency, and manageability. NFV is an approach to combine network and compute virtualizations together. However, network and compute resource domains expose different virtualizations and programmable interfaces. In the UNIFY framework[1] we introduced a joint compute and network virtualization and programming. In this paper we analyze how the joint virtualization and control of UNIFY can be applied to a OpenStack-OpenDaylight Data Center. We show through an implementation, that the proposed unified recurring control has clear advantages compared to standard separated cloud and WAN orchestration approaches.**

I. Introduction

Today, rigid network control limits the flexibility of service creation. It is not unusual that an average service creation time cycle is over 90 hours, which must be decreased to the order of minutes [2]. Therefore, the EU-FP7 UNIFY project [3] envisions rapid and flexible service programmability in a "unified production environment" integrating data centre and carrier network resources. We pursue a service and resource programmability framework capable of overcoming existing limitations in terms of efficiency in operations, flexibility in service deployment and fast and flexible service provisioning. One of the crucial enablers to support this unified production environment is the *definition of open and standardized programmatic interfaces* from users (end or enterprise users, other service providers, over the top providers, etc.) to the infrastructures [4].

Network Function Virtualization (NFV) [5], Software Defined Networking (SDN) [6] and Cloud, e.g., OpenStack (OS)[7], are key technology enablers nowadays for virtualization. Throughout virtualization, the community pursues telecommunication grade flexible services with reduced costs. European Telecommunications Standards Institute (ETSI) made significant effort to boost open demonstrations of proofs of concept (PoCs) to build industrial awareness and confidence in NFV [8]. Many of the PoCs investigate how legacy Data Centers (DCs) can be used as a NFV Infrastructure (NFVI) for NFV. For example, PoC#1: "Demonstration of CloudNFV Open NFV Framework" shows how an OS controlled DC could be brought under the NFV framework.

Internet Engineering Task Force (IETF) Service Function Chaining (SFC) working group [9] looks into the problem how to deliver end-to-end services through the chain of service functions. Many of such service functions are envisioned to be transparent to the client, i.e., they intercept the client connection for adding value to the services without the knowledge of the client [10].

However, deploying service function chains in DCs with Virtualized Network Functions (VNFs) are far from trivial. For example, examine the simple case of a DC connected through one link to a Wide Area Network (WAN). Assume that a service provider wants to add one or more transparent network function to some customer services available on the WAN for the users. The provider's service orchestrator takes the service definition, e.g., in the form of a Virtualized Network Function Forwarding Graph (VNF-FG), and maps it to the infrastructure domain. The OS controller instantiates the Virtual Machines (VMs), establishes intra-DC networking and last but not least the explicit forwarding overlay is configured on the WAN by the network controller.

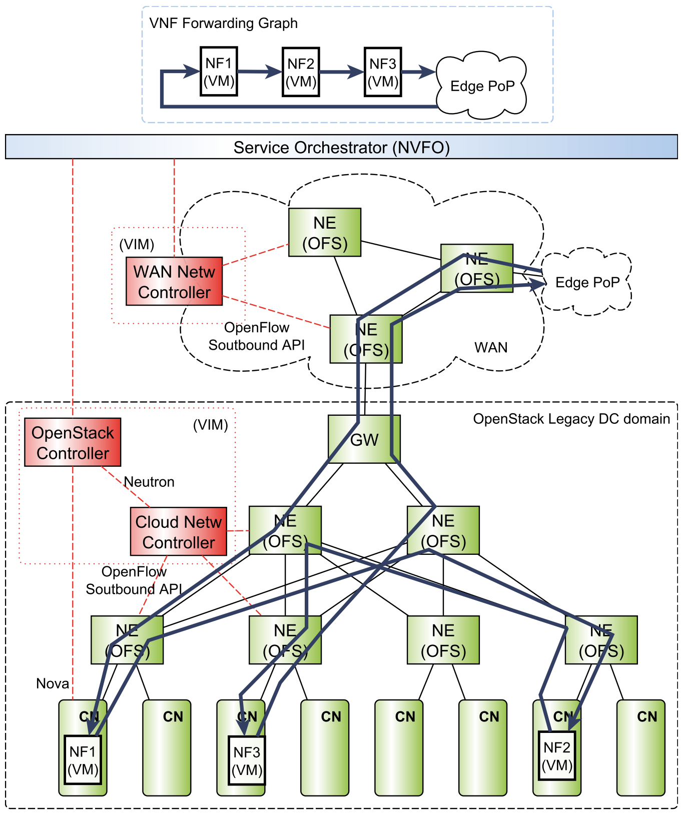

Fig. 1. VNF Forwarding Graph in a DC

Now the main question: Do the different technology domains offer proper abstractions and expose powerful Application Programming Interfaces (APIs) for the service orchestrator to be able to decide and execute optimized placement of Network Functions (NFs) and configuration of the forwarding chain? We argue that with nowadays separated compute and network control APIs, network overlay is typically just an afterthought after compute orchestration. Furthermore, the lack of powerful network abstraction and lack of exposed control for compute domains also constrains the options available for chain forwarding[4].

Fortunately, the EU-FP7-UNIFY project pursues joint abstraction and control for compute and network resources. In this paper, we present our design for the joint virtualization with associated control API for DC domains *to enable telco grade network function chains*. Our design follows the UNIFY architecture and its joint compute and network virtualization concept.

In the rest of the paper in Sec. II we discuss related work together with an introduction to the UNIFY concepts. In Sec. III we present our design and evaluation for the unified DC scenario. Finally, we draw conclusions in Sec. IV.

II. Related work

A. NFV, Cloud and SDN

Flexible service definition and creation start by abstracting and formalizing the service into the concept of network function forwarding graphs (see for example ETSI VNF-FG in [11]). These graphs represent the way in which service end points (e.g., customer's access) are interconnected with the desired network functions (such as firewalls, load balancers) or other functionalities (e.g., web-server) to deliver a service. Service graph representations form the input for the management and orchestration to instantiate and configure the requested service. ETSI defined a Management and Orchestration (MANO) framework in [12]. Among others, MANO components are the Network Function Virtualization Orchestrator (NFVO) for the lifecycle management of the services and the Virtualized Infrastructure Managers (VIMs) for controlling and managing compute, storage and network resources (see Fig. 1 and Fig. 2).

The cloud view of the ETSI MANO framework is that OS controller can act as a VIM responsible for the compute domain together with its internal networking (as shown in Fig. 1). The problem, however, is that OS's northbound API is not powerful enough to control networking beyond basic L3 network configuration. That is, Neutron[13] only supports traditional L2 and L3 networking, some extensions like load balancer, but no direct forwarding control over the DC switches. Basically from networking point of view, all the VMs in an OS network are identified with their private IP. This IP only visible to VMs in the same network slice. If one needs a publicly available IP for a VM, a floating IP must be created. From external domains only through this IP can the given VM be reached. This already imply the requirements for some kind of L3 tunneling technology for traffic steering into a VM VNF. Additionally, the Nova scheduler of OS use least loaded, random or availability zones to place VMs, independent of their communication patters, networking constraints, etc. (see the randomly deployed VMs in Fig. 1 and the resulting forwarding overlay).

On the networking part, Open Networking Forum (ONF) works on the definition of an SDN architecture[14]. ONF focuses on three layers: data, control and application plane layers, but also include traditional management plane and end user systems into their architecture. SDN applications are defined as control plane functions operating over the forwarding abstraction offered by the SDN Controller The SDN control plane's main responsibilities are $i)$ creating the domain wide abstraction for internal use; $ii)$ creating application or client SDN controller specific virtualization and policy enforcement; and $iii)$ coordinating resources and operations for virtualization. The data plane of the SDN architecture constitutes of Network Elements (NEs) that deals directly with customer traffic. NEs' forwarding behavior is configured by an SDN Controller.

Recently, the OpenDaylight (ODL) open-source SDN Controller project developed a northbound neutron adapter plugin, therefore an ODL SDN Controller can be used to configure the DC internal network (see Cloud Network Controller in Fig. 1). As long as the DC and the WAN of Fig. 1 belong to the same administration, direct control of the Cloud Network Controller from the Service Orchestrator could remedy the intra DC forwarding control problems.

B. UNIFYing cloud and network resources

In the UNIFY programmability framework we anticipated a three-layered architecture[1] distinguishing infrastructure, orchestration and service layers (see big dashed boxes of Fig. 2). The major differences between the ETSI MANO and the UNIFY frameworks are that UNIFY $i)$ logically centralize the VIM related resource management functions into a Resource Orchestrator (RO) and $ii)$ that the UNIFY RO operates at a joint compute and network virtualization with control unlike having the network as an afterthought. A more detailed comparison of UNIFY to ETSI MANO and ONF SDN, as well as an architecture description is given in [15].

Central to the UNIFY concept is the joint virtualization and control of compute resources and network forwarding behavior. The joint virtualization is realized as a Big Switch with Big Software (BiS-BiS) abstraction, which combines both compute and networking capabilities into a single representation, which can accommodate VNFs, create VNF ports and receive SDN forwarding control for these ports.

The UNIFY architecture is recursive, i.e., arbitrary number of RO with virtualizers can be stacked on top of each other. This native UNIFY reference point is denoted by `Sl-Or` (see the multi-level recursion in Fig. 2 and also [1]).

The role of the Controller Adapter (CA) in the UNIFY architecture is $i)$ to combine virtualizations and split control for native UNIFY domains and $ii)$ to create appropriate virtualization and to translate control from the Network Function Forwarding Graph (NF-FG) abstraction to technology specific domains. Fig. 2 shows two technology domains similar to Fig. 1: an SDN WAN and a DC domain. The problem is twofold: $i)$ how to create UNIFY abstraction and translate control for legacy SDN and Cloud domains and $ii)$ how to transfer a legacy DC into a native UNIFY domain. In this paper, we answer these two questions.

C. Joint compute and network virtualization with control

We are not the first to argue for the introduction of a unified and programmable system abstraction. For example,

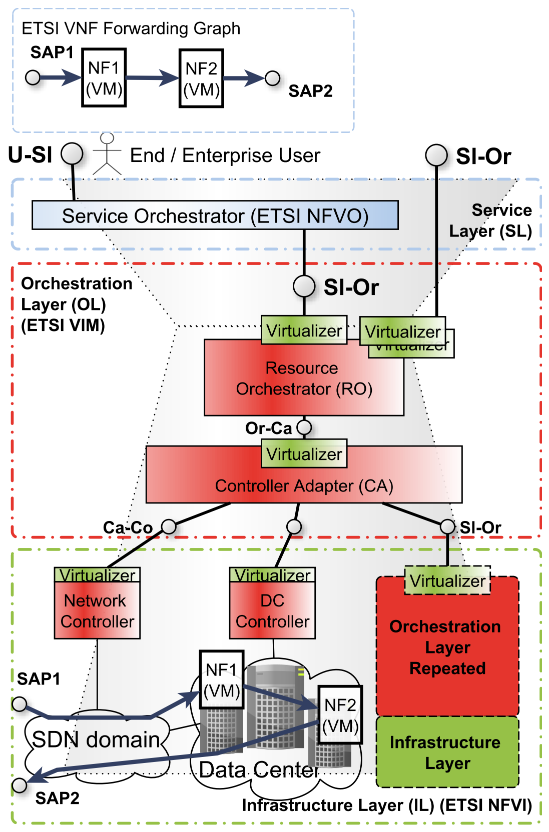

Fig. 2. UNIFY narrow waist architecture with programmatic reference points and some exemplary infrastructure options

in their “middlebox manifesto”, Sekar et al. [16] argue that despite the critical role that middleboxes play in introducing new network functionality, they have been largely ignored so far for designing networks that are amenable to innovation. The authors envision a world with software-centric middlebox implementations running on general-purpose hardware platforms that are managed via open and extensible management APIs. However, while we share the philosophy of [16], our work focuses more on the specific case of DC and carrier network unification, and proposes a concrete architecture and programmability framework.

The Stratos framework[17] aims at composite virtual middle-box orchestration in a correct and scalable manner in DCs. The framework design is to be easy to use by the service consumers, but no exact description of the interface for deployment requests are given. Moreover we focus on the broader spectrum of NFs while it concentrate on middle-boxes. A very similar approach is presented by Benson et al. [18], but none of these two works consider the unification of multiple domains.

It is worth to mention the OpenNF control plane[19], where the focus is on the state transition of NFs in case of scale-out events. While it is important to have a failproof method for resource provisioning, our main contribution is on the unified controlling. We believe, however, that our system is fully compatible with the solutions provided by the above control plane.

III. Design and Evaluation

Let’s assume that the Service Provider (SP) pursues joint compute and network virtualization with programmatic control as described in Sec. II-B. This can only be achieved if compute and networking requests and requirements are matched and merged together within the DC for joint considerations. However, this yields the idea of genuinely combining and transmitting compute and networking requests together. According to the UNIFY programmatic framework, the NF-FG at the `Sl-Or` reference point (see Fig. 2) is one such combination of compute and network request with associated constraints. However, one need to build an additional layer on top of the existing datacenter controller, in order to support the joint programmatic API. Moreover the DC inner working must be modified to overcome the scheduler’s limitations described in the I. section.

With a legacy OS DC one can only use L2/L3 networking in the internal network, that is, the VMs can be reached only through its public IP. Consequently, to steer traffic into an NF deployed as a VM in such a DC some tunneling must be used. We investigated a scenario like this and found that the creation of VxLAN tunnels can be appropriate in such a case. However, the additional processing coupled with the tunneling and the overhead on the packet size can be a not acceptable limitation. Moreover, the OS scheduler can not provide optimal VM placement on its own. In this section we show how a joint compute and network virtualization with control can remedy the situation as part of the UNIFY concept.

1) Toward telco grade NFC: A typical OS DC consists of a couple of compute nodes connected with Open Flow Switches (OFSes) and a control node on which the OS control functions are running. The control node expose a standard REST API through which one can instantiate VMs. In our scenario there is also an ODL controller on the control node that configures the OFSes on the request of OS controller. The ODL controller provides a REST API too. These two API make possible the joint orchestration. We implemented the `CA-ODL` controller

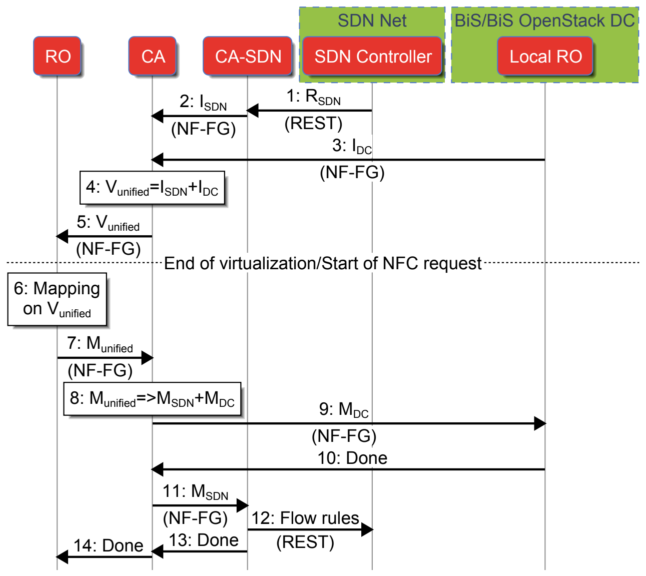

Fig. 3. NF-FG scenario sequence diagram

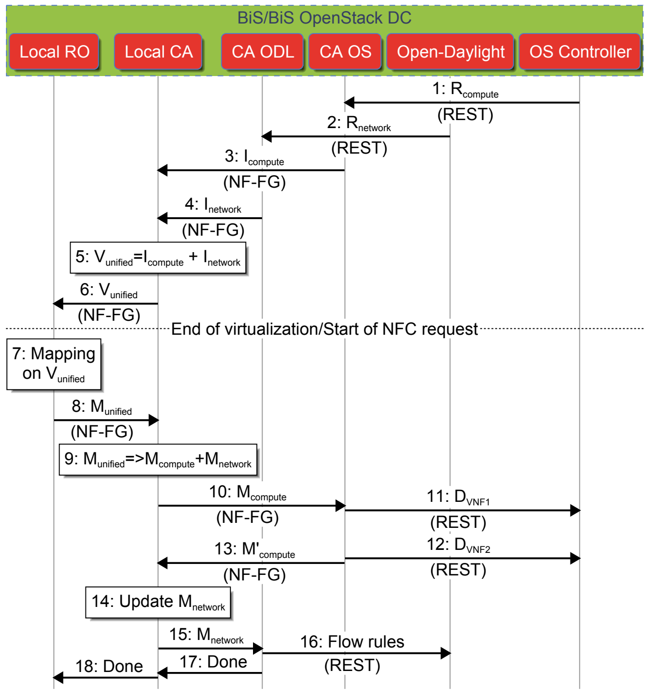

Fig. 4. NF-FG scenario sequence diagram

adapter towards the ODL and the CA−OS adapter towards the OS controller. In that way we have direct control over the DC internal network therefore we do not need to create VxLAN tunnels, and knowing the exact locations of the compute nodes we can achieve optimal VM deployment by commanding the OS controller to deploy VMs on a specific node.

In that case, however, the RO on top of these two controller sees all the underlying resource details. This would give away too much details and control if the overarching orchestration belongs to a separate business entity. In the typical case a third party DC owner wouldn't allow to give such an access to the RO and expose its whole intra DC network topology. Therefore, we introduced a DC domain virtualization on the top of the DC that not just hides (abstracts) the internal resource views but at the same time provides equivalently powerful control as if all the details were exposed.

The UNIFY scenario introduces such a joint compute and network virtualization with control API in the form of the NF-FG. Shall the DC domain become a native UNIFY domain (see bottom right recursion in Fig. 2), the need for an external technology adapter diminishes and only a simple NF-FG multiplexer/de-multiplexer CA is needed. The new domain contains a Local RO that advertises the infrastructure virtualization with NF-FGs to the global CA, that aggregates it to the global view of the (overarching) RO. Moreover the Local RO is waiting for service requests again in the form of NF-FG that can be directly given by the CA.

We describe now in details the process of infrastructure discovery and an Network Function Chaining (NFC) deployment process.

2) Virtualizations: An overall picture of the discovery process can be seen in Fig. 5. On the bottom right hand side resides the OS/ODL domain with its Local RO and multiple controller adapters. Note the Service Access Point (SAP) on the gateway switch (SAPb) that is eventually propagated to the Local RO and also to the global CA. On the left can be seen the SDN domain with three OpenvSwitches (OVSes) and three SAPs from which the SAPa port is connected to the OS/ODL domain's SAP. The discovery process is also showed in the upper part of the sequence diagram Fig.3 where the relevant information flows and the appropriate communication protocols are showed. The process start with the capturing of the networking resources, R_{SDN}, in (step_1). This information is processed and converted by the CA−SDN. In (step_2) the SDN domain's infrastructure description, I_{SDN}, is sent to the global CA as a NF-FG. In the same manner the Local RO captures its underlying infrastructure and creates a virtualized infrastructure description, I_{DC}, that is sent to the global CA in (step_3). In (step_4) the two infrastructure is combined to a unified view, $V_{unified}$, and also the interconnection between the two domain is inserted into this view. This external configuration is given by an NF-FG to the CA, and can be seen in Fig. 5 as the blue configuration data. In (step_5) the unified view is sent to the global RO by the global CA. The informations sent as NF-FG can be seen in Fig. 5 on the appropriate interfaces.

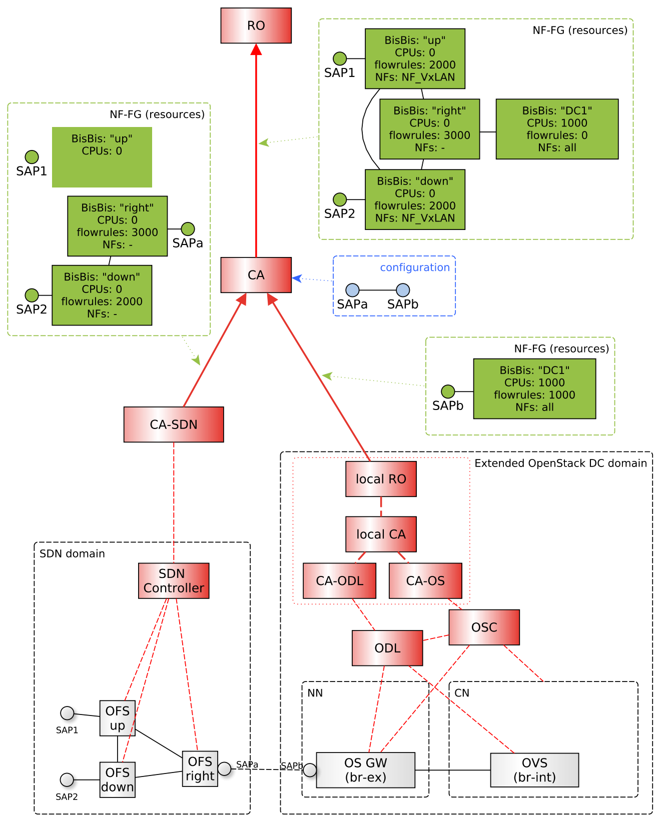
RO
NF-FG (resources)
SAP1
BisBis: "up" CPUs: 0 flowrules: 2000 NFs: NF_VxLAN
BisBis: "right" CPUs: 0 flowrules: 3000 NFs: -
BisBis: "DC1" CPUs: 1000 flowrules: 0 NFs: all
BisBis: "down" CPUs: 0 flowrules: 2000 NFs: NF_VxLAN
SAP2
NF-FG (resources)
BisBis: "up" CPUs: 0
SAP1
BisBis: "right" CPUs: 0 flowrules: 3000 NFs: -
SAPa
BisBis: "down" CPUs: 0 flowrules: 2000 NFs: -
SAP2
CA
configuration
SAPa SAPb
NF-FG (resources)
SAPb
BisBis: "DC1" CPUs: 1000 flowrules: 1000 NFs: all
CA-SDN
Extended OpenStack DC domain
local RO
local CA
CA-ODL
CA-OS
SDN domain
SDN Controller
ODL
OSC
NN
CN
SAP1
OFS up
SAP2
OFS down
OFS right
SAPa
SAPb
OS GW (br-ex)
OVS (br-int)

Fig. 5. Unified NF-FG infrastructure view

Inside the DC domain, the OS and the ODL controllers are used by the CA−OS and CA−ODL modules to collect the network topology and the corresponding compute nodes. This process is showed on the upper part of sequence diagram in Fig.4. First the compute resources, $R_{compute}$, then the network resources, $R_{network}$ are captured by the two controller adapters in (steps_1,2). From these information both create an infrastructure description, $I_{compute}$ and $I_{network}$ that is sent to the Local CA in (step_3,4). The two description then combined in (step_5) by the Local CA and a unified view, $V_{unified}$ is created. In (step_6) $V_{unified}$ is sent to the Local RO. The Local RO, based on the combined virtualization and additional configuration data, creates the virtualized view of the domain that is sent to the global CA (step_3) in Fig. 3. Note, that the internal domain wide virtualization and the external virtualization can differ; actually, the role of the Local RO is to hide the internals of the DC and present a virtualization best suited for the client/consumer but at the same time provide powerful resource control capabilities. In Fig. 5 the virtualized view is a single BiS-BiS with a single SAP (SAPb).

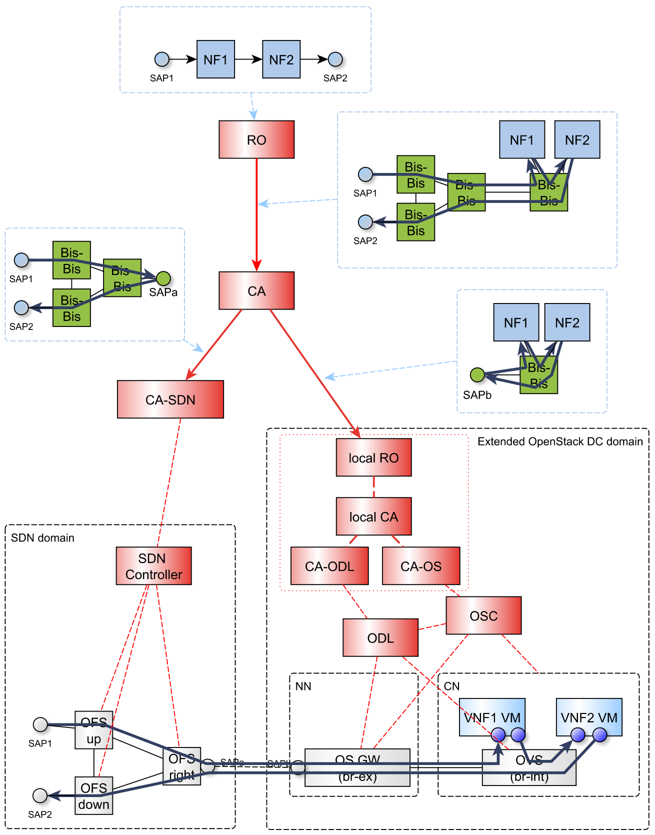

Fig. 6. Unified NF-FG mapped request

3) NFC request: A simple example of the deployment process can be seen in Fig. 6. The abstract request given to the global RO consists of two NFs inserted between SAP1 and SAP2. The sequence of this process can be seen in the lower part of Fig. 3 starting with the mapping of the given request by the global RO in (step_6). In the resulted mapping the two NFs are placed on the BiS-BiS representing the OS/ODL domain and flow rules are created in the up and right BiS-BiS Nodes for the inward traffic and in the right and down BiS-BiS Nodes for the outward traffic. Moreover flow rules are mapped to the DC1 BiS-BiS for the inward and outward traffic and also for the interconnection of the two NFs. Then the created mapping, $M_{unififed}$, is sent to the global CA in (step_7). The CA converts the flow rules to domain specific rules (see section III-4) and splits the unified mapping into M_{SDN} for the SDN domain and M_{DC} for the DC domain in (step_8). The CA sends first the M_{DC} NF-FG to the Local RO in (step_9) and after getting a Done response in (step_10) sends the M_{SDN} NF-FG to the CA−SDN. In response the CA−SDN installs the requested flow rules in (step_12) through the SDN controller REST interface and sends a Done message to the global CA in (step_13). With this the CA is ready with the deployment and indicates this by sending a Done message to the global RO.

The orchestration process in the DC domain is showed in the lower part of Fig. 4. Getting the request the Local RO maps it to the unified view advertised by the Local CA. In (step_8) the created mapping is sent to the Local CA as a NF-FG. Again, the CA splits these request to a $M_{compute}$ and a $M_{network}$ part in (step_9) and sends first the compute part to the CA−OS in (step_10). The $M_{compute}$ request contains only the two NF, therefore the only request sent to the OS controller is two deployment message through the REST interface in (step_11,12). If the deployment was successful both REST requests return a response containing the ports created for the new VMs in the DC. With this information the $M_{compute}$ NF-FG is updated and the new NF-FG, $M'_{compute}$, is sent back to the Local CA in (step_13). The port informations then inserted into the $M_{network}$ NF-FG by the CA in (step_14) and sent to the CA−ODL in (step_15). The necessary flow rules are instantiated by the ODL controller for the request of the CA−ODL in (step_16) and in case of a success a Done message is sent to the Local CA what in turn responds also with a Done message in (step_18) for the Local RO original request.

4) Flow rule mapping and tagging: After getting a request the RO maps it onto the unified view showed by the CA. This mapping consists not just finding an appropriate place for the NFs in the request, but selecting a paths between those and creating flow rules that realize these paths. For creating the paths we used a type of tunneling technology which consists a classifier rule, a tunnel starting rule, one or more tunnel matching rule and a tunnel closing rule. From the RO point of view it is irrelevant what type of specific technology is used. It only selects an abstract tag for a given path that is converted to a concrete tunneling type by the CA. The possible tunneling technologies are populated in the CA in the process of discovery when all the ports in the infrastructure advertise what it is supporting.

In our example there are three paths: 1) SAP1 to NF1, 2) NF1 to NF2 and 3) NF2 to SAP2. As we did not give any filtering criteria in the original request the RO will classify all the traffic coming from SAP1 in BiS-BiS up as an input to the NF1 and tag them with tag T_1. This tag is matched in BiS-BiS right and also in BiS-BiS DC1 where it is deleted and the packets are sent to NF1 without any tag. As the two NFs are residing on the same BiS-BiS there is no need for a tag in the forwarding rule between them. The outward path is created in the same manner as the inward but with tag T_2.

In our implementation all the ports support the VLAN tunneling and this is advertised from the SDN domain through the CA−SDN domain as parameters of the ports and from the DC domain as a parameter of SAPb. Moreover the available VLAN ID space in the domains is also known by the CA, therefore it picks available IDs VID_1 for tag T_1 and VID_2 for tag T_2. The so created flow rules then propagated to the

CA-SDN where are instantiated as OpenFlow rules. The DC domain, however, gets a request in which the traffic coupled with SAPb has a filtering criteria indicating the VLAN ID VID_1 and VID_2. The Local RO can decide not to use this ID as tag in its internal domain, but from SAPb to classify only the traffic with the VLAN ID VID_1 and send traffic belonging to the requested path only with the VID_2. With this procedure we fully decouple the orchestration from the used domain specific technology and provide efficient mechanism for traffic steering in a inter-domain consistent manner.

5) Summary: With the introduction of a Local RO in a DC domain we can enable joint virtualization and control for external orchestrators without necessarily exposing domain internal details. This paves the way to be able to offer telco grade NFC services via DCs.

In our simple example, the view provided by the Local RO was a single BiS-BiS node. However, the virtualization is not limited to a single BiS-BiS but arbitrary topology of BiS-BiS Nodes can be used. For example, with a virtualization of two BiS-BiS nodes with different set of available NF types one can introduce cost diversity by providing low cost services in one of the BiS-BiS and value added services at a higher cost in the other BiS-BiS. In summary, with the new layer at the DC we created a powerful tool to enable NFC in DCs.

IV. Conclusions

In the context of the UNIFY architecture we designed and evaluated a method to deploy NFC in DCs. We analyzed how an SDN domain and a DC can inter-work to provide bump-in-the-wire middlebox services.

We showed that our NF-FG model is a powerful abstraction and can be used to express complex policies in different environments.

We designed and evaluated a UNIFY native virtualization for DC domains, with joint compute and network control API. Based on our analysis, it is not only possible to retain the business boundaries between the DC and the overarching domain but also possible to effectively control resource allocation. Furthermore, the UNIFY framework enriches us with powerful joint compute and network virtualization, definition of arbitrary topologies and resource slices to be presented to clients for their policy control.

We believe, that our DC virtualization and control framework paves the way for telco grade NFC deployment.

Acknowledgment

This work was conducted within the framework of the FP7 UNIFY project, which is partially funded by the Commission of the European Union. The views expressed here are those of the authors only. The European Commission is not liable for any use that may be made of the information in this document. The authors are thankful to the UNIFY team for the inspiring discussions.

References

[1] R. Szabo, B. Sonkoly, M. Kind *et al.*, "D2.2: Final Architecture," UNIFY Project, Deliverable 2.2, Nov. 2014. Available here

[2] 5G-PPP Association, "Contractual Arrangement: Setting up a Public-Private Partnership in the Area of Advance 5G Network Infrastructure for the Future Internet between the European Union and the 5G Infrastructure Association," Dec. 2013. Available here

[3] UNIFY, "Use Cases and Initial Architecture," UNIFY Project, Deliverable 2.1, Aug. 2014. Available here

[4] R. Szabo, A. Csaszar, K. Pentikousis *et al.*, "Unifying carrier and cloud networks: Problem statement and challenges," Working Draft, IETF Secretariat, Internet-Draft draft-unify-nfvrg-challenges-01, March 2015, http://www.ietf.org/internet-drafts/draft-unify-nfvrg-challenges-01.txt. Available here

[5] ETSI, "White Paper: Network Functions Virtualisation (NFV)," 2013. Available here

[6] ONF, "Open networking foundation," 2014. Available here

[7] The OpenStack project, "Openstack cloud software," http://openstack.org, 2014.

[8] ETSI network functions virtualisation industry specification group, proofs of concept. Available here

[9] IETF, "Service Function Chaining (sfc) Working Group," online. Available here

[10] J. Halpern and C. Pignataro, "Service function chaining (sfc) architecture," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-sfc-architecture-07, March 2015. Available here

[11] ETSI, "Architectural framework," ETSI, Group Specification v1.1.1, Oct. 2013. Available here

[12] ——, "Network Function Virtualization (NFV) Management and Orchestration," ETSI, Group Specification V0.6.1. (DRAFT), Jul. 2014. Available here

[13] OpenStack, *OpenStack Networking (neutron): APIv2.0 and Extensions Reference*, OpenStack, Mar. 2015. Available here

[14] Open Networking Forum (ONF), "SDN architecture," ONF, TR (Technical Reference), non-normative, type 2 SDN ARCH 1.0 06062014, Jun. 2014. Available here

[15] R. Szabo, M. Kind, F.-J. Westphal *et al.*, "Elastic network functions: opportunities and challenges," *Network, IEEE*, vol. 29, no. 3, pp. 15–21, May 2015. Available here

[16] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 24–24. Available here

[17] A. Gember, A. Krishnamurthy, S. S. John *et al.*, "Stratos: A network-aware orchestration layer for middleboxes in the cloud," *CoRR*, vol. abs/1305.0209, 2013. Available here

[18] T. Benson, A. Akella, A. Shaikh, and S. Sahu, "Cloudnaas: A cloud networking platform for enterprise applications," in *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, ser. SOCC '11. New York, NY, USA: ACM, 2011, pp. 8:1–8:13. Available here

[19] A. Gember-Jacobson, R. Viswanathan, C. Prakash *et al.*, "Opennf: Enabling innovation in network function control," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 163–174. Available here