NERV: A Constraint-Free Network Resources Manager for Virtualized Environments

Mauro Moura, Flávio Silva, Pedro Frosi
Faculty of Computing
Federal University of Uberlândia, Brazil
Email: {mauro.moura, flavio, pfrosi}@ufu.br

Rui Aguiar
Instituto de Telecomunicações
Universidade de Aveiro, Portugal
Email: ruilaa@ua.pt

***Abstract*—In this work, we present NERV, a constraint free network resource manager for virtualized environments capable of supporting multiple network architectures simultaneously. Using a protocol agnostic forwarding element, programmed with the P4 language, and the design principles of MicroServices architectures, we built a framework composed of small applications, each one responsible for manipulating a group of resources of the network, allowing the implementation of different policies to manage systems with different protocol stacks and control strategies. Also, we promote the reuse of these applications creating a Standard API that expose generic functions to manipulate network resources, permitting the combination of low-level applications to compose high-level network manipulation services.**

1. Introduction

Cloud computing platforms have received much attention during recent years. Platforms like OpenStack are being used in production environments helping companies to reduce both OPEX and CAPEX. Besides the broad industry adoption of these platforms, there is still a challenge to properly use innovative network technologies in cloud environments. Most of the current solutions considered are heavily tied to the IP protocol, and even the ones that are more flexible, force the network developer to re-implement functionality due to incompatibility with current interfaces, what makes this process cumbersome and time-consuming.

In this work, we propose an architecture of a network resource manager that allows the creation and manipulation of networks with different protocols stacks and different control policies over a cloud computing environment, through the use of a standard extensible API. That way, it is possible to have on the same machine multiple tenants using different network protocol architectures to communicate with tenants into another physical machine. Also, by using a common API for the components of the platform, we promote the reuse of network functions and the adaptation of the platform to be deployed in multiple environments. Given this scenario, the aim of the platform is to be flexible enough to allow the simultaneous use of disruptive network technologies in a cloud environment and promote the reuse of common services needed by different network architectures (such as the multiple Future Internet proposed under research) [1].

This work is organized as follows: Section 2 presents some related work and shows the contribution of NERV proposal. Section 3 presents the NERV architecture and describes is main components and services. Section 4 presents how NERV was realized. Section 5 details the experimental evaluation and discusses the results and, finally, Section 6 presents some concluding remarks and future work.

2. Related Work

OpenStack [2] is an open-source cloud computing platform capable of operating compute, storage and network resources from a pool of servers, creating an abstraction over the hardware infrastructure for the creation of virtual resources on a self-service and on-demand manner. OpenStack has a modular architecture, where each module is responsible for a distinct function in the cloud environment.

Since release Folsom of OpenStack, the default module responsible for offering network connectivity is called Neutron. It provides an API that allows OpenStack users to manage the network resources from the virtualized environment. Neutron also offer a variety of embedded network services, like Firewall as a Service (FaaS) and Load Balancer as a Service (LaaS). Like OpenStack itself, it also has a modular infrastructure and is composed of plugins and agents. Each plugin has its way of materializing the requests made through the Neutron API, being through the manipulation of virtual or physical resources.

Since release Havana, OpenStack adopted the Modular Layer 2 (ML2) as the default Neutron plugin. It is a plugin that allows the use of multiple Layer 2 technologies on the same virtualized infrastructure through the utilization of drivers. There are two categories of drivers supported by ML2, Type Drivers, and Mechanism Drivers. A Type Driver dictates how to realize an OpenStack network. It can be configured as Flat, Virtual LAN (VLAN), Virtual Extensible LAN (VXLAN) and Generic Routing Encapsulation (GRE). Mechanism Drivers specify what technology will be used to access these network types. Examples of Mechanism

978-1-5386-1629-1/17/$31.00 ©2017 IEEE

Drivers are OpenvSwitch, Linux bridge, Single Root I/O virtualization (SR-IOV), and OpenDaylight.

An alternative method of virtualization of infrastructure is the use of containers in the place of virtual machines. Containers offer virtualization at the Operating System level, where multiples containers share the kernel of a hosting Operating System. This fact makes containers a more lightweight approach when compared to virtual machines, since there is not hardware virtualization [3] [4]. Besides containers being implemented using software already present for a long time in the Linux Operating System, like network namespaces, chroot and cgroups, their use has been popularized by Docker [5], a tool that offers a user-friendly interface for the manipulation of containers.

When working with containers in a virtualized environments, there are multiple solutions to provide network connectivity between containers.

Calico [6] is a network virtualization solution that can work both with OpenStack and Docker. Calico considers itself a pure Layer 3 virtualization solution. It uses one agent called Felix into each physical node in conjunction with BIRD [7] as a Border Gateway Protocol (BGP) client. Through the use of Ip tables, Calico creates virtual networks manipulating the routing tables of each node. The traffic is sent direct to the destination tenant, being independent of the interconnect network. In [8] is discussed how Service-Oriented architecture (SOA) could support the convergence between Networking and Cloud Computing and how it is already used to enable network virtualization in telecommunications. In [9] and [10] is presented a SDN-enabled Network as a Service (NaaS) Architecture. It also utilizes many concepts of the SOA architecture, like as service broker and an API, called network exposition layer. In [11] is presented a Framework of a NaaS Platform, where the focus is to provide to the cloud tenants a more detailed view of the network. According to the paper, this could provide better resource usage from applications that have traffic patterns and requirements not well attended by current technology.

For this work, we focus on a particular use case, that is running over a virtualized infrastructure multiple heterogeneous virtual networks, including ones that not use the IP protocol. We want to be able to run virtual networks with different protocol stacks and control policies, independent of their localization in the virtualized infrastructure. In current platforms, this scenario is achieved through the creation of plugins or drivers, like in the case of OpenStack, or with the rewrite of the whole application responsible for the network in the cloud platform. We are aware of the advantages inherent of the employment of the IP protocol in these platforms. Being a mature protocol, IP provides a production-ready network environment, allowing cloud platforms to attend a broad range of use-cases and be compatible with the today current standards of the Internet. But this approach also restrains the experimentation of different network architectures, that could better suit cloud environments networks in certain situations.

We provide a platform that has native support to multiple network architectures, and that do not assume the use of an IP network.

Besides all the flexibility offered by current technologies, our new model of network virtualization capable of accelerating the deployment of innovative network architectures over the cloud.This framework receives the name of NERV, an acronym of NEtwork Resource manager for Virtualized environments. We adopt the use of a standard API for the manipulation of network resources and a set of services that supports the coexistence of multiple network architectures over the same virtualized infrastructure, even in the same physical node. OpenStack can run with multiple Mechanism drivers that can provide different ways to access the network, yet is not possible do so on only one physical server. Neutron does not offer any guarantee that two different mechanism drivers can be used in the same node because it does not have any knowledge of the access technology. Also, despite it is pluggable architecture, the exchange between plugins is somewhat cumbersome. For example, to use OpenDaylight [12] as an agent of OpenStack instead of the default OpenvSwitch agent into a production environment, it is necessary to clean Neutron database of all the information about the current network.

With this, we can that consider the main difference between NERV and current proposals are:

- The use of a common API by a series of services responsible for the manipulation of network resources
- The support to multiple networks of different architectures, including those not based on IP, on the same physical machine
- Liability to change between network architectures in real time

3. Architecture

This Section describes the NERV architecture and presents its main components.

3.1. Architecture Overview

NERV design is based on the interaction between small applications called Networking Services, each one responsible for managing a well-defined aspect of the network through the use of a standard API. With this, we expect to abstract the internal features of each service and promote the reuse of the network functions offered by each service. To enable the access of these services to the external network, we use a Network Virtualization Edge, one application that acts as an intermediary between the networks created by the Networking Service and the external network. The interaction between the Network Operator and the platform is done through the use of an API-Gateway, an application that has knowledge of the Networking Services that compose the platform. Finally, we also have a set of Auxiliary Services that offers functions necessary to the operation of the platform but that are not directly related to the manipulation of network resources. Figure 1 shows a conceptual view of the components of the architecture.

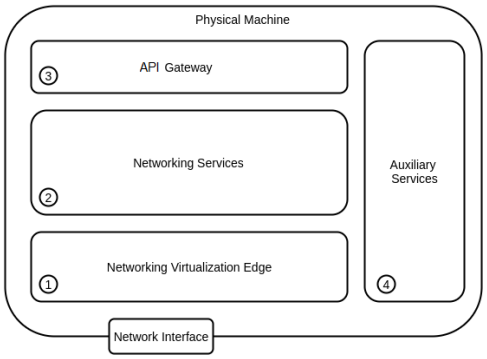

Figure 1. Conceptual view of the architecture

3.2. Networking Services

Networking Services are applications responsible for managing a type of resource in the network. A Networking Service can be responsible for manipulating virtual-bridges, while other can be responsible for handling one type of Software-defined networking (SDN) Controller. Each of these services offers a common base API that specifies the basic operations supported. This API enables the manipulation of the following resources:

- Networks
- Endpoints

In the platform context, a network is a Virtual Network created on a physical machine isolated from other virtual networks. Endpoints are points of access to the network that will be used by applications and operators to exchange data in the network. The API of each Networking Service should be capable of creating, reading, updating and deleting each one of the previously cited objects. This API enables that Networking Services be composed of a combination of other Networking Services. For example, a simple Networking Service responsible for creating IP networks can use the API of a Networking Service for the creation of virtual bridges and then later apply the necessary configurations to the virtual network. A Networking Service responsible for creating virtual SDN networks can use the services of the IP Networking Service to create both, data and control networks, and then configure the network to be attached to an SDN controller. Figure 2 shows a class diagram representing the NERV Standard API.

Each Networking service has an identifier that differentiates the different kinds of networks that are created on the same machine. This identifier can express what kind of technology is employed by the network, the paradigms adopted in the network construction or even the network architecture. In NERV, these identifiers are called Archetypes. IP, SDN, eXpressive Internet Architecture (XIA) [13], and Entity Title Architecture (ETArch) [14] are examples of names that can be used as Archetypes.

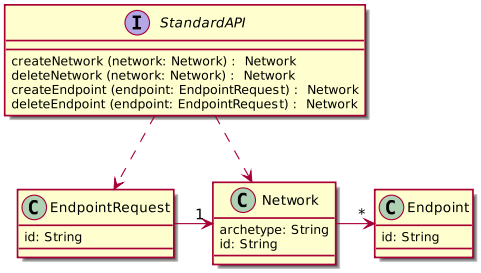

Figure 2. NERV Standard API Class Diagram

3.3. Network Virtualization Edge

This application is responsible for offering a connection between the virtual networks managed by the Networking Services to the external network. It abstracts the topology of the physical infrastructure, delivering to the internal networks just endpoints that they see as physical machine interfaces, but in reality are the ingress ports offered by the Network Virtualization Edge. Figure 3 shows how the Network Virtualization Edge interacts with the virtual networks and the physical machine Network Interface. There are two kinds of ports that are enabled in each Network Virtualization Edge:

- **ingress_ports:** These are the ports exposed to the internal networks managed by a Networking Service of a certain Archetype, being that every traffic in and out of every network from that archetype passes by that port. The number of ports needed by each Archetype depends on the network architecture. Conventional networks may only need one ingress port, while SDN networks will need two ports, one for the data plane and another for the control plane.
- **egress_ports:** These are responsible for forwarding the traffic of every ingress_port to outside the physical machine. The outgoing data already leaves this interface with the appropriate headers in a way that it will be delivered by the physical infrastructure into the physical node where resides the virtual network with the destination endpoint.

3.4. API Gateway

This module is responsible for translating different forms of request to the REST API of the core module. For example, it could be used to provide an interface between a cloud virtualization platform, like OpenStack, a container engine, like Kubernetes, or even a terminal client. The API Gateway is an optional component. In the case it is absent, clients can make requests direct to the Core module API, but this way require adaptation of the invoking application. Also, a node can have multiple API Gateways modules, resulting in a node that can have different clients managing the network resources.

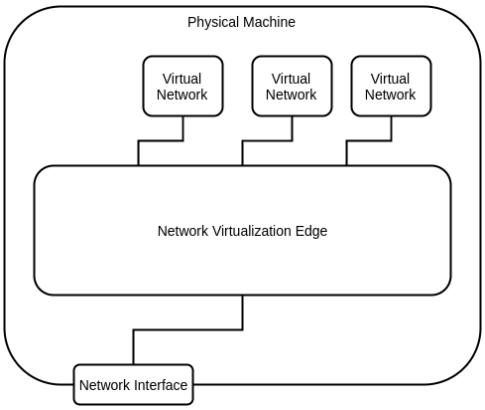

Figure 3. Network Virtualization Edge

3.5. Auxiliary Services

These applications are services not directly related to the manipulation of network resources but are needed for maintaining the system operation. Every module offers an interface or API that could be accessed by any other module on the system. The API permits that the implementation of each module be private to that module, allowing that the substitution of a module with the same interface be transparent to the overall system. Next, are described the modules that realized the architecture and given examples of how this was deployed in our tests.

4. Testbed Realization

To implement the proposed architecture, we used Node.js [15], a Javascript runtime that uses an asynchronous, event-driven model to build lightweight Javascript applications that run on servers. Below are the details of how each component of the platform was implemented.

4.1. Networking Services

Networking services are applications responsible for managing a type of network resource. Each application exposed a REST service with the methods specified in the base API of a networking service. Some services also extended the API to allow more detailed manipulation of network resources.

4.1.1. Linux-Utils. This service was responsible for manipulating Linux programs like ip [16] and netns [17] to manipulate network namespaces. This service also managed the life cycle of virtual-bridges and OpenvSwitch instances.

4.1.2. IP. The IP network made use of the Linux-Utils service to create a virtual bridge in the OS for every create-network request received, and add IP specific properties to the network, like the range of IPs of each network and the use of a DHCP server.

4.1.3. SDN. This application used the services of the IP Networking Service to create two IP networks, one for the control plane using a Linux Bridge and other for the data plane using an OpenvSwitch instance. (Later, the OpenvSwitch instance was attached to an OpenDayLight controller running on the namespace of the SDN network).

4.1.4. ETArch. ETArch [14] is a clean-slate network architecture that uses an Entity Title model to address entities that communicate over a logical bus. This logical bus is called a workspace. A workspace is independent of the underlying topology and specifies the requirements of the entities it encompasses. Entities are any element present in the network, being those applications, network devices, or Operating Systems. Entities can create and participate in workspaces and are identified by unique identifiers called Titles. Like SDN, ETArch also separates the network into a control and a data plane. In the control plane, ETArch uses a Domain Title Service (DTS) to manage the life cycle of Entities and Workspaces. It guarantees that the requirements and security policies specified by the entity that created the Workspace are met. A DTS is composed of applications that control one portion of the network and interact among themselves to manage entities and namespaces called Domain Title Agents. Figure 4 shows the simplified topology of an ETArch network.

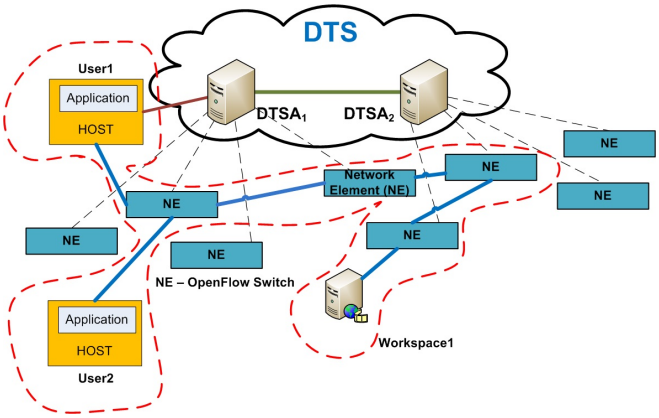

Figure 4. ETArch network Topology

The ETArch Networking Service also made use of the IP Networking Service to create both data and control networks. To establish the control of the network, one DTSA is started inside the namespace of the network and attached to an instance of OpenvSwitch.

4.2. Network Virtualization Edge

The Network Virtualization Edge was implement using a P4 enabled software switch controlled by a Node.js program. P4 is a programming language for network elements independent of protocol. With if, is possible to program the behavior of the switch, from the fields present in each directive to the actions executed based on math-action tables.

The software switch used was the Behavioral Model Version 2 (BMV2). For each network created on the platform, a VLAN ID would be associated with the network, in a way that every traffic originated from the network would be directed to the external network with the related VLAN ID. Also, every frame ingressing to the machine would have the VLAN ID and would be directed to the appropriate network.

4.3. Auxiliary Services

Two such services were developed:

4.3.1. State manager. To manage the state of the applications we used a MongoDB [18] node shared by both instances of the application. MongoDB is a distributed, open-source document-oriented database that store JSON-like documents.

4.3.2. Bootstraper. The Bootstrapper was implemented to manage the state of the application, being responsible for the following functions:

- Manage the initialization and shutdown of Networking Services, the Network Virtualization Edge, and other auxiliary services
- Set all configuration variables
- Manage all the platform logs

4.4. Testbed Description

To realize the experiments, we utilized two physical Machines, both with Ubuntu 14.04. A switch configured with two trunk interfaces connected both physical machines. To access NERV capabilities three networks were created, one using the IP Archetype, one with the SDN Archetype and another with the ETArch Archetype, all of then spanning endpoints in both servers. The traffic in the networks created using the IP, and SDN Archetypes was generated using a virtual machine with a VLC server installed. Other virtual machines connected on the same network then requested the stream of the video file via HTTP, being that clients were created in both physicals compute nodes. Figure 5 shows the logical view of the network created using the IP Archetype, while Figure 6 shows an SDN network created by NERV.

To generate traffic for the ETArch network, it was used a chat application where entities were able to create workspaces and exchange message among the members o the workspace. Figure 7 shows the design of the ETArch network used in the experiments.

5. Results and Discussion

To access the performance of the platform, tests were made using iperf [19] to measure the impact that the layers of virtualization had on the traffic of the networks created with NERV. Figure 8 shows the throughput graph of the test. While testing the link with TCP traffic without virtualization, the test indicates a bandwidth of 93.9 Mbits/sec. With

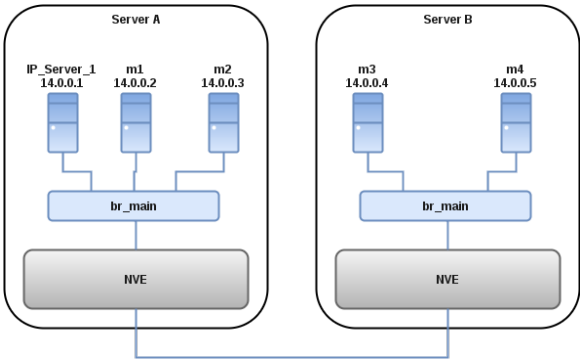

Figure 5. Logical view of an IP network

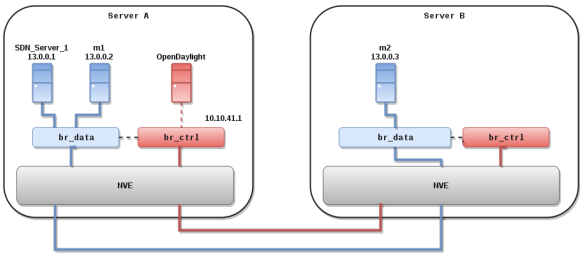

Figure 6. Logical view of an SDN network

iperf running on a network created with the IP Archetype, the bandwidth obtained was reduced to 40.1 Mbits/sec. With UDP traffic, without virtualization, the network presented a Jitter of 0.008 ms, and a total of 476190 Datagrams were transmitted in 60 seconds without any loss of packets, giving a total of 668 Mbytes transmitted. In the virtual network, the Jitter increased to 0.242 ms and a total of 204082 Datagrams where transmitted, being that 2125 Datagrams were lost, a 0.96% loss. Also, only 283 Mbytes were transmitted during the 60 seconds test.

While NERV uses a standard and simple API to easy the process of creating new network services and promote the reuse of functions, it can turn the management of the network more complex than current solutions to manage network resources in virtualized environments. This is acceptable by the fact that the objective of the platform is to be a testbed for experimental technology, not a complete solution for production environments. Regarding the performance results, the poor results of NERV are mainly

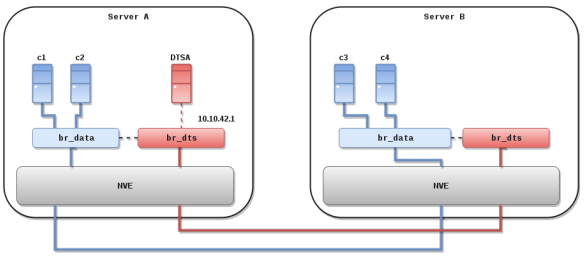

Figure 7. Logical view of an ETArch network

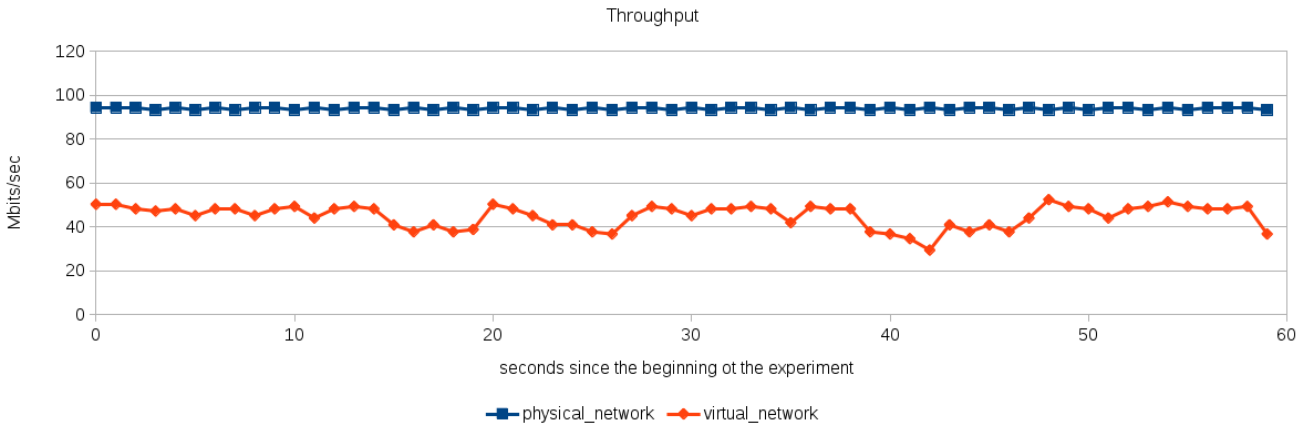

Figure 8. Throughput Graph

attributed to the use of the BMV2 as the software switch of the NVE. BMV2 is a software switch implemented in user space, developed primarily to test the functionalities and new features of P4. The overhead caused by BMV2 could be reduced by the use of the compiler presented at [20], a P4 compiler that uses Intel DPDK [21] to produce a high-performance software switch, but that was not available during the development phase of NERV. It is important to notice that NERV by itself does not intervene in the data-plane of the networks, only manages how networks are created. Any impacts on performance are dependent on the components used to integrate the network.

6. Concluding Remarks and Future Work

Management of network resources in virtualized environments is usually heavily dependent of the IP protocol, which becomes a barrier to our objective: to run disruptive network architectures over the same physical machine. Furthermore, while complete cloud systems are modular, the System responsible for managing network resources is monolithic. Both aspects turns the evolution of these systems complicated and difficult the integration of disruptive technologies. Following this rationality, we designed an architecture capable of running multiple network architectures through the use of a Protocol Independent Forwarding element as the Network Virtualization Edge of each physical machine. With this element serving as a bridge between the external network and the virtual networks created on our platform, our proposal is entirely agnostic to the protocol used by the virtual networks and the resources manipulated. If one specific network has requirements not yet supported by our platform, these can be implemented directly in the forwarding element inside the Network Virtualization Edge using the P4 language. We also separated our platform in a group of independent applications, each one responsible for one aspect of the network, following the design principles of the MicroServices architecture. This approach has as objective to promote the reuse of these applications for the construction of high-level Networking Services and ease the integration of new technologies in the platform. The results proved the realizability of this design.

For future experiments, the use of a P4 forwarding element instead of a common L2 switch will enable implementation of a different algorithm to enable the forwarding of traffic between physical machines with NERV deployed, completely freeing the platform of the use of the IP protocol. Also the integration of NERV with a complete cloud virtualization platform, like OpenStack, to completely provide a complete protocol independent cloud solution. During the tests, commands to manipulate the network resources were sent to both machines. In next experiments, should be developed an auxiliary service capable of receiving commands and deciding which application should execute the command, fully exploring NERV capabilities.

Acknowledgments

This work was conducted inside the project Arquitetura Adaptável para Redes Convergentes (A2RCON), funded by Coordination for the Improvement of Higher Education Personnel (CAPES) in Brazil under grant agreement number 88881.062211/2014-01 and the National Council for Scientific and Technological Development (CNPq). In the Scope of R&D 50008, this work is financed partially by the applicable financial framework (FTCMEC) through Portuguese funds, and when applicable co-funded by FEDER-PT2020 (with reference number UID/EEA/50008/2013).

References

[1] J. Pan, S. Paul, and R. Jain, "A survey of the research on future internet architectures," *IEEE Communications Magazine*, vol. 49, no. 7, pp. 26–36, July 2011.

[2] 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, 2012.

[3] P. R. Desai, "A survey of performance comparison between virtual machines and containers," *ijcseonline. org*, 2016.

[4] S. J. Vaughan-Nichols, "New approach to virtualization is a lightweight," *Computer*, vol. 39, no. 11, 2006.

[5] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[6] TiGERA, "A new kind of virtual network," 2016. [Online]. Available: https://www.projectcalico.org/

[7] O. Filip, "Bird internet routing daemon."

[8] Q. Duan, Y. Yan, and A. V. Vasilakos, "A survey on service-oriented network virtualization toward convergence of networking and cloud computing," *IEEE Transactions on Network and Service Management*, vol. 9, no. 4, pp. 373–392, 2012.

[9] A. Boubendir, E. Bertin, and N. Simoni, "Naas architecture through sdn-enabled nfv: Network openness towards web communication service providers," in *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2016, pp. 722–726.

[10] ——, "On-demand dynamic network service deployment over naas architecture," in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, April 2016, pp. 1023–1024.

[11] P. Costa, M. Migliavacca, P. Pietzuch, and A. L. Wolf, "Naas: Network-as-a-service in the cloud," in *Presented as part of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, 2012.

[12] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, 2014.

[13] A. Anand, F. Dogar, D. Han, B. Li, H. Lim, M. Machado, W. Wu, A. Akella, D. G. Andersen, J. W. Byers, S. Seshan, and P. Steenkiste, "Xia: An architecture for an evolvable and trustworthy internet," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, ser. HotNets-X. New York, NY, USA: ACM, 2011, pp. 2:1–2:6. [Online]. Available: http://doi.acm.org/10.1145/2070562.2070564

[14] F. Silva, J. Castillo-Lema, A. Neto, F. Silva, P. Rosa, D. Corujo, C. Guimarães, and R. Aguiar, "Entity title architecture extensions towards advanced quality-oriented mobility control capabilities," in *2014 IEEE Symposium on Computers and Communications (ISCC)*, June 2014, pp. 1–6.

[15] S. Tilkov and S. Vinoski, "Node. js: Using javascript to build high-performance network programs," *IEEE Internet Computing*, vol. 14, no. 6, p. 80, 2010.

[16] "ip(8) - linux manual page," http://man7.org/linux/man-pages/man8/ip.8.html, (Accessed on 11/09/2016).

[17] "ip-netns(8) - linux manual page," http://man7.org/linux/man-pages/man8/ip-netns.8.html, (Accessed on 11/09/2016).

[18] K. Chodorow, *MongoDB: the definitive guide*. " O'Reilly Media, Inc.", 2013.

[19] V. GUEANT, "iperf - the ultimate speed test tool for tcp, udp and sctptest the limits of your network internet neutrality test," 2016. [Online]. Available: https://iperf.fr/

[20] S. Laki, D. Horpácsi, P. Vörös, R. Kitlei, D. Leskó, and M. Tejfel, "High speed packet forwarding compiled from protocol independent data plane specifications," in *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM, 2016, pp. 629–630.

[21] INTEL, "Dpdk," 2016. [Online]. Available: http://dpdk.org/