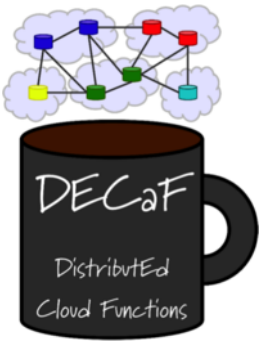

Project Plan

Authors

Sergio Djoum
Thorsten Götte
Kristian Hinnenthal
Mihir Janaj
Andreas Krakau
Ashish Rohilla
Tarun Sarkar
Sven Uthe
Guruprasad Zapate

Supervisors

Sevil Mehraghdam
Manuel Peuster
Holger Karl

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

— Department of Computer Science —
Computer Networks Research Group

Version	Description	Date
v1.0	Initial Document	*June 07, 2015*

Table 1: Version History

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 MOTIVATION AND PROBLEM DEFINITION

1.1 MOTIVATION

Network operators currently have infrastructure to provide different Network Service (NS) for mobile communication, internet access or Virtual Private Network services to customers [14]. But there is a substantial development in evolution of hardware technology, and to keep up with the market requirements of providing the latest services first, the Network operators seek an alternative to the conventional approach of investing for every new Network service they want to introduce to their customers. As new services are introduced into the market, additional overhead costs for deploying these NS are needed, eg. middleboxes and their storage, switches, energy costs and on-demand deployment costs (time taken to procure-design-integrate-deploy).

In such cases, Network Function Virtualisation (NFV) proves to be a more efficient and viable option for the service providers as it inculcates standard IT Virtualisation by separating hardware from the software, reducing costs incurred by Network operators for individual Services, shared resource utilization for multiple operators on industry standard hardware and minimizing time to market for delivering NS. NFV is co-related to Software Defined Networking (SDN), but their implementation can be independent of each other, although we can use SDN for NFV as it facilitates network abstractions and vice versa as NFV provides a platform for the SDN to run on [6] (refer to chapter 4 section 4.2).

1.2 PROBLEM DEFINITION

The Network Functions (NFs) define functional behaviours of a NS. NFs are executed on physical nodes in a distributed network and are linked with each other to complete a NS. NFs separated from the underlying hardware are known as Virtual Network Function (VNF) and the environment we obtain is a Network Function Virtualisation Infrastructure (NFVI) [12]. There is a need to manage, control and govern these NFs on a virtualised environment and our project aims to target this problem by defining a proposed orchestrator as we can see in chapter 2.

This orchestrator has to be flexible in terms of user requirements and multi tenant atmosphere for operators, scalable in terms of compute functionality of NFs and it can reuse operators support system as a

part of the NFV architecture [7]. VNFs require resources such as CPU, memory, bandwidth, storage etc. To co-ordinate these infrastructure attributes for a particular operator, we need a component to assist the orchestrator. This component has its instances for each operator domain managed by the orchestrator.

As the NFVI is geographically diverse, network operators should possess control over the VNF instances interfaces. As far as flexibility is concerned, operators need customized functionalities to manage services, hence they need to be enabled with options like plugins as we will be learning in chapter 2.

2 GOALS, USE CASES, INTERFACES

2.1 GOALS

In this section, we describe the rough goals of our project.

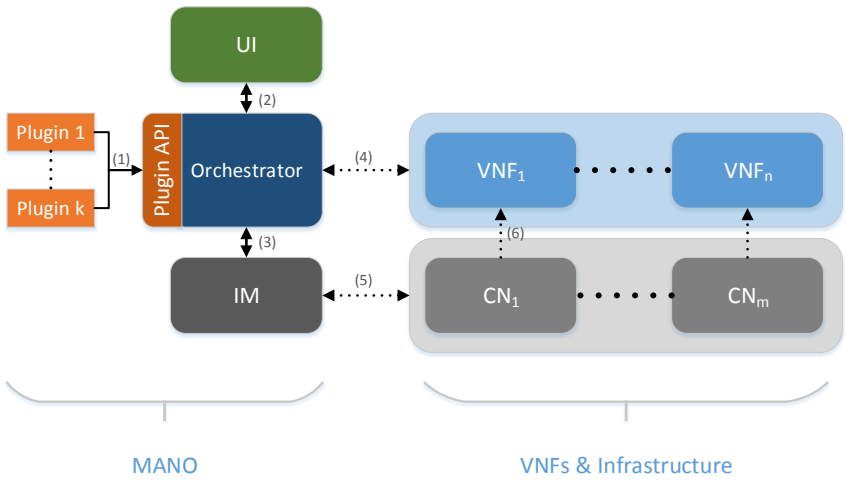

Figure 1: This figure visualizes the basic structure of our software.

Figure 1 presents a first sketch of the structure of the project. The project can be loosely divided into two components. A component that manages and orchestrates the network (*Management and orchestration*, MANO) and the other comprising of network functions and the infrastructure required for the network. The goal of the project is to build an orchestrator that provides the consumer with the ability to manage, distribute and optimize his network functions, depicted as *VNF* in the figure. The process of management of the resources and the infrastructure is abstracted in a lower layer, visualized by the gray boxes. The *infrastructure manager* (IM), that abstracts many responsibilities related to the management and monitoring of physical resources, controls the *compute nodes* (CM). The lines between compute nodes (6) abstract the virtualization layer between running network functions and the physical layer.

The functionalities provided by the orchestrator are exposed to the customer via an Application Programming Interface (API), depicted by the orange box labeled *Plugin API*. A customer can invoke the API to interact with the orchestrator through plugins, also using the API as the interface for (graphical) *user interfaces* (UI) can be possible. Hence, line (2) in the figure can either rely on the existing API or be connected

otherhow, this we will have to decide later. The interaction is bidirectional i.e, the customer can invoke functions and give specification to the orchestrator which provides diagnostic data and events to the plugin. The customer has the ability to build his custom plugin, for instance, a plugin that monitors the network and dynamically alters the service by interacting with the orchestrator. One important feature is that the orchestrator is capable of running multiple plugins of different users simultaneously. This, therefore, offers a lot of flexibility.

Another important goal is that the customer can opt to acquire different levels of service. The first level would be obtaining NFVI instances. Further he can choose to run his NFs or Network Function Chain (NFC). Overall, the customer is able to provide a plugin, using the full API functionality to acquire a complete dynamically scalable virtual network.

2.2 USE CASES

USE CASE 1: MANAGE NFVI A service provider provides Network Function Infrastructure instances to his consumer, which the latter accesses through an API in order to manage physical resources. For example, the consumer acquires infrastructure to create and run NFV instances, network connections and cloud applications.

The service provider isolates the resources amongst NFVI instances and also offers instances with varied constraints, regulations and scalability of resources.

For instance, the service provider provides NFVI instances to a consumer who doesn't possess the investment to own his physical infrastructure. The consumer with this virtual infrastructure in turn resells these resources along with his custom network services to another consumer. As a consequence, a consumer can buy NFVI instance near a large user base in order to provide faster service.

USE CASE 2: MANAGE NF The service provider provides the consumer with a selection of NFs, virtualized and/or non-virtualized, which the latter defines and configures to his requirements. The consumer cannot, in this case, control the underlying infrastructure that is made available to him, such as the amount of storage or compute infrastructure. He uses it on an expense basis and is not responsible for managing the NFV instances. The service provider is in charge of the management and must provide and maintain the expected level of service to the consumer.

For example, the consumer requires a set of NF instances with high performance needs and doesn't have the investment to maintain them. The service provider handles the infrastructure

of the NFs such that it provides service to a large number of applications and users. The service provider scales the resources to the NFs based on the usage of the service.

USE CASE 3: DEFINE NF CHAINS Giving the option for the consumer to configure the NFs and chain them according to their requirements is an important characteristic of an orchestrator. Currently, Forwarding Graphs are used to specify a chain of NFs.

A consumer can provide a total order of NFs using Forwarding Graph (FG). However, when the consumer wishes to provide a partial order of NFs or if the order of chaining is unknown or irrelevant for the functioning of the service, then the orchestrator is tasked with finding a way to produce the service.

There can be several permutations of FGs that can be created from this request. Orchestrator chooses the best that suits the request. The orchestrator can also provide a list of NFCs for the service providers to choose from.

Also, the orchestrator provides a list of NFCs for the service providers to choose from.

USE CASE 4: The service provider provides diagnostic, monitoring and notification tools to the consumer to make the best use of their service. The abilities specified in the previous use cases can be accessed via an API.

This enables the customer to provide a program as a plugin that dynamically manages NFC and other services or he can manually change his requirements using say, a web interface. As a result, the customer decides how to manage, distribute and optimize his NFC.

Actors:

USER: The customer who uses the services offered by the Cloud Provider.

3 NON-FUNCTIONAL REQUIREMENTS

In this chapter we want to define important non-functional requirements for our software. We will give an overview over some requirements and describe their meaning for our project in particular.

- *Flexibility.* Flexibility abstracts the ability to change the system according to changing requirements. Regarding our project, the orchestrator has to provide means for configuring the system according to the requirements of the customer, which might be changing. More practically, the orchestrator has to provide an API for plugins making use of all functionalities of the system. This API can be used by customers to implement their specific requirements by defining and using different plugins. The orchestrator has to provide this degree of flexibility.

- *Scalability*. One of the properties that comes in mind when discussing about networks nowadays is scalability especially in cloud-based networking. By definition, a system is considered scalable if it can adapt its throughput to varying load [40]. In our system, this can be applied not only to the overall load, but also to the demand on particular network functions. Our system does not only face the challenge of adapting the throughput of the network, but should also offer means to relocate network functions in order to improve the latency. Using virtualization is a mean to make our system more scalable. The requirement of scalability will be approached by exploiting the possibilities of virtualization. The orchestrator will have to monitor the load of the physical instances and adapt the virtual machines and network functions accordingly.

- *Agility*. Network Agility refers to the ability of the network to automatically configure itself. According to the example of adapting the varying load above, agility would describe the aspect of the system, that it is not only able to *be* reconfigured accordingly, but does that *automatically*. Hence, we need interfaces and protocols between the entities of the system, such that no external intervention is needed in this process.

- *Performance*. This term summarizes many possible aspects of systems. In our network scenario, we will basically consider throughput and latency. This especially means, that our software and virtualization has to be efficient, also the physical nodes have to be appropriately powerful.

- *Usability*. This requirement could be paraphrased as *ease of use*. As already introduced in chapter 2, our software will be used in many different scenarios and by different kinds of users. There will be interfaces to configure the system according to every use case. In particular, since not only developers are going to configure the system, the provided interfaces have to be easy to use and understandable. Although usability is not the most important requirement in our project, it should also be taken into account.

- *Reliability and Robustness.* The requirement of reliability aims at the ability of the system to function under specified conditions and for a specified time. This is a pretty weak requirement, that should be met in any case. Robustness refers to the case of unattended errors. It means that even in scenarios that differ from specified conditions, the system has to be able to perform appropriately. This is a core requirement in networks that can suffer from connection losses, congestion or server failures. The orchestrator in particular has to be able to cope with such scenarios to a certain degree.

4 TECHNOLOGIES

In this chapter we present and discuss technologies that are related to our project or can be beneficial at some point. We recap some topics we presented in the mini seminar, and also introduce some other technologies in addition. For each topic, we especially address the relevance and integration of it in our project.

4.1 OPENFLOW

Open Flow (OF) is a communication protocol for Flow-based Routing, i.e. it allows a controller to access and manipulate the Forwarding Plane of virtual and physical switches. In our reference architecture (cf. figure 1), it will be used at interface 5 as it handles communication between IM Module and Infrastructure. It was initially developed at Stanford University [35], but is now actively maintained by the Open Networking foundation (ONF)[1]. It poses to be the de-facto standard in this area and is supported by devices of numerous vendors. Hence, our Infrastructure Manager (IM) Module should not only support OF to archive widespread compatibility, but also exploit one of its main features -the pipelining mechanism- to archive a high performance. Therefore, we will give a short overview over this feature.

An OF-enabled switch has at least one, but possibly several flow tables $t_0, \ldots, t_n$. The flow tables are sequentially ordered, an incoming packet is first processed by t_0, then t_1 and so forth until t_n. In each table, a packet can only match to exactly one entry, otherwise it is dropped. The matching is based on three properties: the ingress port, the packet header and OF's own metadata for a packet. Fields and tags in the metadata can be choosen at will, but changes are lost when the packet leaves the switch.

Every entry contains a set of instructions which are applied to the packet if it matches. These instructions can be grouped into three categories: Firstly, the package header can be persistently modified. Secondly, the metadata can be changed. Lastly, there is the so-called *Action Set*, which can be altered. It contains actions on how to proceed with the packet after the last table.

When a packet went through all tables, the action set is applied. In essence, it determines whether a packet is forwarded to the next hop, the controller or dropped. However there are more sophisticated

[1] opennetworking.org

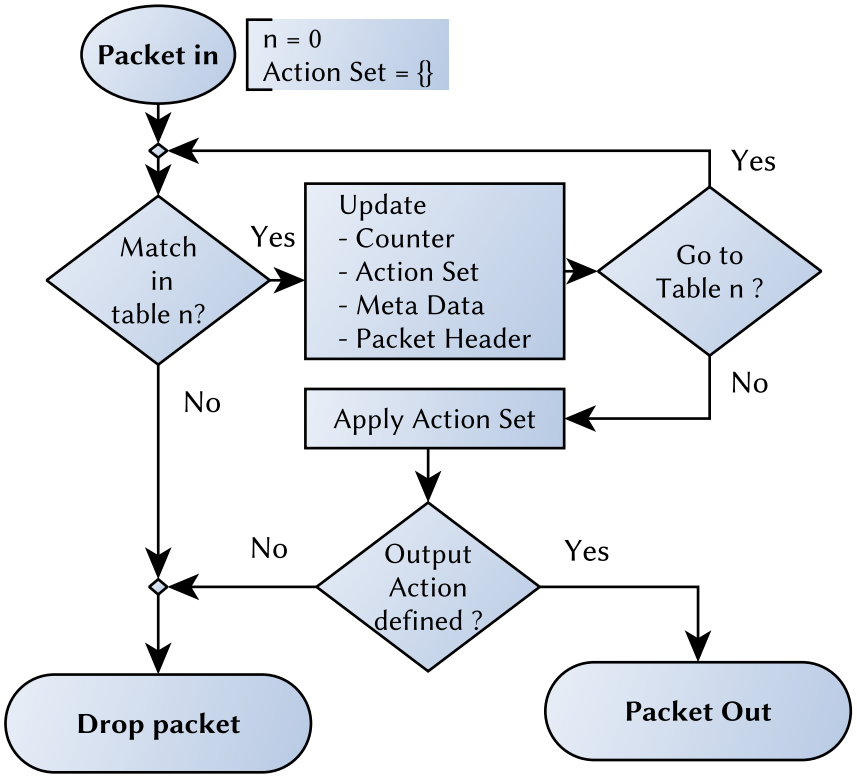

Figure 2: Flowchart of OpenFlow's matching mechanism

actions which are beyond the scope of this section. A graphical representation of this this process can be seen in figure 2.

4.2 SDN CONTROLLER

As defined in the reference architecture the use cases, our IM Module has to configure the manage packet flows and monitor network activity and the orchestrator has to manage and configure the VNFs. These are executed over the interfaces 4 and 5 in the reference architecture (cf. figure 1). For both these tasks, we will use an existing SDN Controller and extend it with the necessary functionality. In this section we will give a quick overview over the candidates.
Conceptually, an SDN Controller is the sole connection to the infrastructure and therefore a single-point-of-failure. That makes it subject to many of our non-functional requirements: It has to be *highly available*, otherwise the orchestrator cannot interact with the infrastructure. Also it must be compatible with a variety of devices and protocols in order for our orchestrator to be *flexible*. Lastly, it has to tolerate changes to the infrastructure, for example when the orchestrator allocates new virtual machines. In other words, it has to be *agile* in order for the orchestrator to be *agile*.

From existing state-of-the-art controllers, we take into considerations are *OpenDaylight*(ODL), *Floodlight*(FL) and *Ryu*, because they fulfill these requirements. All these controllers support latest version of *OpenFlow*. Internally they are component- or plugin-based, so additional protocols, e.g. to communicate with VNFs, and interfaces for our applications can be added to their core functions. All three controllers feature a module for high availability, which dynamically starts several instances of the controller to handle failures. Furthermore *ODL* and *FL* provide an extendable RESTful API, such that the orchestrator and the controller may run in different address spaces and have more independence from each other. Lastly, all Controllers feature a web-based UI, which could be incorporated in our UI Module.

4.3 NETCONF / YANG

The Network Configuration Protocol (NETCONF) is a protocol used to manage and configure network devices. Developed by the IETF in 2006, one of its goals is to lower implementation cost by reducing complexity. YANG is the corresponding data modeling language especially designed for NETCONF. Developed by the IETF in 2010, it is used to model both data and operations of the network device. The development of NETCONF and YANG was initiated, after the Internet Architecture Board (IAB) expressed the urge for a new technology to configure network devices in 2002. They also stated objectives and requirements for new protocols, from which the properties of NETCONF and YANG were derived.
NETCONF and YANG have many advantages over prior solutions such as SNMP or CIM, for example focusing on ease of use and readability. Supporting that, they both heavily rely on existing technologies such as XML, RPC or SSH. They also provide means for a secure and authenticated connection and allow users to lock configurations to ensure mutual exclusion. Another main advantage is the possibility to distinguish between configurational and state data, making clear which parameters were configured and which were obtained somehow else in the current state.
Understanding the structure of both languages is fairly easy, hence applying them in our setting should not be a problem for us. Because of this, and also the other advantages of NETCONF and YANG over other solutions, we should seriously consider using them in our project. They could be applied in the setting of configuring the VNFs, and also we could think about configuring the actual physical nodes of the network using NETCONF.

4.4 CONFIGURATION MANAGEMENT SYSTEMS

To manage thousands of similar server nodes administrators can use tools to distribute the desired settings to the nodes. Such tools, called Configuration Management System (CMS), introduce an abstract layer to specify configurations. These configurations are specified in a CMS dependent language and stored on a central master server, which then deploys them to the managed clients [48, 49]. In an environment, like the one of this project, a CMS is especially useful, because the configuration files can be managed by a version control system. This introduces the possibility of a reproducible environment [34]. Moreover a CMS like Puppet can be used to query informations about the state of a single node and expose interfaces for more advanced monitoring solutions to keep track of installation problems [34, 25].

4.5 OPEN VSWITCH

One of the goals of our project is to allow the orchestrator to dynamically create VNF. Each VNF can be in fact located in individual Virtual Machine (VM) which can be in its turn located in a hypervisor. The role of the hypervisor is to emulate a hardware platform to create virtual machine(s). However as there are many VM all running on the same hypervisor, hence on the same host machine, they will need to interconnect with each other, in other words to be nodes of a virtual network [20]. Using virtual networking, we can network virtual machines in the same way we do with physical machines and can therefore build complex networks within a single or across multiple hosts. One of the tools allowing that, is Open vSwitch. Open vSwitch is a multilayer virtual switch licensed under the open source Apache 2.0 license, giving the possibility to network administrators to manipulate the forwarding state using OF and to manage the configuration state at runtime by providing a configuration interface. It supports many Linux-based virtualization technologies such as Xen/XenServer, KVM, and VirtualBox.
Open vSwitch supports many features namely:

- *Security* by using VLAN (Virtual Local Area Network) isolation and and traffic filtering
- *QoS (Quality of Service)* by using traffic queuing and traffic shaping
- *Monitoring* by using netflow, sflow, SPAN, RSPAN
- *Automated control* by using a SDN controller which uses OF in its turn . It is not mandatory to use a SDN controller but it is mostly recommended.

Open vSwitch is the most suitable virtual switch, which can be used in our project because it is designed for production deployments and for testing purposes like vmware, vswitch and Cisco Nexus 1000V Virtual Switch and it is an open source software switch while its counterpart are licensed.

4.6 INFRASTRUCTURE MANAGEMENT FRAMEWORKS

Infrastructure management frameworks provide head-start for creating public and private clouds, that will be required to implement and test our Orchestrator. In this section, we will give a very brief overview of different prominent open-source IaaS frameworks and advantages & disadvantages over other frameworks. Detailed architecture and workings of these frameworks are out of the scope for this document.

1. *OpenStack*
   OpenStack, is an open-source project managed by OpenStack foundation. `Scalability` and `Flexibility` were the prime concern in the design of OpenStack architecture. OpenStack is collection of components where each component provides individual services and these components communicate with each other using their public APIs [10]. The prime components of OpenStack project are:
   a) `Compute` used to manage and automate pools of computer resources
   b) `Swift` scalable redundant storage system
   c) `Glance` is used to store and retrieve disk & server images
   d) `Keystone` provides central authentication and authorization

   Due to its components based design and high scalability, it is advantageous to create testbed using this framework, but at the cost of ease of installation. With comparison to other cloud orchestrator tools, OpenStack attracts the largest number of developers and contributors.

2. *OpenNebula*
   OpenNebula is a cloud computing platform for creating and managing heterogeneous distributed cloud infrastructures. It is started as a research project and now it is backed by OpenNebula Systems. It is one of the earliest cloud frameworks for IaaS [9]. It has been written using multiple languages such as C, C++, Java, Ruby and shell scripting []. Basic components of an OpenNebula system are:
   a) `Front-end` that executes OpenNebula Sevices
   b) `Hosts` provides resources needed by VMs

Framework	Language	Architecture	Installation	Administration	Security
OpenStack	Python,Shell Scripting	Fragments	Difficult	Multiple CLI, UI	Keystone
CloudStack	Java	Monolithic	Medium	UI, CLI, EC2	Baseline
OpenNebula	C,C++,Java,Ruby,Shell	Fragments	Medium	UI, CLI, EC2	Auth
Eucalyptus	Java, C	5 Parts, AWS	Medium	EC2, CLI	Registered

Figure 3: Comparison of different Frameworks

c) `Datestore` that hold the base images of the VMs

d) `Networks` used to support services such as interconnection, VLANs for VMs.

3. *Apache CloudStack*
   Apache CloudStack is an open-source cloud framework to build and manage networks of virtual machines. It is managed by widely known Apache Software Foundation. CloudStack cloud has a hierarchical structure which enables it to scale to manage tens of thousands of physical servers, all from a single management interface. Organizational units of CloudStack deployment consists of: `Hosts`, `Zones`, `Pods`, `Clusters`, `Secondary Storage`, `Primary Storage`. CloudStack is written completely in Java and provides command line and GUI based interfaces for monitoring and controlling hosts.

4. *Eucalyptus*
   Eucalyptus, is an open source cloud framework which provides an EC2-compatible cloud computing platform and S3-compatible cloud storage platform. Eucalyptus is written using Java and C , and its been managed by Eucalyptus Foundation [8]. Eucalyptus is the acronym for Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems. Eucalyptus contains five parts: `Cluster Controller`, `Cloud Controller`, `Node Controller`, `Walrus Storage Controller`, `Storage Controller`.

Figure 3 shows the features of different cloud orchestration frameworks [32] [51]. Paper [16] defines various criteria to compare the cloud frameworks to not just with each other but with respect to bare-metal also. As our project is focused on development of orchestrator for VNFs, we may need these frameworks to construct and manage cloud infrastructure of the testbed. As any one framework is not the best solution in every situations and on every parameters, the table below is an effort to compare these frameworks with respect to desired and applicable criteria that is aligned with our project [45] [50] [54].

Framework	OpenStack	OpenNebula	CloudStack	Eucalyptus
Scalability	High	High	Medium	Medium
Flexibility	High	High	Medium	Medium
Maturity	High	High	Medium	Medium
Network Management	High	High	Medium	Medium
Usability	Medium	Medium	Medium	Low

Table 2: Framework Comparison based on desired criterias

4.7 MININET

Mininet is a network emulator software that allows us to create virtual networks with hosts, switches, controllers, and links [38]. Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking.

- It provides the ability to quickly create a network testbed for developing OpenFlow applications.
- It helps us to try out different protocols and complex topologies, and to do experiments without real network equipment.
- System-level regression tests, which are repeatable and easily packaged, are supported.
- For debugging or running network-wide tests, a Command Line Interface (CLI) that is topology- and OpenFlow-aware, is provided.
- It also provides a straightforward and extensible Python API for network creation and experimentation.

The main objective of our project group is to develop a network orchestrator, which is supposed to be able to support different kinds of controllers, switches, and different underlying network topologies. With mininet, we can simulate any kind of network infrastructure and test our orchestrator in different environments. Also, during the development phase, mininet can be a great tool for our project group to experiment with SDN controllers and applications.

5 RELATED WORK

In this chapter we will discuss some related work which can be used to accomplish the Use Cases (cf. 2.2). Since our desired framework has variety of responsibilities several research topics may be addressed: Firstly, we look at scientific efforts in examining a proper specification for chains of VNFs as described in Use Case (cf. Section 5.1). Secondly, we take a look at techniques of how to maintain the deployed VNFs in the network in section 5.2. This includes methods to efficiently route packages along the chains, techniques to handle multiple users in one infrastructure and general techniques to manage VNFs. All these activities are part of Use Case 1. Lastly there are several existing orchestration frameworks for NFV. These Frameworks use, combine or extend the aforementioned techniques. Section 5.3 will give a brief summary of their main features and ideas, thus providing us with additional reference points for our orchestrator. We want to point out that this chapter only provides starting points for our final architecture. The suggested solution are neither final nor complete. If we find more suitable solutions, we will explain them in future documents.

5.1 SPECIFICATION OF VNFS

Proper Specification of VNF chains as defined in Use Case 5 is a fundamental task, as the specification provides reference points for all decisions of orchestrator and the plugins. Based upon the provided information VNF are placed, scaled and traffic is routed through them. Hence, the other Use Cases are also affected by this.
The problem of service chaining has been subjects to several drafts of the standardization agencies ETSI and IETF. The general consensus is, that unaltered Forwarding Graph are not feasible and they need to be augmented with additional information to enable efficient placement and scaling. Among their contributions are a white paper [22], use-cases [52], an architectural framework [23]. However, they do not do give a dedicated specification model, but instead list assumptions such a model must fulfill in order to be compatible with their work. A very forwarding-oriented specification is made in [2] by Blendin et al. The model focuses on Gi Lan. They classify middleboxes based upon two aspects: On which layer in the ISO/OSI-Model they apply changes to the traffic, and whether they can be shared between multiple users in the network. This model does not consider placing and scaling at all, but seems to be helpful when forwarding traffic

through the VNFs in a single data center. We will address this further in section 5.2 when we discuss related work for use case 1. In [26] by Joseph and Stoica, some selected middleboxes are modeled. Though their functions are detailed, they do not capture hardware requirements like computational resources or side effects like the additional traffic that is put on the network. Therefore this solution does not seem to be a good candidate.
Mehraghdam et al. [37] deliver the most rigorous attempt to model service chains in a network with multiple data centers. In their model, a VNF can either be deployed to a switch or a data center. The properties for a chain are described using a context-free grammar. The grammar is designed to leave several degrees of freedom when deploying the chain. These can be exploited for the optimal placement. The internal topology of the data centers is not a part of the model. This makes it a good choice for use case 2, as the infrastructure is also obscured from the user here. Furthermore, there exits an implementation of the model using YANG [36], which we could use or extend for our purposes.

5.2 MAINTENANCE OF VNFS

Forwarding along Service Chains

The orchestrator must make sure that packets are always correctly forwarded along the chain. This is complicated by several factors. Firstly, there is *packet mangling*, i.e. the fact that certain VNFs, e.g. Load Balancers, change the destination or other fields of the packet header. Another problem is caused by the fact that stateful middleboxes may get cloned or change their location. In those cases, we need to ensure *flow affinity*, i.e. that a flow is routed through the right instance of the middlebox. This must not be confused with *flexibility* and *scalability*, which state that more chains and services are added dynamically during runtime. Lastly, the number of rules on a switch should be relatively small.
In general there are two approaches to handle routing, Flow-based Routing and Tag-based Routing. The first one has the obvious disadvantage that a packet of each flow has to traverse the controller. Unmodified flow rules do not handle packet mangling and older approaches like the PLayer [27] only consider static setups and are not flexible or scalable. A very recent approach to counter this is *StEERING* [56], which greatly benefits from *OpenFlow*s pipelining mechanism (cf. section 4.1) as they encode the chains in the metadata.
Another problem of most flow-based solutions is, that the middleboxes which mangle traffic need to be specified by the user. We already mentioned an example for this in the previous section. However, this could be countered with the *SiMPLE* Framework [41]. It collects several packets of each flow before and after they leave a

middlebox. This way it can determine where packets are mangled if its not in the specification. The downside is increased network load and high memory usage. Since our use cases either require user specification (cf. use case 1) or the platform itself provides the specification(cf. use case 2), this solution loses even more appeal. On the other hand, *Network Service Headers* [42] and *FlowTags* [15] are examples of tag-based routing. This decreases the number of rules at the switches and the communication with the controller, because the tags can contain much more information on where to route next than normal packet headers. On the other hand, bigger packets increase the network load and more importantly the middleboxes have to be adapted to compute and add the tags to the packets. In existing tools like *Bro*[1] or *Squid*[2] this is an non-trivial task.

Configuration and Management of VNFs

As stated in sections 4.1 and 4.3, OpenFlow and NETCONF have been widely accepted for the configuration of switches. Such a uniform and widely accepted interface is missing for middleboxes. This has several reasons: Most importantly there is the diversity of functions of VNFs, which leads to more complex internal states. These are handled differently by each vendor to archive different optimizations. Despite these drawbacks there are proposals to this problem, which may be interesting for our orchestrator: Gember et al. propose the concept of software-defined middlebox networking [17]. The state of a VNF is represented by simple key-value pairs which can be altered by the controller. This trades off vendor optimization with flexibility for control application. Also, this approach requires to make changes to the implementation. The work of Wang and Odini has a different approach [3]. They propose a generic framework to manage arbitrary VNFs without changing their source code. Instead, for each middlebox a (possibly vendor-specific) plugin has to be written. This could be just as much work as changing the implementation. Thus, at this point we cannot say which of these approaches is more feasible for our orchestrator.

Handling Multiple Users

Since our orchestrator will provide support for multiple users, each of them must be provided with an interface to configure the infrastructure on which their VNFs run. In general, this can be archived through virtual overlay networks or through slicing. Slicing gives a user control only over a fraction of the flow table. This way the users are given more control over the actual network but in return they have to handle changes in the infrastructure, e.g. migration of a

[1] bro.org
[2] squid-cache.org

Name	Feature	Finished	Source
CoMb	Consolidation	✓	?
Stratos	Routing Approach	✓	✗
Cloud4NFV	Data Modelling	✓	?
T-NOVA	Multi-User Support/NFaaS	✗	(✓)
OpenMANO	Custom VIM API	✗	✓

Table 3: Overview over existing orchestration frameworks and their main focuses

VM, themselves. As defined in use case 1 the our platform should release the user of this and therefore we use Virtualization. This approach builds up on top of slicing and gives a user the illusion to be the only administrator in the network. Moreover users are not presented the actual network topology but an abstraction that only consists of their VNFs. A single link in the virtual instance may consists of several hops in the real network. This has the benefit that all conflicts with other tenants, machine migrations etc. are hidden from the user and handled by the platform.
There are several scientific works which focus on virtual SDNs: The oldest are Onix [31], a proof-of-concept for an distributed SDN controller with an integrated virtualization module and ADVisor [43], which lacks support for topologies, where multiple virtual nodes may need to be deployed on top of a single switch. More recent and mature solutions are vSDN [4] and OpenVirtex[3]. vSDN is part of the T-NOVA project, which we will introduce in the next section. OpenVirtex is open sourche has a running beta version which even supports OpenStack. Both these solutions require changes to the hypervisor and therefore affect interface 6 in the reference architecture (cf. figure 1).

5.3 FRAMEWORKS AND IMPLEMENTATIONS

There are several approaches for NFV Frameworks in existence or in development. In the following we will present all frameworks we found and explain how they handle some of the challenges proposed in the previous sections. An additional overview is given in table 3. *CoMb* [44] is the oldest orchestration framework we found. It was released before the ETSI Reference Architecture but is surprisingly similar. The outstanding feature of CoMb is the concept of the *CoMb Box*: Based on ClickOS, it runs on every server in a data center and acts as both a hypervisor and a switch. Furthermore it tries to maximize consolidation by reusing certain software elements. If for

[3] ovx.onlab.us/

example two middleboxes need information from the application layer of the incoming traffic, a single software element provides it to both. Their evaluations show that this methods saves 26% - 89% of computational resources. The concept of the *CoMb Box* is interesting and could be possibly be rebuilt with modern technologies like Open vSwitch.

Stratos [18] is an NFV Solution developed and maintained by the University of Wisconsin. It is designed to manage the deployment of VNFs in a single data center. Most notably is the way Stratos handles the forwarding as it combines Flow- and Tag-based approaches. Therefore each service chain is divided into logical subchains. Each time a package header is altered by a VNF, a new subchain begins. When the first packet of a flow arrives at a subchain, the SDN Contoller computes a tag for the flow which is written into the packet header. Subsequent routing along the subchain is then determined by the tag. A downside of this approach is that the logical chaining sometimes requieres redundant clones of virtual middlebox. The design of *Stratos* differs in two ways from our planned framework: Firstly, it only models single data centers instead of carrier grade networks with several cloud sites. Secondly it lacks the ability to add custom plugins. A positive Aspect however is the combination of different routing techniques.

T-NOVA [53] is a collaborative research project funded by the European Union. Its main goal is to provide a marketplace for Network Functions as a Service (NFaaS). This is similar to our use case 2: Customers are offered a virtual network in which they can place their desired network functions. The VNFs themselves can be chosen from a catalogue similar to an "App-Store". Interesting for our project could also be that *T-NOVA*-Orchestration takes the presence of several data centers and several tenants into account. That means the placement of virtual network is in DCs close to the customers traffic and the performance of a VNF is not affected by the VNFs of other user deployed inside the same DC. The proposed mechanisms to bill customers are beyond the scope of this project. Despite the project is open source, no code has been published so far, possibly because it not finished yet.

Cloud4NFV [47] is a concept by *Portugal Telecom*[4] and the University of Averio. Just as *T-NOVA* it aims to provide Wheras the former is in rather early state of development and only envisions taking hardware and location into account in the placement decision, Cloud4NFV already proposes a data model for a deployed VNF. The model includes Compute and Link Flavors as well as information about the network's topology. This modeling of the infrastructure could be useful for our project. The various models are stored in MySQL and

[4] telecom.pt

CouchDB Databases. Furthermore the OCCL protocol is used for communication between different cloud sites.
OpenMANO is the most recent of the frameworks. In contrast to the other frameworks it wasn't developed in collaboration with a university. Instead it is an internal project by spanish Telecommunication Company (telco) *Telefonica*[5]. Like *Cloud4NFV* its architecture remains very close to the ETSI Reference. Also they build their own API for the VIM to better support the specific needs of VNFs. What really separates this project from the others is, that we have access to source code. That means, we can actually perform tests to evaluate the design decisions and have a working basis to try other approaches. Hence this project will be of special significance for our rapid prototyping phase.

[5] tid.es

6 TIMEPLAN

6.1 OVERVIEW

In this chapter we present a timeplan, that lists the upcoming tasks and schedules time durations and deadlines. The timeline is supposed to only give a rough idea over the tasks and their times, the actual plan might be adapted and defined further during the project. For now, we divide the project into seven main tasks depicted in table 4.

#	Task	Start	End
1	Project organisation	14.04.15	31.05.15
2	Preparation & Presentation of mini seminar	14.04.15	07.05.15
3	Developing project plan	07.05.15	09.06.15
4	Reviewing technologies	05.06.15	31.07.15
5	Designing architecture	15.06.15	14.09.15
6	Implementation & Deployment	24.08.15	08.01.16
7	Presentation	11.12.15	31.03.16

Table 4: List of all tasks in the timeplan

The following list further defines the tasks:

1. Project organisation. The main objective of this task is to agree on project management means, including development strategies and management and communication tools. We also set up our working environment, introduce each other to technologies like versionising or typesetting tools, and review our skills and expertises. One goal is to divide our group into subgroups, working on different tasks.

2. Preparation & Presentation of mini seminar. Goal of this already completed task was to get familiar with some tools and standards, and discuss their relevance for our project. Also, it was a starting point to get to know each other and to get an impression over what the project is going to be. It ended with the presentation of everybody's topic and a group discussion. The work can be discussed and referenced later in our project.

3. Developing project plan. This task is about defining what we are going to do in the next months. Since the outcome of this

task should be the project plan document, reading this assures oneself the task is successfully finished. For our project group, the document is a valuable mean to get an overview over the overall problem, related technologies and required subtasks.

4. Reviewing technologies. Based on the technologies we discussed in the mini seminars, we will once again recap some of them and briefly discuss their relevance for our project.

5. Designing architecture. This will be one of the main tasks, since this is the step where we will talk about the actual architecture of the system. We will decide on which existing technologies and design possibilities to use. Accompanying this process, we will also document our decisions in an architecture document. This document will be the basis for the later implementation.

6. Implementation & Deployment. Implementing the system is expected to require the most time and effort. Outcome should be the running software on a testbed, in a setting feasible to test and present exemplary scenarios. Implementing, deploying, testing and documenting will be a continuous process during the task. Once again we will divide our group into subgroups working on different assignments in order to achieve the overall goal.

7. Presentation. After implementing the system, we will present it to our supervisors and professor. For that, we are going to simulate some use cases and exemplary traffic on a running instance of our software.

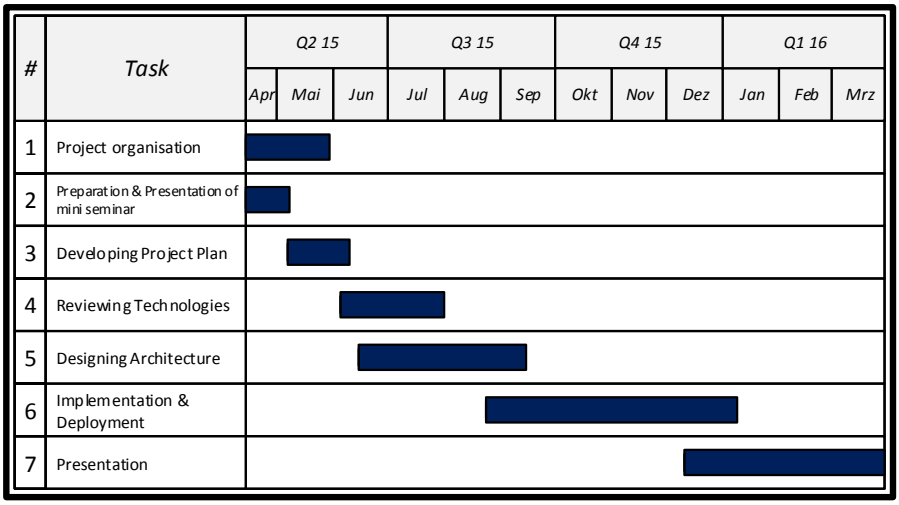

Figure 4: A gantt diagram visualising the timeplan.

The sequence of this tasks is visualized in figure 4. As it is apparent in the figure, the tasks are not enclosed in theirselves, but crossing

over into each other. The deadlines serve to assist ourselves to stay in the schedule. Actual milestones and deadlines for the task are to be defined specifically later in the project.
This particularly holds for the tasks 5 and 6. The process of designing the architecture and implementing and deploying the software will be performed in a rapid-prototyping manner. Therefore, the actual task of implementing will most likely start even earlier, concerning related tasks such as trying out different tools and existing software. But since this process can also be seen as part of an architectural discussion, we decided to place a larger timespan for task 5, and regard task 6 more as the challenge of adapting and implementing on top of what we build and deployed in the architecture phase.

6.2 NEXT STEPS

Starting immediately after the presentation of this document, our next task is to develop an architecture for our project. For this, we already decided on an approach of deliberating upon different possibilities. In the first phase, starting Tuesday, 9 June, we divide our group in two teams in the following way:

1. *Team OpenMANO*. This groups task is to deploy and test OpenMANO (cf. section 5.3), and particularly work out which of our requirements are met, and which components are yet missing. They will have to estimate the effort needed to adapt OpenMANO to our needs, especially regarding the plugin-functionality our software will have to provide. This tasks focus lies on deploying OpenMANO and reverse-engineering its code. The team consists of five members, namely Mihir, Pranuthi, Tarun, Andreas and Tasneem.

2. *Team Self-Made*. The task of this group is a bit broader, since the idea here is not to use an existing NFV-solution such as OpenMANO, but build our own system on top of existing tools. They will have to evaluate different technologies such as OpenStack, OpenDaylight or ClickOS, and figure out how and at which places our project can benefit from them. In the end, they will come up with a proposal of an architecture apart from OpenMANO.
   Seven members will participate in this team, namely Kristian, Guruprasad, Sergio, Thorsten, Ashish, Sven and Arjun.

This first phase is estimated to take about 3-4 weeks. After this time, the two groups get back together, present their results, and decide on one of the two solutions together. The whole group will then continue to work on the solution they agreed upon.
The practice of having stand-up meetings, that we have already established in the last weeks, will serve our exchange of progress and

regular presentation of first results. If we find problems regarding the subdivision of the group or the specific tasks we work on, we can then rearrange directly.

7 CONCLUSION

As the presented document made apparent, the project will be a big challenge for us. There are yet many taks to solve and many decicions to take. The requirements and possibilities we stated in this project plan will be our basis for this process.
The next big step for us, is to take the results of this document and transfer them to an architectural discussion. Based on the possibilities and advantages of the technologies we presented in this document, we will have to discuss and agree on a composition ot them. Also, we have to match these tools to the conceptual architecture we will have to develop. For this, we already introduced some related work and talked about the ETSI framework, which we will certainly have to look much deeper into during the architectural discussion.

BIBLIOGRAPHY

[1] .

[2] J. Blendin et al. "Position Paper: Software-Defined Network Service Chaining." In: *Software Defined Networks (EWSDN), 2014 Third European Workshop on*. 2014, pp. 109–114. DOI: 10.1109/EWSDN.2014.14.

[3] Wang Bo and M.-P. Odini. "Short Paper: Lightweight VNF manager solution for virtual functions." In: *Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on*. 2015, pp. 148–150. DOI: 10.1109/ICIN.2015.7073823.

[4] Z. Bozakov and P. Papadimitriou. "Towards a scalable software-defined network virtualization platform." In: *Network Operations and Management Symposium (NOMS), 2014 IEEE*. 2014, pp. 1–8. DOI: 10.1109/NOMS.2014.6838411.

[5] Zheng Cai. "Maestro: Achieving Scalability and Coordination in Centralizaed Network Control Plane." AAI3521292. PhD thesis. Houston, TX, USA, 2012. ISBN: 978-1-267-53931-1.

[6] Chunfeng Cui et al. "Network Functions Virtualisation." In: ().

[7] Chunfeng Cui et al. *NFV White paper 2*. Version 2. https://portal.etsi.org/nfv/nfv_white_paper2.pdf. 2013.

[8] Eucalyptus Documentation. https://www.eucalyptus.com/docs/eucalyptus/4.1.1/index.html/#install-guide/euca_components.html.

[9] OpenNebula Documentation. http://docs.opennebula.org/.

[10] OpenStack Documentation. http://docs.openstack.org/.

[11] David Erickson. "The Beacon Openflow Controller." In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. HotSDN '13. Hong Kong, China: ACM, 2013, pp. 13–18. ISBN: 978-1-4503-2178-5. DOI: 10.1145/2491185.2491189. URL: http://doi.acm.org/10.1145/2491185.2491189.

[12] ETSI. *Network Functions Virtualisation: Architectural Framework, Version: 001 V1.1.1*. http://docbox.etsi.org/ISG/NFV/Open/Published/gs_NFV002v010201p-ArchitecturalFramework.pdf. 2014.

[13] ETSI. *Network Functions Virtualisation: Infrastructure Overview, Version: 001 V1.1.1*. http://docbox.etsi.org/ISG/NFV/Open/Published/gs_NFV-INF001v010101p-InfrastructureOverview.pdf. 2015.

[14] ETSI. *Network Functions Virtualisation (NFV): Management and Orchestration)*. Version 1.1.1. http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf. 2014.

[15] Seyed Kaveh Fayazbakhsh et al. "Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions using FlowTags." In: *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, 2014, pp. 543–546. ISBN: 978-1-931971-09-6. URL: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/fayazbakhsh.

[16] Saurabh Kumar Garg, Steve Versteeg, and Rajkumar Buyya. "A framework for ranking of cloud computing services." In: *Future Generation Computer Systems* 29.4 (2013), pp. 1012–1023.

[17] Aaron Gember et al. "Design and Implementation of a Framework for Software-defined Middlebox Networking." In: *SIGCOMM Comput. Commun. Rev.* 43.4 (Aug. 2013), pp. 467–468. ISSN: 0146-4833. DOI: 10.1145/2534169.2491686. URL: http://doi.acm.org/10.1145/2534169.2491686.

[18] Aaron Gember et al. "Stratos: A Network-Aware Orchestration Layer for Virtual Middleboxes in Clouds." In: *arXiv preprint arXiv:1305.0209* (2013).

[19] Aaron Gember et al. "Toward software-defined middlebox networking." In: *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. ACM. 2012, pp. 7–12.

[20] The Open vSwitch Group. *Overview of functionality and components*. 2014. URL: https://github.com/openvswitch/ovs/blob/master/README.md (visited on 12/02/2014).

[21] Natasha Gude et al. "NOX: Towards an Operating System for Networks." In: *SIGCOMM Comput. Commun. Rev.* 38.3 (July 2008), pp. 105–110. ISSN: 0146-4833. DOI: 10.1145/1384609.1384625. URL: http://doi.acm.org/10.1145/1384609.1384625.

[22] R. Guerzon. "Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges and Call for Action, Introductory white paper." In: (2012).

[23] Joel Halpern and Carlos Pignataro. *Service Function Chaining (SFC) Architecture*. Internet-Draft draft-ietf-sfc-architecture-08. http://www.ietf.org/internet-drafts/draft-ietf-sfc-architecture-08.txt. IETF Secretariat, 2015. URL: http://www.ietf.org/internet-drafts/draft-ietf-sfc-architecture-08.txt.

[24] Bo Han et al. "Network function virtualization: Challenges and opportunities for innovations." In: *Communications Magazine, IEEE* 53.2 (2015), pp. 90–97. ISSN: 0163-6804. DOI: 10.1109/MCOM.2015.7045396.

[25] Ansible Inc. *How Ansible works.* http://www.ansible.com/how-ansible-works, Accessed: 2015-04-27. URL: http://www.ansible.com/how-ansible-works (visited on 04/27/2015).

[26] D. Joseph and I. Stoica. "Modeling middleboxes." In: *Network, IEEE* 22.5 (2008), pp. 20–25. ISSN: 0890-8044. DOI: 10.1109/MNET.2008.4626228.

[27] Dilip Antony Joseph et al. *A Policy-aware Switching Layer for Data Centers*. 2008.

[28] M. Keller et al. "A topology-aware adaptive deployment framework for elastic applications." In: *Intelligence in Next Generation Networks (ICIN), 2013 17th International Conference on*. 2013, pp. 61–69. DOI: 10.1109/ICIN.2013.6670895.

[29] Zuhran Khan Khattak, Muhammad Awais, and Adnan Iqbal. "Performance Evaluation of OpenDaylight SDN Controller." In: ().

[30] R. Khondoker et al. "Feature-based comparison and selection of Software Defined Networking (SDN) controllers." In: *Computer Applications and Information Systems (WCCAIS), 2014 World Congress on*. 2014, pp. 1–7. DOI: 10.1109/WCCAIS.2014.6916572.

[31] Teemu Koponen et al. "Onix: A Distributed Control Platform for Large-scale Production Networks." In: *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*. OSDI'10. Vancouver, BC, Canada: USENIX Association, 2010, pp. 1–6. URL: http://portal.acm.org/citation.cfm?id=1924968.

[32] Daniel Kranowski. *Iaas Private Cloud Brief Comparison*. 2012. URL: http://www.slideshare.net/bizalgo/cloudstack-vs-openstack-vs-eucalyptus-iaas-private-cloud-brief-comparison.

[33] Rakesh Kumar et al. "Open Source Solution for Cloud Computing Platform Using OpenStack." In: *International Journal of Computer Science and Mobile Computing* 3.5 (2014), pp. 89–98.

[34] Puppet Labs. *Overview of Puppet's Architecture*. http://docs.puppetlabs.com/puppet/3.8/reference/architecture.html, Accessed: 2015-04-24. URL: http://docs.puppetlabs.com/puppet/3.8/reference/architecture.html (visited on 04/24/2015).

[35] Nick McKeown et al. "OpenFlow: Enabling Innovation in Campus Networks." In: *SIGCOMM Comput. Commun. Rev.* 38.2 (Mar. 2008), pp. 69–74. ISSN: 0146-4833. DOI: 10.1145/1355734.1355746. URL: http://doi.acm.org/10.1145/1355734.1355746.

[36] Sevil Mehraghdam and Holger Karl. "Specification of Complex Structures in Distributed Service Function Chaining Using a YANG Data Model." In: *CoRR* abs/1503.02442 (2015). URL: http://arxiv.org/abs/1503.02442.

[37] Sevil Mehraghdam, Matthias Keller, and Holger Karl. "Specifying and Placing Chains of Virtual Network Functions." In: *CoRR* abs/1406.1058 (2014). URL: http://arxiv.org/abs/1406.1058.

[38] *Mininet Org*. URL: http://mininet.org/ (visited on 06/01/2015).

[39] White paper.

[40] The Linux Information Project. *Scalable Definition*. 2006. URL: http://www.linfo.org/scalable.html (visited on 05/21/2015).

[41] Zafar Ayyub Qazi et al. "SIMPLE-fying Middlebox Policy Enforcement Using SDN." In: *SIGCOMM Comput. Commun. Rev.* 43.4 (Aug. 2013), pp. 27–38. ISSN: 0146-4833. DOI: 10.1145/2534169.2486022. URL: http://doi.acm.org/10.1145/2534169.2486022.

[42] Paul Quinn et al. *Network Service Header*. Internet-Draft draft-quinn-sfc-nsh-07. http://www.ietf.org/internet-drafts/draft-quinn-sfc-nsh-07.txt. IETF Secretariat, 2015. URL: http://www.ietf.org/internet-drafts/draft-quinn-sfc-nsh-07.txt.

[43] E. Salvadori et al. "Generalizing Virtual Network Topologies in OpenFlow-Based Networks." In: *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*. 2011, pp. 1–6. DOI: 10.1109/GLOCOM.2011.6134525.

[44] Vyas Sekar et al. "Design and Implementation of a Consolidated Middlebox Architecture." In: *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. San Jose, CA: USENIX, 2012, pp. 323–336. ISBN: 978-931971-92-8. URL: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/sekar.

[45] Peter Sempolinski and Douglas Thain. "A comparison and critique of eucalyptus, opennebula and nimbus." In: *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. Ieee. 2010, pp. 417–426.

[46] Justine Sherry et al. "Making Middleboxes Someone else's Problem: Network Processing As a Cloud Service." In: *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM '12. Helsinki, Finland: ACM, 2012, pp. 13–24. ISBN: 978-1-4503-1419-0. DOI: 10.1145/2342356.2342359. URL: http://doi.acm.org/10.1145/2342356.2342359.

[47] J. Soares et al. "Cloud4NFV: A platform for Virtual Network Functions." In: *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*. 2014, pp. 288–293. DOI: 10.1109/CloudNet.2014.6969010.

[48] Ylva Torberntsson Kim;Rydin. *A Study of Configuration Management Systems.* http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-228139, Accessed: 2015-04-25. 2014. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-228139 (visited on 04/25/2015).

[49] Paul Venezia. *Review: Puppet vs. Chef vs. Ansible vs Salt.* http://www.infoworld.com/article/2609482/data-center/data-center-review-puppet-vs-chef-vs-ansible-vs-salt.html, Accessed: 2015-04-29. 2013. URL: http://www.infoworld.com/article/2609482/data-center/data-center-review-puppet-vs-chef-vs-ansible-vs-salt.htm (visited on 04/29/2015).

[50] Gregor Von Laszewski et al. "Comparison of multiple cloud frameworks." In: *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE. 2012, pp. 734–741.

[51] Xiaolong Wen et al. "Comparison of open-source cloud management platforms: OpenStack and OpenNebula." In: *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*. IEEE. 2012, pp. 2457–2461.

[52] Will et al. *Service Function Chaining (SFC) Use Cases*. Internet-Draft draft-liu-sfc-use-cases-05. http://www.ietf.org/internet-drafts/draft-liu-sfc-use-cases-05.txt. IETF Secretariat, 2014. URL: http://www.ietf.org/internet-drafts/draft-liu-sfc-use-cases-05.txt.

[53] G. Xilouris et al. "T-NOVA: A marketplace for virtualized network functions." In: *Networks and Communications (EuCNC), 2014 European Conference on*. 2014, pp. 1–5. DOI: 10.1109/EuCNC.2014.6882687.

[54] Sonali Yadav. "Comparative Study on Open Source Software for Cloud Computing Platform: Eucalyptus, Openstack and Opennebula." In: *International Journal Of Engineering And Science* 3.10 (2013), pp. 51–54.

[55] Frank Yue. *Network Functions Virtualization—Everything Old Is New Again*. 2013.

[56] Ying Zhang et al. "StEERING: A software-defined networking for inline service chaining." In: *Network Protocols (ICNP), 2013 21st IEEE International Conference on*. 2013, pp. 1–10. DOI: 10.1109/ICNP.2013.6733615.