Towards Unified Programmability of Cloud and Carrier Infrastructure

Pontus Sköldström[1], Balázs Sonkoly[2], András Gulyás[2], Felicián Németh[2], Mario Kind[3], Fritz-Joachim Westphal[3], Wolfgang John[4], Jokin Garay[5], Eduardo Jacob[5], Dávid Jocha[6], János Elek[6], Róbert Szabó[6], Wouter Tavernier[7], George Agapiou[8], Antonio Manzalini[9], Matthias Rost[10], Nadi Sarrar[10], Stefan Schmid[11]

Author list sorted alphabetically by institution.

[1] *Acreo, Sweden;* [2] *BME, Hungary;* [3] *Deutsche Telekom AG, Germany;*

[4] *Ericsson AB, Sweden;* [5] *EHU, Spain;* [6] *ETH, Hungary;* [7] *Uni Ghent& iMinds, Belgium;*

[8] *OTE, Greece;* [9] *TI, Italy;* [10] *TUB, Germany;* [11] *TUB & T-Labs, Germany*

***Abstract*—The rise of cloud services poses considerable challenges on the control of both cloud and carrier network infrastructures. While traditional telecom network services rely on rather static processes (often involving manual steps), the wide adoption of mobile devices including tablets, smartphones and wearables introduce previously unseen dynamics in the creation, scaling and withdrawal of new services. These phenomena require optimal flexibility in the characterization of services, as well as on the control and orchestration of both carrier and cloud infrastructure. This paper proposes a unified programmability framework addressing: the unification of network and cloud resources, the integrated control and management of cloud and network, the description for programming networked/cloud services, and the provisioning processes of these services. In addition proofs-of-concept are provided based on existing open-source control software components.**

I. Introduction

Virtualization is arguably the main innovation motor in today's Internet. Over the last years, virtualization has revamped the server business, and heralded the cloud computing era which radically changed the way we think about services and resource allocation in the Internet. But also the network is becoming more and more virtualized and software-defined.

Besides virtualization, we also witness a strong trend towards *programmability*: For example, the cloud-computing project OpenStack [1] provides a platform for infrastructure as a service (IaaS) clouds. And OpenFlow [2], the predominant incarnation of the software-defined networking (SDN) [3] paradigm, introduces programmability in the network core.

However, the different components of today's distributed systems are still managed independently, and a *unified* programmability and orchestration framework is missing. Such a unified programmability framework would not only radically simplify the network management and operation, but could also enable faster innovation and more efficient services. In general, a unified framework should cover service creation and provisioning in heterogeneous network environments, from home to enterprise networks, through aggregation and core networks to datacenters.

Today, innovation is also hindered by the lack of openness and competition. One of the crucial enablers to support openness is the definition of open interfaces (application programming interfaces, short: APIs) between all possible components and layers of the network architecture.

We argue in this paper that the desired and required flexibility in network control is and will be enabled by network and service virtualization as well as the definition of interfaces supporting some type of abstraction. Virtualization is controlled through network controllers and orchestrators (in the datacenter and network), which offer *northbound interfaces (NBIs)* to various users. Again, the possibility for innovation highly depends on the capabilities and openness of these northbound interfaces, best implemented in open APIs. These interfaces should introduce high level programmability besides policy and service descriptions.

Interfaces can be made more flexible by introducing *abstraction*, e.g., by the definition of *abstract resources*: abstract resources could include networking, computing or storage; but also *abstracted software functions*, which could include firewalls, policing, etc. In general, abstraction allows the decoupling of two independent layers and introduces flexibility and innovation on both sides of the abstraction: as long as the abstraction does not need to change.

We believe that the logical next step after the virtualization of monolithic platforms is the introduction of *modularity* and *service function chains*: network devices are turned into flexibly combinable substrates with fine granular components; these components can in turn be used by the service and network operators to offer flexible and dynamic services combinations to their customers. This may be achieved through allowing them to program (directly or indirectly) the service chains.

This paper is situated in the context of the EU project *UNIFY* [4]. The UNIFY project targets flexible service creation and provisioning in all kinds of heterogeneous network environments. In general, our goal with the introduction of UNIFY's programmability framework is to enable on-demand processing anywhere in the physically distributed network and clouds. Concretely, we aim to create a programmability framework for dynamic and fine granular service (re-)provisioning, which can hide significant part of the resource management complexity from service providers and users, hence allowing them to focus on service and application innovation similarly

to other successful models like the IP narrow waist, Android or Apple IOS.

In the UNIFY project, we will combine and extend the existing virtualization and orchestration approaches by defining abstractions, appropriate open interfaces and reference points, as well as a a multi-stage programmability process.

Our Contribution. Our objective is to create a programmability framework for dynamic and fine granular service (re-)provisioning, which can hide significant parts of the resource management complexity from service providers and users, hence allowing them to focus on service and application innovation.

Concretely, we make the following contributions:

1) *Unified programmability framework*: We present a layered architecture model applying the narrow-waist design principle. (Section III)
2) *Programmability reference points*: We define a set of interfaces which provide virtualization, resource and functional abstraction. (Section III)
3) *Reference point information models*: We define a set of graphs introducing the abstraction between the different architecture layers. (Section III)
4) *Proof of concept*: We report on different implementations of our unified programmability framework. (Section IV)

II. Related work

Cloud computing in itself is of little use if no access and network guarantees are provided. Accordingly, over the last years, several network virtualization prototypes have been built (e.g., GENI, AKARI, OFELIA, CLOUDNET), which try to fill this gap, making the network a first class citizen and introducing innovation also in the network core. A particularly interesting technology to enable virtual networks is OpenFlow. [5] Network virtualization is attractive for internet service providers (ISPs): ISPs benefit from the improved resource utilization as well as from the possibility to introduce new services. [6] While the corresponding resource allocation and virtual network embedding problems are fairly well-understood, both from an offline (e.g. [7]) as well as from an online perspective (e.g. [8]), the vision of this paper goes beyond virtual networks, and considers complex distributed services which can be managed in a unified manner.

We are not the first to argue for the introduction of a unified and programmable system abstraction. [9], [10] For example, in their "middlebox manifesto", Sekar et al. [9] 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 [9], our paper focuses more on the specific case of datacenter and carrier unification, and proposes a concrete architecture and programmability framework. ClickOs [10] is a virtualized software middlebox platform which consists of small but quickly bootable VMs.

To the best of our knowledge, so far there is no work on flexible, modular and programmable service chain architectures.

With the advent of cloud computing, many different tools to orchestrate virtualized resources (compute, storage, connectivity) have been made publicly available. Prominent examples are OpenStack, OpenDaylight or OpenNebula. OpenStack is tailored to seamlessly instantiate virtual machines (VMs) in datacenter environments. OpenDaylight on the other hand allows for provisioning virtual links between virtual network functions (VNFs), usually provided as virtual machines. Within the proposed UNIFY architecture, the orchestrator (e.g., OpenStack) would interface e.g., with OpenDaylight's northbound interface to provision network connectivity between VNFs. Note however that there is no standardized northbound protocol.

During the past decades, also several so-called southbound interfaces—the lower layer interface toward the networking elements—and protocols have been standardized to control, manage, or configure the network nodes. These protocols address different aspects. According to the SDN paradigm, the OpenFlow protocol separates the data and control plane of the network and defines an interface to control the forwarding path. Other protocols, such as SNMP or NETCONF focus on the management plane providing methods to manage, configure, monitor, etc. network devices.

III. Unified Programmability Framework

An overarching programmability framework as envisioned in this paper is still missing today [11]. In order to introduce an interdependent orchestration and programmability layer, the following two main aspects need to be taken into account:

1) Harmonization and unification of all the underlying physical and virtual resources and their capabilities.
2) Provisioning of common enablers for the service layer, with service (function) chaining, defined as logical interconnection of network functions.

The simplest realization of the above design principles is a *3-layered architecture* (see also Fig. 1), where resource orchestration and control take place in the *middle layer*, while an infrastructure layer is defined below and a service layer is defined above. The middle-layer provides the required independence, similarly to the narrow-waist design principle of IP. This orchestrator must work on abstract resources and capability types, virtual resources corresponding to network, compute and storage virtualization. In addition, the orchestration layer must not understand any higher layer logic, function, configuration, etc. The three layered model can be augmented with an application layer corresponding to the users of the services (shown as *Service + SLA*).

This design ensures that:

- The orchestration layer can operate on abstract resources and capabilities, i.e., virtual resources corresponding to network, compute and storage virtualization.
- No service function logic must be understood in the orchestrator.

- The orchestrator is technology-independent.

Additionally, in order to be able to re-use existing technologies (e.g., for network and compute), we aim to rely and integrate available "managers" into the architecture.

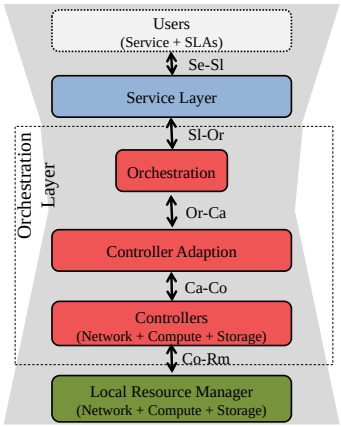

Fig. 1. Three layered programmability framework

The *service layer* turns the service chain provisioning into consumable services by defining and managing service logics; establishing programmability interfaces to users (residential, enterprise, network-network, OTT, etc.) and interacting with traditional OSS/BSS systems. The service layer is also responsible to create further service abstractions as needed toward the different users (e.g., BigSwitch topology) and to realize the necessary adaptations according to the abstraction.

The *orchestration layer* is split into three major subcomponents: the orchestrator and the controllers are interconnected with an adaptation component. The orchestration is a logically centralized function realizing the narrow-waist of the architecture. However, there could be many underlying controllers corresponding to different sub-domains (zones). The controllers are responsible to offer technology independent control interfaces and to virtualize their resources. Hence, the orchestrator collects and harmonizes virtualized resources into a global virtualized resource view at its compute, storage and networking abstraction.

It is important to note here, that the aim of the orchestrator is to collect a global resource view (as shown by the controllers) and this is offered to the service layer intact, i.e., no abstractions are done beyond harmonizing the resource views. But controllers may abstract their resources during virtualization (e.g., a multi-hop optical path may appear as a direct connection between the end points or a datacenters internal topology is hidden by the cloud controller).

The global resource view in the orchestrator consists of the following main components: forwarding elements, compute hosts, network functions, and the data plane links that connect them. All of the resources must have some associated abstract attributes (capabilities) for the resource provisioning to work, e.g., execution environment type for compute hosts. We will define the additional functionality needed in the orchestrator to be able to map service chain (re-)provisioning requests to these global resources.

The *infrastructure layer* encompasses all network, compute and storage resources providing physical means to deliver services. By exploiting suitable virtualization technologies the Infrastructure Layer can support the creation of virtual instances (networking, compute and storage) out of the physical resources.

We consider primarily four types of resources: i) our universal node [12] (based on commodity hardware), ii) SDN enabled network nodes (e.g., OpenFlow switches), iii) datacenters (e.g., OpenStack) and iv) legacy network nodes or network appliances. One of the challenges is to harmonize virtualization above these resources by proper abstraction in the orchestration layer. However, the diversity of the infrastructure (shown with the widening lower layer of the hourglass architecture) demands the support of various interfaces to the resource management agents in the infrastructure. These are handled by the various controllers present in the orchestration layer, whose role is to interact with the (local) infrastructure resource managers.

For a top-down programmability flow, we have identified the service adaptation, the orchestration, the controller adaptation, controllers and local resource managers as key components. We will describe each of the corresponding reference points between these components (see Fig. 1):

- `Se-Sl`: Users (services) and the service layer.
- `Sl-Or`: Service layer adaptation function and the orchestrator.
- `Or-Ca`: Southbound interface of the orchestrator toward an adaptation logic scoping and interfacing with various controllers.
- `Ca-Co`: Northbound interfaces of controllers.
- `Co-Rm`: Southbound interfaces of controllers.

Our primary interest is to design `Se-Sl`, `Sl-Or`, and `Or-Ca` reference points, while exploiting and interfacing to various state of the art controllers available or under development through `Ca-Co` interfaces.

A service graph (`Se-Sl` mapped to Fig. 2) describes the service requested by a user and defines: which service is provided, where the service is provided, and how successful delivery of the service is measured. The provided service is described by network functions/apps and their logical connectivity. Where the service is provided is represented by connectivity to service attachment points.

The key quality indicators (KQIs) attached to both network functions and the logical connectivity describe how the delivery is measured. The network functions/apps offered at this level to define the service may be either elementary network functions (ENF), which perform a specific function (such as a NAT, traffic classifier, transparent HTTP proxy, firewall function, etc.) or compound network functions (CNF) that internally consist of multiple ENFs. A CNF performing for example a parental control function could internally consist of a sub-graph starting with a traffic classifier followed by a HTTP proxy and firewall. In the service graph we make no distinction whether the function requested is a CNF or ENF: they are both simply represented as network functions/apps.

The network function forwarding graph (NF-FG) passed through the `Sl-Or` reference point is a translation of the service graph to match the orchestration layer, at a level of detail suitable for orchestration (see Fig. 2). The translation is done by the service layer. The NF-FG includes all the components of the service graph; network functions/apps are translated/expanded into elementary network functions (ENF) to which *known decomposition to corresponding instantiable network function types* exist in the orchestration function (for example turning the previously mentioned parental control function into three NFs with internal connectivity); SAPs are translated/expanded into endpoints, identifiers meaningful at the network level such as a certain port on a switch or a collection of IP addresses; KQIs are mapped to resource facing and measurable KPIs and requirements on the ENFs. The KQI mapping may result in insertion of additional NFs into the NF-FG for measuring certain KPIs that cannot be provided in other ways.

The difference in the information passed in the `Sl-Or` compared to the `Se-Sl` reference point is that

- (Compound) network functions must be translated and decomposed into network function types, which are known to the orchestrator for instantiation. (Note: known network function are defined in a network function information base.)
- All constraints and requirements must be formulated against compute, storage and networking resources.

The orchestration component bears with the global compute, storage and networking resource view at the corresponding abstraction level. The orchestration function breaks down the network functions defined in the NF-FG until they are instantiatable according to the given service constraints (e.g., proximity, delay, bandwidth, etc.), available resources and capabilities and operational policies (e.g., target utilization). The output of the orchestration is an instantiatable network function forwarding graph, which at the model level corresponds to a resource mapped network function forwarding graph (see Fig. 2 and the `Or-Ca` interface).

The `Ca-Co` reference point captures various northbound interfaces related to different virtualization environments. We pursue the reuse and integration of some well accepted virtualization infrastructure managers like OpenStack for data centers and OpenDaylight for software networks. Therefore, the Controller Adaptation must translate from NF-FG to various interfaces.

The `Co-Rm` reference point (not shown in Fig. 2) captures various southbound protocols of the different controllers (e.g., OpenFlow, Nova).

IV. Proofs of Concept

The infrastructure layer encompasses all the network, compute and storage resources, which can be our universal node, SDN enabled network, data center or legacy network nodes. We aim to show state-of-the-art components which can capture the essence of these resources. Moreover we introduce these technologies as important test environments for future development. Despite their limitations these components validate our unified framework's necessity.

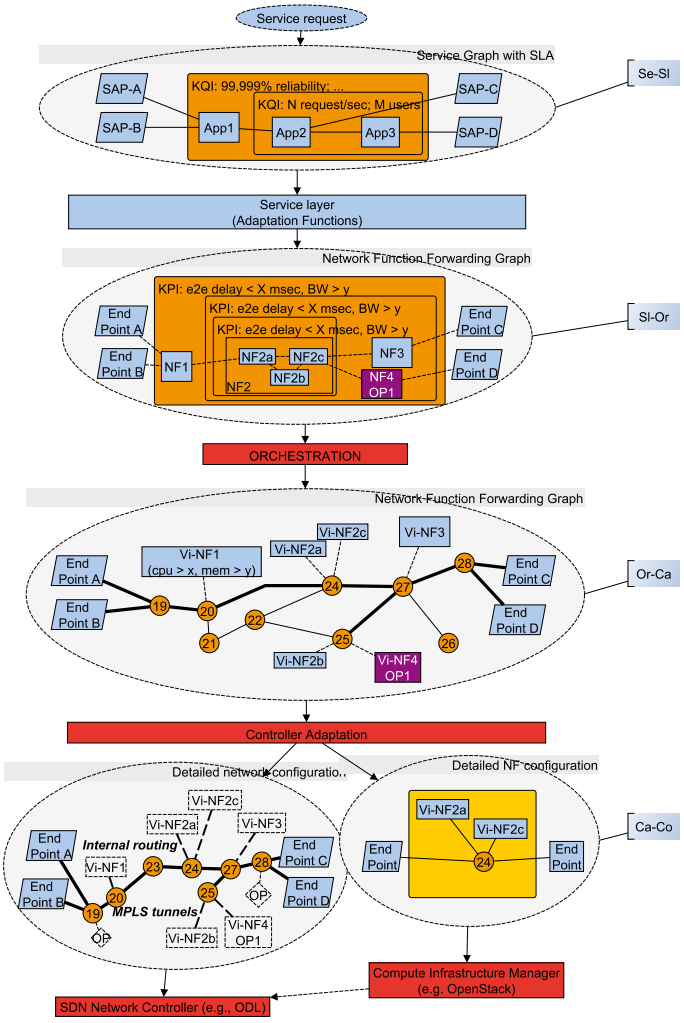

Fig. 2. Reference points' information models

We introduce *Click* as an alternative platform to our universal node which provides a set of ENFs, designed for a specific problem. Then we present a prototyping framework which enables the instantiation of service graphs built from simplified universal modes. Finally we show how an existing cloud infrastructure can be used to run some of the generic network functions and instantiate service graphs cooperatively with an SDN enabled network.

A. Click Service Function Graphs

Click [13] is a software router framework for *nix operating systems focusing on data-plane logic. Click configurations, defined by a script, are composed by interconnecting a collection of simple packet processing components.

Fig. 3 illustrates a potential Click implementation of the NF-FG for an elastic router fitting the proposed framework. The elastic router consists of 1) a control plane VNF (CP VNF) and 2) data plane VNFs (DP VNFs). The CP VNF and the DP VNFs can interact using for example an OpenFlow interface. The Click switching component presented in [14] can be used for implementing such DP VNF. Each VNF is implemented as a Click configuration consisting of one or more Click elements (*dark grey*). Each Click VNF has a particular

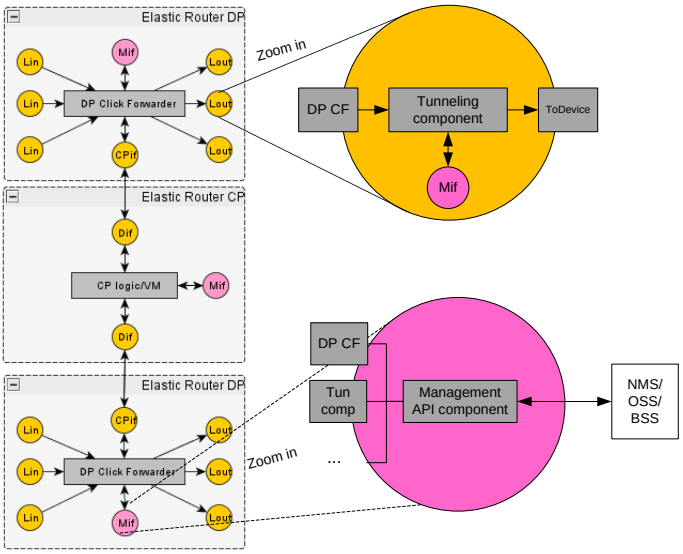

Fig. 3. NF-FG in Click

component for interfacing with the management system (*pink*), relying for example on SNMP or NETCONF. This component is made such that it can interact with the handler configuration interface of standard Click. Click VNFs interface to other VNFs or to customer endpoint interfaces (*yellow circles*) including tunneling functionality for: i) traffic steering between VNFs, and ii) service graph isolation (e.g., involving VLAN tagging). Elasticity can be achieved by adding more DP VNFs when needed. Support for forcing constraints on functions and to guarantee KQ/PIs can be implemented in several ways: in the implementation of Click elements themselves, or by adapting the interconnection of click elements. Click VNFs can be deployed on light-weight VMs in ClickOS or on regular VMs provisioned by an OpenStack cloud control framework. In the latter case, the interconnection of Click VNFs and configuration of related tunneling functionality (*yellow circle*) could be steered through a OpenStack plugin.

B. Mininet-based prototyping framework

Mininet [15] is a light-weight network emulation tool enabling rapid prototyping. It is a proper candidate to build a UNIFY development framework around that in order to make agile prototyping possible. Therefore, we have established such a framework including all layers of the architecture. The main components are shown in Fig. 4.

Our goal is to support the development of several parts of the service chaining architecture. On the one hand, the framework fosters VNF development by providing a simple, Mininet-based environment where Service Graphs, built from given VNFs, can be instantiated and tested automatically. Here, VNFs are implemented in Click and run as distinct processes with configurable isolation models (based on Linux cgroups), while the infrastructure consists of OpenFlow switches (e.g., Open vSwitch). A dedicated controller application (implemented in the POX [16] OpenFlow controller platform) is responsible for steering traffic between VNFs.

On the other hand, the framework supports the development and testing of orchestration components. Mininet is extended

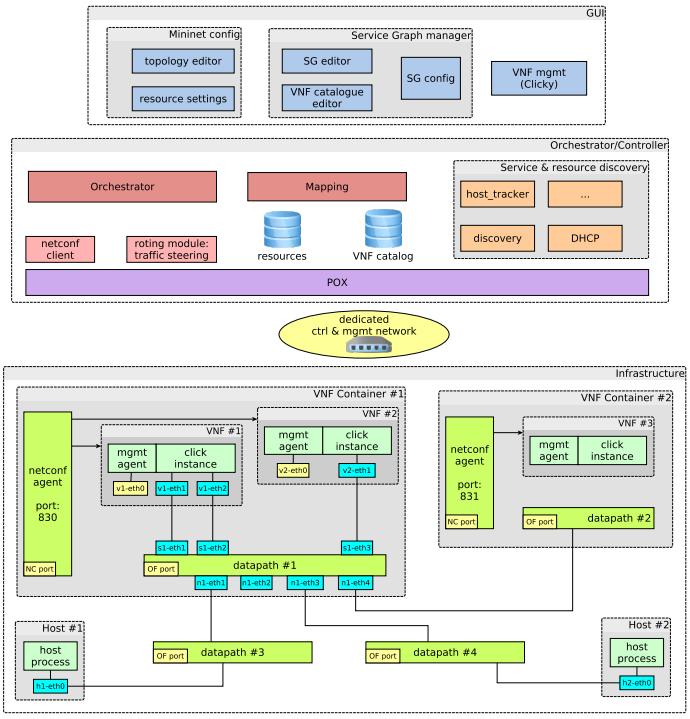

Fig. 4. Mininet-based prototyping framework

by NETCONF capability in order to support managed nodes (VNF containers, i.e., simplified Universal Nodes) hosting VNFs. The orchestrator also communicates the NETCONF protocol and it is able to start/stop VNFs on demand. The paths are handled similarly by our POX controller. On top of these, a MiniEdit based GUI can be used to describe Service Graphs (with requirements) and test topologies (resources in the network). Then, given Service Graphs are mapped to available resources dynamically, and lower-layer orchestration methods are invoked.

C. OpenStack and OpenDaylight

In this scenario we implement a simple service graph (SG) between two consumers. The SG consists of a network function (NF), which is located in a data center. Our aim with this setup is to demonstrate the possibility of using resources provided by legacy cloud infrastructure and traditional network services, by placing one or more functional parts of a service into the data center. We construct a simple virtual network as the public network between the consumers which was managed by an SDN controller. The data center is connected to this network at a single edge point and configured to run a specific image that provided a given type of network functionality.

For the actual implementation we use a traditional cloud infrastructure realized by OpenStack (OS) and a network controlled by OpenDaylight (ODL) controller. Although not necessary, we successfully boot up the OS and ODL controller in a cooperative mode, in which the cloud internal network was also controlled by ODL. The public network between the two consumer is realized by Mininet that uses OpenVSwitches (OVS) as switching elements. This approach not only gives us a lightweight test environment, but it is also good for the *Mininet-based prototyping framework*. Our OS datacenter is composed of two components: a control node for the basic OS

services and an array of compute nodes for the virtual servers in which our NF is installed. The layout of the described elements can be seen in Fig. 5.

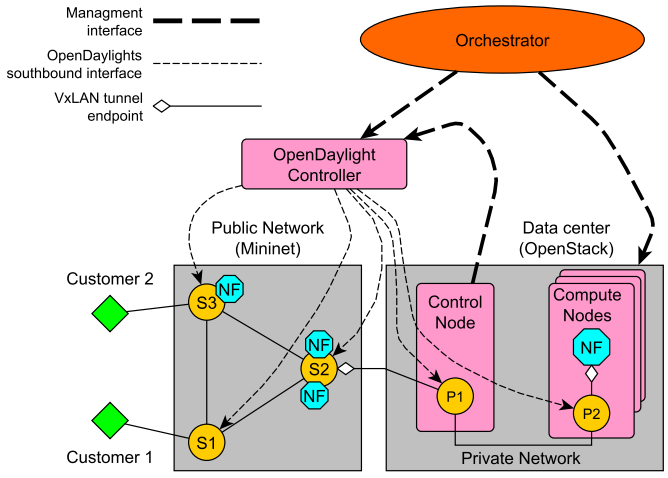

Fig. 5. OpenStack & OpenDaylight proof of concept scenario.

In the legacy cloud the virtual servers are identified by their IPs. Every traffic going to the server must be properly addressed to the specific server in Layer 3. For this reason we need some type of tunneling on the edge of the data center, more specifically the switch S2 in Fig. 5 had a VxLAN tunnel endpoint. Moreover, the NF in the data center must be aware of this tunnel and must have an appropriate VxLAN tunnel endpoint. This is the main complication caused by the legacy nature of the scenario.

After the creation of the basic setup, the NF instantiation is as follows:

1) Chose suitable server image for the network function.
2) Instruct OS to boot up a new virtual server with the image.
3) Allocate an IP address for the virtual server through OS.
4) OS instructs ODL to configure data center network for the virtual server.
5) Create a VxLAN tunnel endpoint in switch S2 and also in the network function.
6) Instruct the ODL controller to install appropriate rules in the public network switches for traffic steering.

In these steps we as the orchestrator use an interface to the OS and another to the ODL. Additionally for the VxLAN tunnel creation we need a third interface to the S2 switch and a fourth to the network function. However the last two interfaces are not necessary, since the S2 switch can be configured with ODL through the OVSDB protocol and the network function with OS through boot up scripts.

With this setup we show that a current cloud solution can be used with a popular network controller to deploy network functions and instantiate service graphs, assuming an appropriate orchestration manages them.

V. Conclusion

We understand our paper as a first step toward more flexible and programmable distributed services which can be managed in a unified manner. We have presented the main concepts behind our Unified Programmability Framework, and sketched prototypes that demonstrate the feasibility of the approach, by showing how existing networking hardware, open source software and open interfaces can be combined to realize flexible network function forwarding graphs.

Acknowledgment

This work was conducted within the framework of the FP7 UNIFY project, which is partially funded by the Commission of the European Union.

References

[1] "Openstack: Open source cloud computing software," 2014. [Online]. Available: https://www.openstack.org/

[2] "Openflow switch specification 1.4.0," 2013. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf

[3] "Software-defined networking: The new norm for networks (onf white paper)," 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[4] A. Csaszar, W. John, M. Kind, C. Meirosu, G. Pongracz, D. Staessens, A. Takacs, and F.-J. Westphal, "Unifying cloud and carrier network: Eu fp7 project unify," in *Utility and Cloud Computing (UCC), 2013 IEEE/ACM 6th International Conference on*, 2013.

[5] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," *Internet Computing, IEEE*, no. 99, 2012.

[6] G. Schaffrath, C. Werle, P. Papadimitriou, A. Feldmann, R. Bless, A. Greenhalgh, A. Wundsam, M. Kind, O. Maennel, and L. Mathy, "Network virtualization architecture: Proposal and initial prototype," in *Proc. ACM SIGCOMM VISA*, 2009.

[7] G. Schaffrath, S. Schmid, and A. Feldmann, "Optimizing long-lived cloudnets with migrations," in *Proc. 5th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, 2012.

[8] G. Even, M. Medina, G. Schaffrath, and S. Schmid, "Competitive and deterministic embeddings of virtual networks," in *Proc. 13th International Conference on Distributed Computing and Networking (ICDCN)*, 2012.

[9] V. Sekar, S. Ratnasamy, M. K. Reiter, N. Egi, and G. Shi, "The middlebox manifesto: Enabling innovation in middlebox deployment," in *Proc. 10th ACM Workshop on Hot Topics in Networks (HotNets)*, 2011, pp. 21:1–21:6.

[10] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in *Proc. USENIX NSDI*, 2014.

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

[12] U. project D5.1, "Universal node functional specification and use case requirements on data plane," 2014.

[13] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, pp. 263–297, 2000.

[14] Y. Mundada, R. Sherwood, and N. Feamster, "An openflow switch element for click," in *in Symposium on Click Modular Router*. Citeseer, 2009.

[15] "Mininet, an instant virtual network," 2014. [Online]. Available: http://mininet.org/

[16] "The POX controller," 2014. [Online]. Available: https://github.com/noxrepo/pox