DEGREE PROJECT IN COMMUNICATION SYSTEMS, SECOND LEVEL
STOCKHOLM, SWEDEN 2015

# A Data Model Driven Approach to Managing Network Functions Virtualization

## *Aiding Network Operators in Provisioning and Configuring Network Functions*

KRISTIAN SÄLLBERG.

KTH ROYAL INSTITUTE OF TECHNOLOGY
*INFORMATION AND COMMUNICATION TECHNOLOGY*

# A Data Model Driven Approach to Managing Network Functions Virtualization

Aiding Network Operators in Provisioning and Configuring Network Functions

Kristian Sällberg

Master of Science Thesis

Communication Systems
School of Information and Communication Technology
KTH Royal Institute of Technology
Stockholm, Sweden

July 24, 2015

Examiner: Professor Gerald Q. Maguire Jr.

# Abstract

This master's thesis explains why certain network services are difficult to provision and configure using IT automation and cloud orchestration software. An improvement is proposed and motivated. This proposed improvement enables network operators to define a set of data models describing how to provision and interconnect a set of Virtual Network Functions (VNFs) (and possibly existing physical network functions) to form networks. Moreover, the proposed solution enables network operators to change the configuration at runtime. The work can be seen as a step towards self managing and auto scaling networks.

The proposed approach is compared to a well known cloud management system (OpenStack) in order to evaluate if the proposed approach decreases the amount of time needed for network operators to design network topologies and services containing VNFs. Data is collected through observations of network operators, interviews, and experiments.

Analysis of this data shows that the proposed approach can decrease the amount of time required for network operators to design network topologies and services. This applies if the network operators are already acquainted with the data modeling language YANG. The amount of time required to provision VNFs so that they respond to connections can also be decreased using the proposed approach. The proposed approach does not offer as much functionality as OpenStack, as it is limited to VNF scenarios.

**Keywords:** NFV, SDN, VNF, Virtualization.

# Sammanfattning

Denna masteruppsats förklarar varför vissa nätverkstjänster är svåra att skapa och konfigurera med IT-automationsverktyg och mjukvara för molnorkestrering. En förbättring föreslås och motiveras. Den föreslagna förbättringen tillåter nätverksoperatörer att definiera en mängd datamodeller, för att beskriva hur Virtuella Nätverksfunktioner (VNF:er) skall instantieras och kopplas ihop till nätverkstjänster. Dessutom tillåter lösningen nätverksoperatörer att ändra konfiguration under tiden nätverken hanterar trafik. Arbetet kan ses som ett steg mot självhanterande och automatiskt skalande nätverk.

Den föreslagna lösningen jämförs med ett välkänt molnorkestreringsverktyg (OpenStack) för att utvärdera om den föreslagna lösningen sänker mängden tid som nätverksoperatörer behöver för att designa nätverkstopologier och tjänster som innehåller VNF:er. Data samlas in genom observationer av nätverksoperatörer, intervjuer, och experiment.

Analys av datan visar att den föreslagna lösningen kan minska tiden som behövs för att designa nätverkstopologier och tjänster. Fallen där detta är applicerbart, är när VNF:er närvarar i nätverk. Dessa är enklare att skapa, konfigurera, och ändra under tiden de exekverar, med den föreslagna metoden. Detta kräver också att nätverksoperatören är bekant med datamodelleringsspråket YANG. Tiden det tar att provisionera VNF:er, tills dess att de svarar till anslutningar, kan sänkas med hjälp av den föreslagna metoden. Den förslagna metoden erbjuder väsentligt begränsad funktionalitet jämfört med OpenStack, den fokuserar på att hantera VNF:er.

**Keywords:** NFV, SDN, VNF, Virtualization.

# Acknowledgements

I have enjoyed the help of my academic adviser and examiner, Professor Gerald Q. "Chip" Maguire Jr., who has provided invaluable feedback, constructive criticism, and a never ending flow of ideas on how to improve this master thesis project.

I want to thank my industrial supervisors Claes Wikström and Johan Bevemyr for suggesting this master thesis topic to me, helping and encouraging me throughout this master's thesis project, and providing many ideas and interesting discussions. Stefan Wallin has continuously provided helpful feedback during the process of writing this master's thesis. Tomas Mellgren helped me with all practical problems I encountered and always encouraged me to keep going.

I want to thank all of the employees of Cisco, Inc., formerly Tail-f, where I worked during the period of this master's thesis. They allowed me take part in the inspiring, innovative, and encouraging environment that they share. A special thank you to the Cisco, Inc. employees who volunteered their time and expertise as participants in the evaluation phase of this master's thesis project. Finally, I wish to thank my beloved family for their everlasting support and encouragement.

# Contents

# List of Figures

# List of Tables

# List of Listings

# Acronyms & Abbreviations

This section defines abbreviations used in this master thesis.

| | |
|---|---|
| **API** | Application Programming Interface |
| **ASA** | Adaptive Security Appliance |
| **ASIC** | Application-Specific Integrated Circuit |
| **BGP** | Border Gateway Protocol |
| **CDB** | ConfDB or Configuration Database, part of ConfD |
| **CIDR** | Classless Inter-Domain Routing |
| **CLI** | Command Line Interface |
| **ConfD** | Configuration Daemon,<br>network configuration software |
| **COTS** | Commercial Off-the-shelf Hardware |
| **CPE** | Customer Premises Equipment |
| **CSR1000v** | Cisco Cloud Series Router |
| **Day0** | Initial configuration of a managed network function |
| **ETSI** | European Telecommunications Standards Institute |
| **FBOSS** | Facebook Open Source Switch |
| **FWaaS** | Firewall as a Service |
| **GD** | Gabbleduck (Proposed approach of managing NFV) |
| **HOT** | Heat Orchestration Template |
| **IAB** | Internet Architecture Board |
| **IEEE** | Institute of Electrical and Electronics Engineers |
| **IETF** | Internet Engineering Task Force |

| | |
|---|---|
| **IOS** | Internetwork Operating System |
| **IP** | Internet Protocol |
| **JSON** | JavaScript Object Notation |
| **KVM** | Kernel-based Virtual Machine |
| **NAT** | Network Address Translation |
| **NCS** | Network Control System |
| **NIC** | Network Interface Controller |
| **NSD** | Network Service Descriptor, concept from ETSI NFV working group |
| **OASIS** | Organization for the Advancement of Structured Information Standards |
| **OS** | Operating System |
| **PNF** | Physical Network Function |
| **QEMU** | Quick Emulator (Hypervisor) |
| **RFC** | Request for Comments. A document published by the IETF. Often a protocol specification based upon an interoperating set of implementations. |
| **RPC** | Remote Procedure Call |
| **SDN** | Software-defined Networking |
| **SNMP** | Simple Network Management Protocol |
| **SSH** | Secure Shell |
| **TOSCA** | Topology and Orchestration Specification for Cloud Applications |
| **VLAN** | Virtual Local Area Network |
| **VM** | Virtual Machine |
| **VNF** | Virtual Network Function, concept from ETSI NFV working group |

| | |
|---|---|
| **VNFD** | Virtual Network Function Descriptor, concept from ETSI NFV working group |
| **VPN** | Virtual Private Network |
| **XML** | Extensible Markup Language |
| **YAML** | YAML Ain't Markup Language |

## Chapter 1:

# Introduction

This chapter describes the problems that led to this master's thesis project. It provides an introduction to this thesis and serves as a preview of the technical and theoretical background that will be given in Chapter 2.

## 1.1 Background

Network management has traditionally been a labour intensive, and expensive activity. When building networks to handle higher amounts of traffic, new network appliances needed to be purchased, installed, and configured by network operators.

During the last decade, Software Defined Networking (SDN) [1] emerged and simplified network configuration and automation. SDN allows network operators to configure (and reconfigure) network devices programmatically. In parallel, cloud computing developed to help Internet businesses create more efficient and scalable architectures.

More recently, Network Functions Virtualization (NFV) [2], was designed and proposed by the European Telecommunications Standards Institute (ETSI) [1]. The concept of NFV makes it possible to fully or partially virtualize network functions on top of commercial of-the-shelf (COTS) hardware. VNF, as mentioned in the abstract, is a part of the broader concept of NFV. NFV is explained in greater detail in Section 2.1.

Cloud computing makes it possible to provision Virtual Machines (VMs) [3] using software. It also provides elastic scaling to match peak loads, high availability, and fault tolerance [4]. Cloud Computing and SDN are merging and synergy effects appear in this overlap [1]. In addition, NFV can be combined with cloud computing and SDN to create even greater value [2].

This intersection is illustrated in Figure 1.1 — highlighting the focus of this thesis project at the intersection of these three areas.

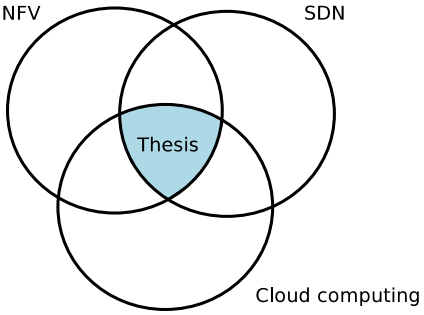


Figure 1.1: Thesis areas and focus.

## 1.2 Problem

When transitioning from the traditional network management approach to the NFV approach, two major problems must be overcome:

- Instead of buying and installing new network appliances, network operators need to be able to specify topologies, then have these topology specifications automatically realized by a system.
- The global state of an NFV network service has to be stored. This state includes all network appliances that are part of a service, and all configurations of these network appliances. This state will affect how new network appliances should be configured when they are added to the set of available services.

Figure 1.2 illustrates these two major problems. In this scenario, two new network appliances (Box1 and Box2) are added between two existing networks. Box1 and Box2 need to be (1) provisioned, this is, VMs are created and booted, and (2) configured to set up network interfaces, routing tables, etc. Then, the existing networks will need to be re-configured to be connected to Box1 and Box2. Box1 and Box2 could be instances of any proprietary or open source virtual or physical network appliance.

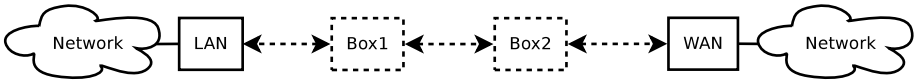


Figure 1.2: Provisioning and defining networks between network appliances.

Currently, there is a gap between cloud computing software and SDN tools [5]. Existing cloud management systems limit network operators in terms of network configuration capabilities, as these systems were not designed to connect to network appliances and to configure them. SDN tools, on the other hand, often specialize in finding existing network appliances and configure them. Their main purpose is not to provision VMs.

Cloud management systems were originally designed to provision and orchestrate enterprise web applications [5]. Components of enterprise applications, implemented as VMs, are usually only configured once.

The problem of provisioning and elastically scaling web services is well understood and has been solved by several projects, such as OpenStack (described in Section 2.3.2), Eucalyptus [6], and Google Ganeti [7]. This topic is examined and explained further in the master's thesis by Md.Iqbal Hossain and Md. Iqbal Hossain [8]. A primary focus of cloud management systems is to clone a data center from one location to another. Therefore, runtime configuration is generally not the main focus.

The Organization for the Advancement of Structured Information Standards (OASIS) defined a Topology and Orchestration Specification for Cloud Applications (TOSCA). TOSCA, as described in Section 2.3.4, provides a modeling framework for expressing applications and relationships between these applications. TOSCA focuses on the migration between different data centers, rather than on provisioning and configuring network appliances [9].

Managing Virtual Network Functions (VNFs) is a distinct problem than the typical cloud computing problem, and managing VNFs has a different set of requirements. These requirements are described in Section 2.1.1.

The gap between the tools described in this section and the practical requirements for deploying VNFs, has an important implication for network operators who design network services that include VNFs. This implication is that it is currently time consuming and difficult for network operators to design, deploy, and manage VNFs.

## 1.3 Purpose and Goals

This master's thesis project proposes an approach to address the problem of the gap between cloud management systems and SDN. This approach is referred to as *Gabbleduck* throughout the rest of this report. The purpose of this master's thesis is to evaluate whether the proposed approach can enable network operators to design NFV network services within a relatively low amount of time.

The goal is to provide network operators with technology which they will be able to use to quickly and productively design network services. From a wider perspective, the purpose of this master's thesis project is to evaluate whether a data model driven approach to NFV can serve end-users of many different kinds of applications and whether these fundamental infrastructure services can be improved. If they can be improved, then all applications building upon the resulting networks can benefit from faster development cycles.

## 1.4 Delimitations

As of July 2015, there are many cloud management systems available; however, the evaluations in this master's thesis will only consider a single popular open source project, specifically *OpenStack*. This focus is due to the limited duration of this thesis project.

In the future, elastic scaling of VNFs and network services will be important to meet peaks in the demand for applications executing on top of network services. Unfortunately, elastic scaling is outside the scope of this master thesis, hence the author did not implement any support for elastic scaling.

Virtualization using containers (Linux Containers and Docker) is rapidly becoming a popular phenomenon in cloud computing. However, this approach is outside of the scope of this thesis, as this thesis project will only consider virtualization using VMs.

There is a relation between the complexity faced by network operators to introduce or change network services, and the cost of introducing these changes. This exact relationship between this complexity and cost are outside the scope of this thesis project.

## 1.5 Structure of the Thesis

Chapter 2 provides deeper theoretical and technical background for the thesis. Chapter 3 discusses scientific methods and how to measure and validate Gabbleduck. Chapter 4 presents Gabbleduck in greater detail. Chapter 5 presents and analyzes the results of the evaluations that have been conducted as part of this master's thesis project, and then discusses the reliability and validity of these results. Finally, Chapter 6 presents conclusions and suggests future work.

## Chapter 2:

# Background

This chapter provides some theoretical background for the reader and establishes a technical foundation for this thesis. Section 2.1 gives background information about NFV. Section 2.2 describes network configuration and how this concept is used in this thesis. Section 2.3 discusses related work. Section 2.4 summarizes the chapter.

## 2.1 NFV

NFV, as presented in this thesis, is defined by ETSI. NFV is a concept that proposes to complement, and in the long run, replace, physical network appliances with virtual network functions. For instance, instead of selling a packet router as a hardware unit with specialized circuit boards, NFV enables the software part of this packet router to be sold separately as a VM ready to be instantiated in a cloud environment.

This allows network equipment vendors to innovate faster, since software has shorter development cycles than hardware. Vendors can save money by reducing their manufacturing costs [2]. Expensive production of specialized networking hardware is no longer needed in the same extent as previously.

The left side of Figure 2.1 shows the classical approach to building networks, where physical network appliances from many different vendors are bought and connected to form networks. The right side shows a typical NFV architecture, in which virtual network appliances of different vendors execute together on the same type of underlying hardware (even though physical network appliances may still be used in an NFV context). COTS servers, storage, and switches are bought and deployed to form a cloud environment in which VNFs can execute.

The NFV approach allows higher level network services, such as VLAN and Virtual Private Network (VPN) as a Service, to be constructed using software, rather than the traditional approach of physically connecting and configuring special purpose network devices.

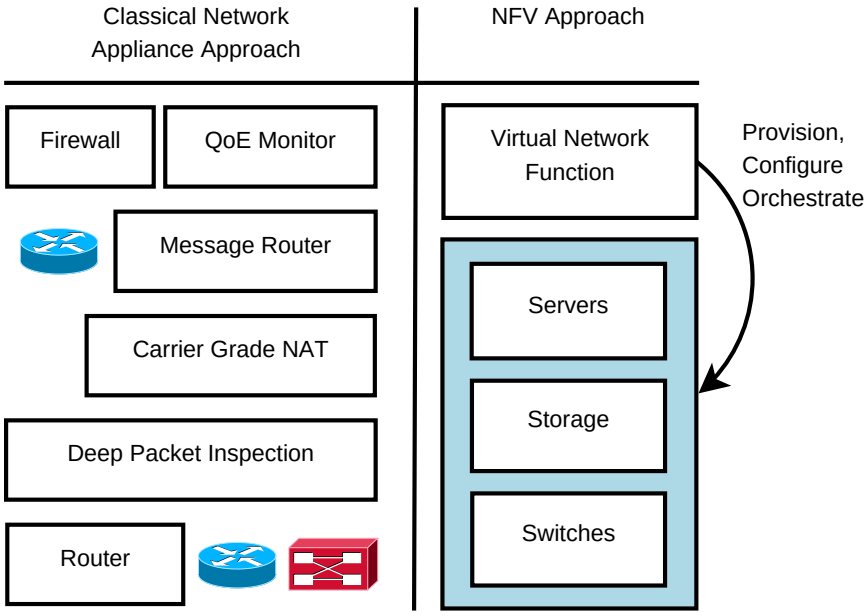


Figure 2.1: Comparison of classical network appliance approach and NFV. (Adapted from Figure 1 in [2].)

VNFs can be placed at and migrated to where they are needed most or where CPU resources are currently available [10]. This can save energy since elastic scaling and migration can be employed to use only as much computing power as needed. Section 2.3.5 describes another approach to networking equipment.

NFV defines Virtual Network Function Descriptors (VNFD), as templates describing how to deploy a single VNF. Network Service Descriptors (NSD) templates define a set of VNFDs. NSD templates are used to describe and design network services [11, 9]. NSD templates can include many VNFDs and rules for how these are interconnected.

### 2.1.1 NFV Requirements

Several requirements put on NFV by telecommunications operators, make NFV be more than just an instance of the traditional cloud computing concept or SDN [12]. Some of the relevant requirements, stated in ETSI's group specifications for NFV use cases [12], are:

- An NFV framework has to be able to manage networks consisting of both physical network functions and virtual network functions,
- An NFV framework should be able to migrate and provision VNFs across different environments,
- An NFV framework needs to be able to optimize placement of VNFs in order to reduce energy consumption and to distribute the workload of network services,
- An NFV framework needs to be able to provision and configure a VNF so that its full potential is reached in terms of performance,
- An NFV framework needs sufficient monitoring so that operational data can be fetched from VNFs under management,
- An NFV framework should be able to elastically scale VNFs so that service level agreements can be met, and,
- An NFV framework needs to provide the same level of service regardless of whether the network function is virtual or physical.

Some of these requirements are typically not met by existing cloud management systems, since they were designed to support a set of requirements [5] mostly concerned with provisioning web applications in the cloud.

## 2.2 Network Configuration

Traditional network configuration involves a network operator physically connecting to a network device, to use a command line interface (CLI) exposed by the network device's operating system (OS). Using this CLI, the network operator can change configurations supported by the network appliance, to realize changes in networks and network services.

In 2003, the Internet Engineering Task Force (IETF) published Request for Comments (RFC) 3535 [13]. It states that Simple Network Management Protocol (SNMP) [14] is used to monitor network devices, but that it is not used to configure them. Instead operators use CLIs to configure network devices. RFC 3535 proposes that a new standard is needed, one that can be general enough to configure devices regardless of the vendor-specific CLIs.

Following the requirements defined in RFC 3535, IETF designed and standardized the network configuration protocol NETCONF [15] and the associated data modeling language YANG [16]. Tail-f Systems (acquired by Cisco, Inc.), the company where this thesis project was carried out, developed tools around NETCONF and YANG to simplify the process of performing network configuration.

One of these tools is Configuration Daemon (ConfD) [17]. ConfD was designed to enable configuration of network devices using NETCONF and YANG. ConfD is placed inside of devices, where it enables these devices to be configured. Later on, another tool, Network Control System (NCS) [17], was designed and developed, to configure network devices over NETCONF, CLI, REST, and other protocols. This master's thesis project builds upon ConfD and NCS.

Each of these concepts will be described further. Section 2.2.2 explains the NETCONF protocol. Section 2.2.3 introduces the YANG data modeling language. Section 2.2.4 explains the uses of ConfD relevant to this thesis, and Section 2.2.5 does the same for NCS. Section 2.3.2 introduces the reader to relevant parts of OpenStack.

### 2.2.1 Cisco IOS

Cisco, Inc. produces a large range of network devices. Many of these devices make use of Internetwork Operating System (IOS) software [18], to enable network operators to configure and monitor devices. IOS also realizes the actual network functions, such as routing.

IOS provides a CLI that network operators can connect to over Secure Shell (SSH) or telnet. Listing 2.1 shows how the IOS CLI can be used to first enter the *privileged EXEC mode*, then to ask to see Border Gateway Protocol (BGP) status, and finally to ask the device to provide the names of its Network Interfaces (NICs). Privileged EXEC mode is marked by the character "#" after the device's name, when the mode is entered. BGP is not enabled on this particular network device, therefore it is not possible to get information about BGP.

```
Router1> enable
Password: *****
Router1# show bgp
% BGP not active
Router1# show html port all names
     this[0] = "GigabitEthernet1";
     this[1] = "GigabitEthernet2";
     this[2] = "GigabitEthernet3";
     this[3] = "LIIN0";
     this[4] = "LI-Null0";
Router1#
```

Listing 2.1: Cisco IOS CLI example.

### 2.2.2 NETCONF

NETCONF is a protocol designed specifically for network configuration [17, 15, 19]. NETCONF is implemented using Remote Procedure Calls (RPC) with the data encoded using the Extensible Markup Language (XML) [15]. NETCONF was designed from a set of requirements specified by protocol developers and network operators following an Internet Architecture Board (IAB) Network Management Workshop [13].

Some relevant and important requirements from this RFC, with respect to this thesis, are:

- Ease of use for network operators,
- Clear distinction between configuration data and operational data,
- Operators should be able to focus on the network as a whole, and not need to consider each device on its own, and
- The granularity of access control needed on management interfaces needs to match operational needs.

NETCONF is a transactional protocol, leading implementations to support network wide transactions. As an example, imagine a network service currently in development by a network operator. This service might be implemented over 20 different network elements, such as routers, firewalls, and load balancers. When deploying the service, the network operator wants to either have the service fully implemented, or if failing to configure one of the elements, then all devices should be rolled back to their earlier states.

This approach to configuring network devices avoids leaving incomplete configurations in network elements when something fails, therefore network operators do not have to manually perform roll backs via a CLI [13].

NETCONF is session based [15]. A session is created to connect a network operator to a network device. Configurations within a session can be local or global. Local configuration changes can only be seen from within the session, while global configuration changes are visible to other sessions as well [15]. NETCONF supports multiple sessions configuring the same device at the same time; however, not all network devices communicating over NETCONF support multiple sessions.

Due to the features listed above, using NETCONF lowers complexity and reduces the time needed for network operators to implement and deploy new network services [17]. Table 2.1 lists the different primitive operations defined by NETCONF and briefly explains their actions. These primitives can be combined to create complex configurations.

Table 2.1: Description of NETCONF operations.

| Operation | Description |
|---|---|
| get | Returns all variables. |
| get-config | Reads the configuration of a device. |
| edit-config | Edits the configuration of a device. |
| copy-config | Create a new configuration from an old. |
| delete-config | Deletes the configuration of a device. |
| lock | Locks a device so that other NETCONF clients can not modify it. |
| unlock | Unlocks a locked device. |
| close-session | Closes a NETCONF session. |
| kill-session | Forcibly kill a session in a non-graceful way. |

Listing 2.2, shows an example of a NETCONF notification that shows an error resulting from failure to provision a VM containing a firewall VNF. The structure of this NETCONF message is defined by a corresponding YANG model, this model is shown in Listing 2.3.

```
<notification
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2015-05-06T14:19:14-00:00</eventTime>
  <domain-creation-error
    xmlns="http://tail-f.com/ns/gabbleduck/libvirt-notif">
    <name>asa-vFirewall</name>
    <error>error-domcreate</error>
    <desc>internal error: process exited while connecting to
          monitor: Cannot set up guest memory 'pc.ram':
          Cannot allocate memory
    </desc>
  </domain-creation-error>
</notification>
```

Listing 2.2: NETCONF example.

### 2.2.3 YANG

YANG is a data modeling language accompanying NETCONF [16]. It describes configuration data and operational data expressed as a tree structure. One of its main benefits is that it can easily be read and interpreted by humans. Another of its main benefits is that YANG models and rules defined in these models, can be validated for correct semantics and syntax by tools [16].

YANG is a *schema language*, YANG models define the overall structure of a service, what database schema is needed, which configuration data can be read-write, and what operational data should be possible to get from the service. YANG files can be divided into modules, modules can be included in other modules as sub modules. This makes it possible to divide services into multiple files, where each file can have high cohesion, while the separate modules can have low coupling.

Since YANG models are schema documents, instance documents following /implementing schemas can be created. Instance documents can be defined and encoded using XML or JavaScript Object Notation (JSON). For example, these instance documents can describe how a router should be configured and where it should be provisioned.

Listing 2.3 shows a YANG model defining the structure of NETCONF notifications to notify a network operator that something has failed during the phase of creating a domain. This YANG model is what defines the structure of the NETCONF message shown in Listing 2.2.

```
notification domain-creation-error {
  leaf name {
    type leafref {
      path "/c:libvirt/d:domains/d:domain/d:name";
    }
  }

  leaf error {
    type gabbleduck-error;
  }

  leaf desc {
    type string;
  }
}
```

Listing 2.3: Example YANG model: NETCONF notification.

Listing 2.4 shows the YANG *container* statement, that is, a structure that can wrap other YANG statements. The container in this example has four children; name, sockets, cores, and threads. Each child has a type, types are checked and enforced by the tool that reads the YANG model. This example also shows the YANG *leaf* statement. This statement is placed at leaf positions in YANG trees. Leafs contain a value, but no children.

```
container topology {
    leaf name {
        type string;
    }
    leaf sockets {
        type uint32;
    }
    leaf cores {
        type uint8;
    }
    leaf threads {
        type uint32;
    }
}
```

Listing 2.4: Example YANG model: YANG container & leaf.

Listing 2.5 shows the YANG *list* statement. A list can have one or two key attributes, and these key attributes are what have to be unique in each child of the YANG list. The list statement is useful when one wants to model several children of the same type as a list, and distinguish these children with a key/index. The listing shows an example of a YANG model where network operators can add any number of reserved hosts to a DHCP server, as long as the MAC addresses of these hosts are unique. In this listing the mac address is the key that enforces all children of the host list to be unique.

```
container dhcp {
    list host {
        key mac;
        leaf mac {
            type string;
        }
        leaf name {
            type string;
        }
        leaf ip {
            type inet:ip-address-no-zone;
        }
    }
}
```

Listing 2.5: Example YANG model: YANG list.

There are two reasons for why NETCONF and YANG is used in this master's thesis project, instead of using plain JSON or YAML Ain't Markup Language (YAML) documents. The first reason is that Tail-f Systems has developed two software systems (ConfD and NCS, which are described

below) designed around NETCONF and YANG. The other reason is that a separation of the schema and instance documents, with the possibility of specifying type constraints and other constraints in the schema documents was desired. The author believes that this can reduce the number of errors network operators produce when designing network services by specifying instance documents.

### 2.2.4 ConfD

ConfD is an on-device software system that provides configuration management capabilities [20]. ConfD makes it possible for developers of network equipment, and other types of hardware and software, to provide a north bound (i.e., interface toward the management system) transactional management interface to devices.

Typically, a YANG model describes all configurable parameters available in a device, as well as operational data that an operator can ask a device to provide. As an example, configuration data could include which gateways the routing table of a router should contain, this is something network operators can add to and change over the course of the operation of the router.

Operational data, on the other hand, could include a counter indicating the total number of packets that the router has routed. This total packet count will increase during the life time of some network service. The operator can read this value at any time, for example, for billing purposes, computing packet rates, dimensioning of other resources, etc.

In the context of ConfD, YANG models serve as contracts for what is (and what is not) supported by the device being managed. These models are mirrored as database schemas and are implemented in a database management system named Configuration Database (CDB). Instance documents implementing the schema documents, are stored as values in these databases. These instance documents can change at runtime when operators load new YANG modules into ConfD.

Device manufacturers must implement the functionality necessary to handle all possible changes that can be made in the context of the YANG schema models. They must also send northbound NETCONF notifications, as defined by the YANG models they implement.

Figure 2.2 shows how this thesis project uses ConfD. Libvirt, described further in Section 2.3.3, is the controlled application. A custom ConfD subscriber is created, acting as a proxy between ConfD and Libvirt. YANG models are defined, specifying what capabilities Libvirt supports. NCS, detailed in Section 2.2.5 can control ConfD. In the figure, NCS is represented by the operator.

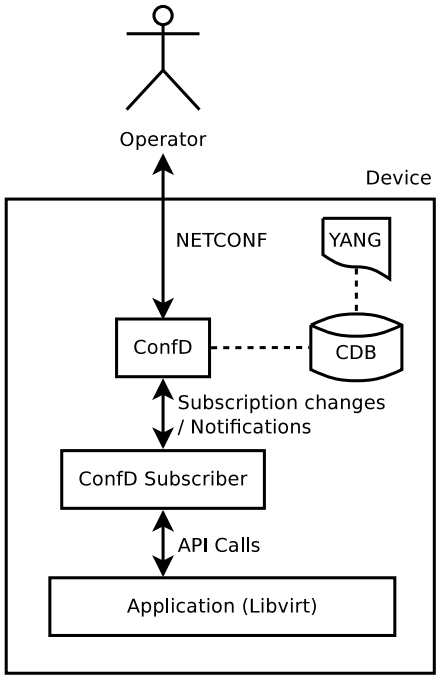


Figure 2.2: ConfD communication flow.

There are several other uses for ConfD, in which the role of this software component would be different. However, in the context of this thesis, the ConfD subscriber application is the most relevant to describe.

Listing 2.6, shows a YANG model where a simplified routing table maps destination addresses to gateway addresses. As described earlier, this will be used to create a database schema in CDB, where a simple table will be realized. The operator can then (using NETCONF or CLI) add or remove any number of rows in CDB.

```
module routing-table {
  namespace "http://acme.example.com/router/routing-table";
  prefix routing-table;
  organization "ACME, Inc.";

  description
    "A simplified router table.";

  grouping table {
    list row {
      key "destination";

      leaf destination {
        type inet:ipv4-address;
        description "Destination address.";
      }

      leaf gateway {
        type inet:ipv4-address;
        description "Gateway of associated address.";
      }
    }
  }
}
```

Listing 2.6: Example YANG model: Routing table.

The ConfD subscriber subscribes to changes in CDB, either to the entire database or only a sub part of the entire database. For instance, a given subscriber might only be interested in configuration changes to the destination addresses or only interested in changes to the gateways.

In either case, the ConfD subscriber will receive these configuration changes, and with its knowledge of the router, configure the router appropriately. The ConfD subscriber can use a CLI, Application Programming Interface (API) calls, or any other protocol that the router supports, to communicate with the router in order to implement these configuration changes.

Using this flow of communication, any network device can be configured via network wide transactions, as long as there are (1) YANG models specifying the configuration capabilities of that network device and (2) a ConfD application that can configure the network device based on existing contracts. These two requirements are important to remember throughout the remainder of this thesis.

### 2.2.5 NCS

NCS, is similar to ConfD in that it configures devices based on the configuration parameters defined in YANG models. However, NCS is more commonly used as an orchestration tool. NCS consists of a *device manager*, and a *service manager*, amongst other things [21].

The device manager is capable of configuring sets of network devices. NCS has a *device tree* in which it can add/delete devices to manage, and by performing all management operations through NCS in this way, NCS also keeps track of all of the device and application configurations. This is in contrast to ConfD, that normally manages only one device.

The NCS service manager accommodates *service applications*. These are applications that can orchestrate or configure certain services, for instance a VPN service. Services can include any number of devices. The *Weaver* application, described in Section 4.1.2, is a collection of NCS services [21].

Note that, in the context of this thesis project, there is a client-server relationship between NCS and ConfD. NCS is the client and ConfD is the server. NCS asks ConfD to perform certain actions and listens for the results of these actions. However, after the initialization of devices, NCS can add provisioned devices to its own device tree and can then configure devices without the need of doing it through ConfD.

The objects symbolizing ConfD subscribers in the Figure 2.3, are instances of the flow described in Figure 2.2, with the network operator replaced by NCS. Both ConfD and NCS automatically generate northbound management interfaces from provided YANG models. These management interfaces include both web user interfaces and CLI. Figure 2.4 shows the relationship between ConfD and NCS in more detail. This figure combines objects from Figure 2.2 and 2.3.

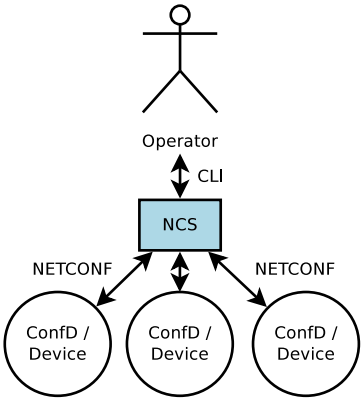


Figure 2.3: NCS communication flow.

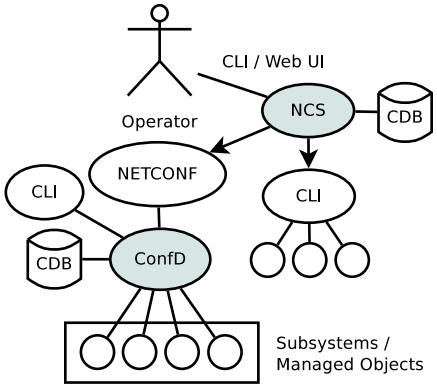


Figure 2.4: NCS and ConfD.

## 2.3 Related Work

This section discusses related work. Alternatives to the work conducted in this thesis, as well as additional background information about the technology used in the thesis project, are briefly discussed.

### 2.3.1 Open vSwitch

Linux has three types of software switches: *bridge*, *macvlan*, and *Open vSwitch* (OVS) [22]. Bridges can be used to connect Ethernet NICs together through a common virtual network [23]. This makes it possible for these NICs to communicate over this network. Macvlan allows a single NIC to be associated with several IP addresses and MAC addresses.

OVS is a virtual, *link layer* switch, designed to interconnect VMs [24]. It is not executing as a VM, rather, it is an application executing on top of the OS of a physical computer. OVS is used by OpenStack (introduced in Section 2.3.2) to create connections between VMs. Moreover, OVS is designed to be a distributed switch, it can execute on many host computers and communicate between instances. This functionality, is something that Linux bridges do not provide [25]. OVS bridges are more advanced than Linux bridges. Forwarding is flow based (OVS supports the OpenFlow [26] protocol), and has caching capabilities [22]. OVS can be used to simulate routing functionality [27]. When comparing Gabbleduck with OpenStack (described in Section 2.3.2), OVS is one of the components that is of interest.

### 2.3.2 OpenStack

OpenStack is one of the most commonly used open source cloud management systems at the time of writing this thesis [28]. OpenStack was created by NASA and Rackspace in order to create a free open source cloud operating system for use in private clouds. Although Amazon at the time of the OpenStack project initialization already offered Amazon Web Services (AWS), a public cloud service, there was a need for software to manage

private clouds.

OpenStack is composed of several more or less independent software components. *Nova Compute* provisions VMs, *Glance* stores machine images, *Heat* orchestrates VMs, and *Neutron* is OpenStack's networking component. Neutron and Heat are of special relevance to this thesis because in Section 5.1 the Gabbleduck system will be compared to OpenStack and these specific OpenStack components. Section 2.3.2.1 describes Heat, and Section 2.3.2.2 describes Neutron.

#### 2.3.2.1 Heat

OpenStack's orchestration software component is Heat [28]. Heat reads Heat Orchestration Template (HOT), YAML, and Amazon CloudFormation templates. It uses these templates to provision and create VMs and network topologies.

Listing 2.7 shows an example of a Heat YAML instance document. The type of a template (its Template Format Version) is specified, followed by a set of properties expressed as YAML objects. A virtual network *network1* is defined. To it, a subnet *VLan1* is attached. This subnet is given an address space, in which a DHCP server can allocate IP addresses. Finally, a Neutron port *port1* is defined. This port allocates an IP address and we can later connect this port to any VM, so that the VM is reachable through its own IP address.

When Heat reads this orchestration template, it will determine what actions to take in order to realize the components defined by the template. It will make API calls to Nova in order to provision needed VMs and uses Glance to access VM images. Neutron might be called in order to configure OVS or other network functions, like in the example described above.

```
HeatTemplateFormatVersion: '2012-12-12'

Description: Simple Network Topology

Resources:
  network1:
    Type: OS::Neutron::Net
    Properties: {name: net1}
  VLan1:
    Type: OS::Neutron::Subnet
    Properties:
      network_id: {Ref: network1}
      ip_version: 4
      cidr: 10.0.0.0/24
      allocation_pools:
      - {start: 10.0.0.5, start: 10.0.0.250}
  port1:
    Type: OS::Neutron::Port
    Properties:
      name: port1
      network: {Ref: VLan1}
      fixed_ips:
      - subnet: public-subnet
        ip_address: 10.0.0.20
```

Listing 2.7: HOT network template example.

Listing 2.8 shows a VM created and configured through OpenStack's Nova component. It is configured to boot a Ubuntu 12.04 cloud image. Cloud images are different from normal images in the sense that when booting, they look for initial configurations, or Day0 configurations. An example of providing a Day0 configuration through OpenStack can be seen in the *user_data* field of the model, where a password is set in the provisioned VM. The VM is connected to the port and network defined earlier.

```
HeatTemplateFormatVersion: '2012-12-12'
  instance1:
    Type: OS::Nova::Server
    Properties:
      name: ExampleVM
      image: "ubuntu-12.4"
      flavor: m1.small
      networks:
      - network: {Ref: network1}
        port: {Ref: port1}
      user_data: "#cloud-config
                  password: passw0rd"
      user_data_format: RAW
```

Listing 2.8: HOT instance example.

#### 2.3.2.2 Neutron

Neutron is a software component providing network function interactions to OpenStack applications [27]. It provides some basic SDN functionality. Users can implement logical networks with logical routers and logical firewall policies. *Logical*, in this context, means there is no physical or even dedicated VM instance providing a firewall in the network. Instead, as mentioned in Section 2.3.1, underlying software such as OVS is configured to provide this functionality [27]. Figure 2.5 shows how OpenStack Neutron could use OVS to interconnect a number of network appliances.

Neutron creates logical routers as a set of OVS configurations, specifying what ports to route to different gateways. Logical networks are identified as IP address ranges with a subnet masks, for instance 10.0.1.0/24. This makes it possible to partition virtual networks. Logical routers can be placed between logical networks to provide Network Address Translation (NAT), just as can be done in physical networks.

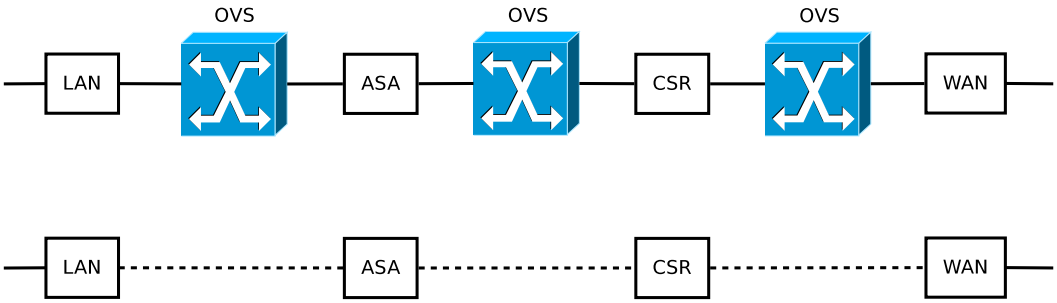


Figure 2.5: Neutron OVS usage example.

Neutron's programmability and configurability, is sufficient for many cloud applications. However, this thesis claims that Neutron is insufficient when one wants to use specific VNFs, such as Cisco or Juniper proprietary virtual routers. Currently it is not possible to provision arbitrary VNFs using Neutron, without first developing a specific plug-in for each such VNF to be used in a desired topology.

Figure 2.6 shows a simple network provisioned by Neutron. The figure depicts an OpenStack *project*, with two private networks, *Vlan1* and *Vlan2*. Vlan1 is connected to the outside world (outside of the project) through *router1* and Vlan2 through *router2*. The model defining this topology is shown in Listing A.1.

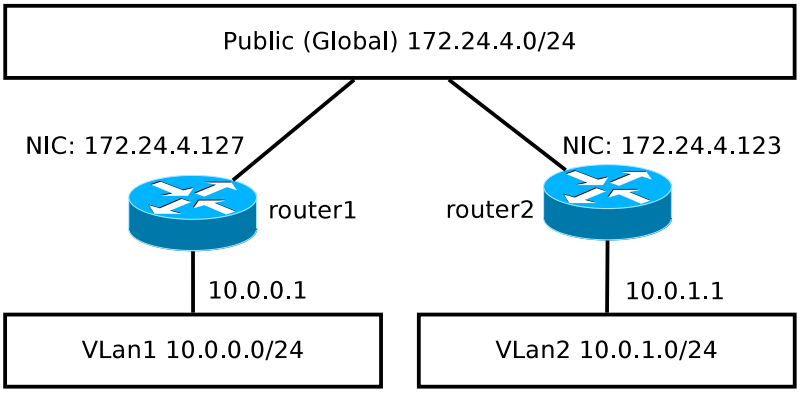


Figure 2.6: Example network topology provisioned by Neutron.

Table 2.2 shows the logical networks created by Neutron in order to realize the network described above. First is the public network, to which the project is facing, and then two private networks accessible through the logical routers.

Table 2.2: Neutron Network List

| **id** | **name** | **subnets** |
|---|---|---|
| 9a87b5... | public | 06e82... 172.24.4.0/24 |
| 0070cb... | Vlan1 | 7d63b... 10.0.0.0/24 |
| f50848... | Vlan2 | 8b469... 10.0.1.0/24 |

Table 2.3 shows the list of logical routers present in OpenStack when the topology depicted in Figure 2.6 is implemented. Each router is connected to a list of subnets.

Table 2.3: List of Logical Routers

| **id** | **name** | **external_gateway_info** | **distributed** | **ha** |
|---|---|---|---|---|
| 8ff12... | router2 | {”net_id”: ”9a87b...”, ”ips”: [ {”subnet_id”: ”06e82...”, ”ip_address”: ”172.24.4.5”}]} | False | False |
| fc930... | router1 | {”net_id”: ”9a87b...”, ”ips”: [ {”subnet_id”: ”06e82...”, ”ip_address”: ”172.24.4.3”}]} | False | False |

Neutron provides functionality to configure a Firewall as a Service (FWaaS) and other network functions as services. However, the network operator is limited in how these services can be configured and controlled. Just as with logical routers, FWaaS instances are not implemented as specific VMs. Rather, they are are implemented as rules in all of the logical routers across a project. These rules are translated to OVS configurations, just as the routers are themselves. The OVS configurations for this example are listed in Listing A.3 in the Appendix.

### 2.3.3 Libvirt

This thesis project uses Libvirt [29] as an abstraction layer to control the underlying physical machines, and to control Virtual Machines. Libvirt manages hypervisors and these hypervisors in turn interface with the operating system of the underlying physical machines. In Libvirt terms, a VM is a *domain*. Domains can be created, destroyed, migrated, etc.

Libvirt offers both a CLI and an API to instruct Libvirt what to do with different domains. Libvirt also manages network interfaces and pools of storage volumes. Domains typically store images which they use as hard drives in these pools. The Libvirt API communicates using data encoded as XML. Figure 2.9 shows an example definition of a virtual network. This virtual network is bridged to the host machine's NIC over a Linux bridge. NAT is performed between the host executing Libvirt, and the address range defined as *192.168.122.0/24*. IP addresses within the virtual network, ranged from 2 to 254, can be allocated.

```
<network>
    <name>default</name>
    <bridge name="virbr0"/>
    <forward mode="nat"/>
    <ip address="192.168.122.1" netmask="255.255.255.0">
        <dhcp>
            <range start="192.168.122.2" end="192.168.122.254"/>
        </dhcp>
    </ip>
    <ip family="ipv6" address="2001:db8:ca2:2::1" prefix="64"/>
</network>
```

Listing 2.9: Libvirt network definition.

Figure 2.10 shows an example definition of a Libvirt domain. It defines a domain, based on a provided Ubuntu image, within the Quick Emulator (QEMU) hypervisor. The domain is attached to the virtual network defined above. When booted, the domain will mount a virtual CD-ROM provided as an ISO image (day0.iso).

```
<domain type='qemu'>
  <name>ubuntu</name>
  <memory>219200</memory>
  <vcpu>2</vcpu>
  <os>
    <type arch='x86_64' machine='pc'>hvm</type>
    <boot dev='hd'/>
  </os>
  <devices>
    <emulator>/usr/bin/qemu-system-x86_64</emulator>
    <disk type='file' device='cdrom'>
      <source file='/home/acme/day0.iso'/>
      <target dev='hdc'/>
      <readonly/>
    </disk>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2' cache='none'/>
      <source file='/home/acme/ubuntu.img'/>
      <target dev='hda' bus='ide'/>
    </disk>
    <interface type='network'>
      <source network='default'/>
    </interface>
  </devices>
</domain>
```

Listing 2.10: Libvirt domain definition.

### 2.3.4 OASIS TOSCA

TOSCA is defined by the OASIS standardization organization. It is a template language that can be used to define instance templates for cloud environments [30], much like HOT and Amazon CloudFormation. TOSCA uses XML, and more recently, YAML, to encode data. It defines a number of models, or templates, that can be combined to describe software components and relations between these components.

Figure 2.7 shows different templates and types within TOSCA. Service templates contain information about entire services that can consist of many nodes, mostly network devices in the context of this thesis. Service templates can also contain plans, plans are rules or definitions for how to perform orchestration.

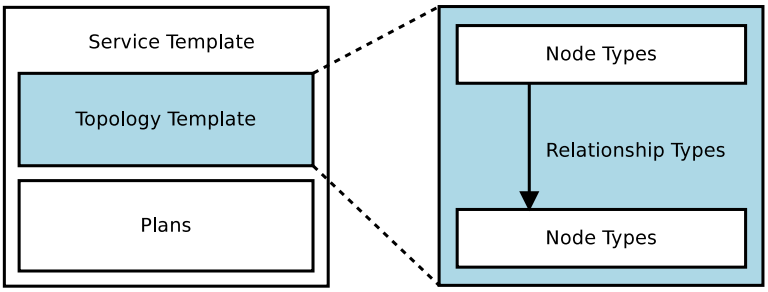


Figure 2.7: TOSCA components overview.

Topology templates model nodes in services, and relationships between nodes. The relationship can denote, for instance, what type a node is, and where this node should be hosted. Models can then be formed and given to an orchestrator, that transforms these models into a set of deployed instances [30]. Figure 2.8 shows an example topology defined using TOSCA. It is a service template, defining both a set of node types, and a topology.

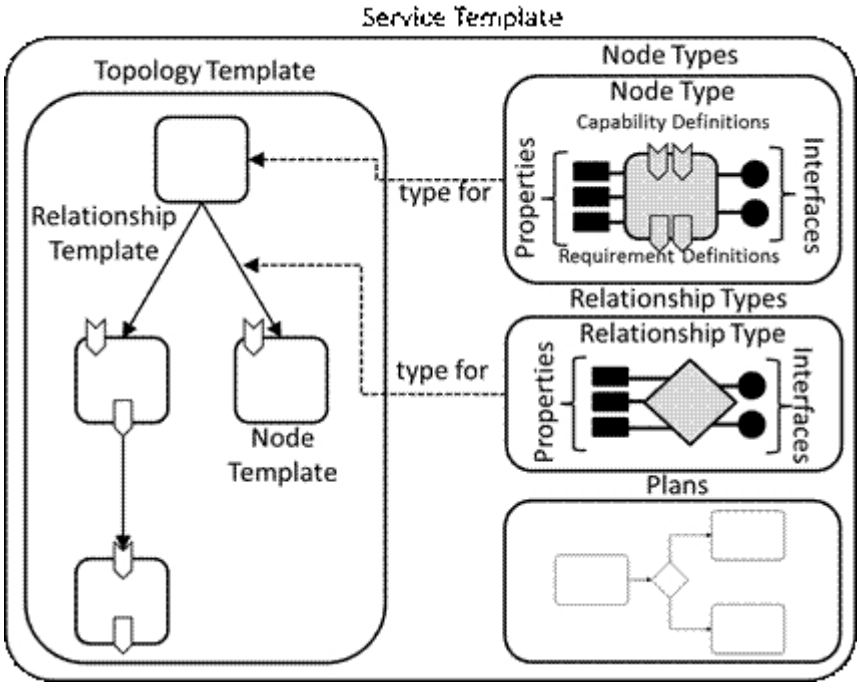


Figure 2.8: TOSCA service template. (Adapted from Figure 1 in [31].)

Listing 2.11 shows a basic TOSCA template that defines two nodes. There is also a relationship defined, connecting the two defined nodes.

```
<Definitions id="MyDefinitions" name="My Definitions"
  targetNamespace="http://www.example.com/MyDefinitions"
  xmlns:my="http://www.example.com/MyDefinitions">
  <Import importType="http://www.w3.org/2001/XMLSchema"
          namespace="http://www.example.com/MyDefinitions">
  <NodeType name="Application">
    <PropertiesDefinition element="my:ApplicationProperties"/>
  </NodeType>
  <NodeType name="ApplicationServer">
    <PropertiesDefinition
      element="my:ApplicationServerProperties"/>
  </NodeType>
  <RelationshipType name="ApplicationHostedOnApplicationServer">
    <ValidSource typeRef="my:Application"/>
    <ValidTarget typeRef="my:ApplicationServer"/>
  </RelationshipTemplate>
</Definitions>
```

Listing 2.11: TOSCA definitions document (Adapted from Example 4.3 in [30]).

### 2.3.5 FBOSS

Facebook Open Source Switch (FBOSS) [32], is another approach at building and managing networks. It consists of open source hardware and open source software. The hardware includes an Application-Specific Integrated Circuit (ASIC): the Broadcom Trident II ASIC. The software executes on top of Linux. In this way, the switch is designed as a UNIX server. Additionally, the resulting switch is managed as a server and not simply as a network device.

In the FBOSS architecture, the hardware and software are not tightly bundled together. Instead, the project separates hardware and software in the switches [32]. As a reminder, this is also what NFV proposes. FBOSS can conceptually be placed between traditional networking equipment and NFV, since it divides hardware and software but still provides specific hardware.

## 2.4 Summary

NFV is a concept that proposes to add virtualized network equipment to standard data center hardware. This can reduce costs, save energy, and reduce the time needed by network operators when provisioning and managing networks.

NFV orchestration places new requirements on cloud management systems. The management system needs to have application layer information available in order to make informed decisions. The network operator must have access to configuration data such as the placement of specific devices and other resources. The network management module of the cloud management system needs to be able to connect to devices and configure them according to both their initial and subsequent configuration settings.

## Chapter 3:

# Method

This chapter provides an overview of the research methods used in this thesis. Section 3.1 describes the research process. Section 3.2 discusses the research paradigm. Section 3.3 explains which project method was used in this thesis project. Section 3.4 focuses on the data collection techniques used during this research.

## 3.1 Research Process

Looking back at Section 1.3, the purpose of this master's thesis project is to evaluate whether the Gabbleduck prototype improves the productivity of network operators when designing network services. A mix of *quantitative* and *qualitative* research methods are used. Some aspects can be measured with statistical data. In addition to this, interviews and case studies are carried out to gather thoughts and opinions.

The abductive method is used when performing observations and case studies of network operators. The abductive method uses both inductive and deductive methods to travel from a set of observations to a set of likely conclusions [33]. This approach is used to collect opinions and concepts that might otherwise be missed out when carrying out experimental methods. When measuring performance parameters, experimental and deductive approaches are used. These methods are helpful when gathering statistical data.

## 3.2 Research Paradigm

This master thesis embraces the realistic [33, 34, 35] philosophical assumption. The realistic assumption is helpful when performing observation of research participants to collect opinions of working with the evaluated systems.

In addition to realism, positivism is assumed when carrying out performance tests. Positivism assumes that the reality is objectively given and independent of the observer and instruments used. Positivism is commonly assumed when using quantitative methods. Criticalism is not relevant to this research, since it focuses mainly on the way society is structured and on cultural aspects [33].

## 3.3 Project Method

The project is carried out in an iterative manner. This suits the project, since initially the requirements were unclear. The understanding of the project, the requirements, and what are good and bad solutions evolved rapidly in the early stages of this project, and continued to evolve throughout the rest of this master's thesis project.

An alternative would be the waterfall method [36]. In this style of project management, parts of the projects are developed in sequence, with each step depending on the previous step. However, in order for this method to be successful the requirements must be clear from the beginning. As the requirements were not clear, the waterfall method was deemed to be inappropriate [36].

## 3.4 Data Collection

In order to evaluate the systems researched in this master's thesis project, data is collected from experiments, case studies, and interviews. The collection of data is divided into four parts, each part is explained in this section. Figure 3.1 gives an overview of how each part of the data collection relates to the methods described above. It also shows in which order these data collections were performed.

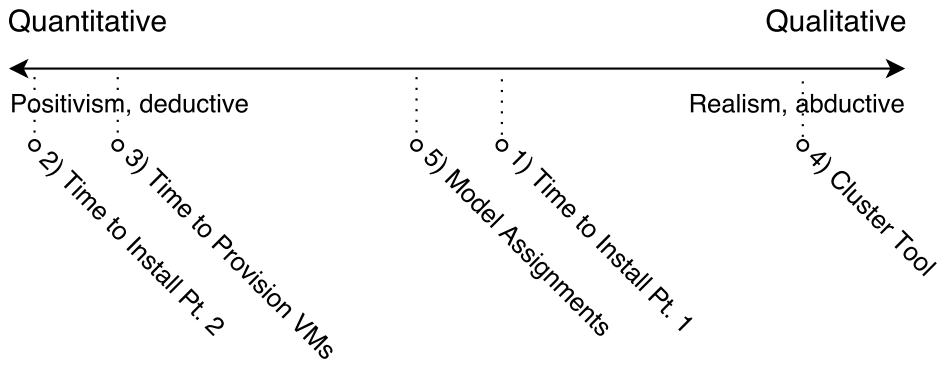


Figure 3.1: Overview of methods used in evaluation.

### 3.4.1 Time to Install

*Time to Install Pt.1* consists of case studies of volunteer participants. This measures the amount of time required for these volunteer network operators to install the evaluated software on their work computers, and also notes issues encountered.

*Time to Install Pt.2* measures the amount of time required to install the evaluated systems on a host computer, assuming that no issues are encountered during the installations. The UNIX command *time* is used to measure the amount of time (in seconds) required to install each system. The results of these measurements and observations are presented in Section 5.1.1.

### 3.4.2 Time to Provision VMs

Both evaluated systems are used to provision VMs in groups from one to ten VMs. Then, the amount of time required for the VMs to respond to ping and SSH connections are measured. Two time stamps are collected for each set of VMs measured. These time stamps represent (1) the wall clock time when starting to provision the set of VMs, and (2) the wall clock time when the last VM answered to ping, as well as SSH connections. The results of these measurements are presented in Section 5.1.2.

### 3.4.3 Cluster Tool

Thoughts, opinions, and issues are collected from two network operators working on a project that uses the Gabbleduck software as part of its tool chain. An interview with these engineers was carried out. This interview, and the outcomes of it, is explained in Section 5.1.3.

### 3.4.4 Model Assignments

In order to measure the amount of time required for network operators to design network services, three short (in terms of time to complete) assignments are created. These assignments ask a set of volunteers to create or modify provisioning templates (such as to introduce new functionality to services) for both Heat and Gabbleduck.

This way, the complexity of the models, and the complexity of changes to the models, can be quantified in terms of the number of required changes, the difficulty of these changes, and how much time it takes to perform these changes. Additionally, this shows whether it is possible to perform these changes in the first place. Data is collected by asking participants to measure the amount of time they need to solve assignments. In addition, participants hand in their solutions, so that they can be analyzed, and so that the investigator can count the amount of failures.

During data collection, there is the risk that participants will be affected and biased by the order in which they use the systems evaluated. This is a

risk difficult to eliminate, however it is taken into account and reduced by dividing participants in two groups and having these two groups solve their own version of the assignments.

The difference of these two versions, is the order in which sub assignments are solved. In the first version, an assignment involving Gabbleduck should be solved first, and an assignment involving OpenStack should be solved later. In the second version, this is reversed. Median results of the combined groups are analyzed.

In addition to the problem of learning order effects, the results are also affected by bias from if participants have worked with OpenStack before, and if they have a good understanding of YANG. Participants with a good understanding of YANG will likely perform significantly better than participants who do not understand the concepts of YANG, during the parts of the evaluation concerning Gabbleduck.

Participants who do not have the understanding of YANG, will instead likely be slowed down in having to understand the link between YANG model and instance document. The assignments are described further in Section 5.1.4.

### 3.4.5 Sampling

In order to test the model assignments described above, the investigator has to collect a group of participants to assist in the evaluation. When this group of participants is collected, Simple Random Sampling (SRS) [37] is applied, out of a group of available participating volunteers provided by Cisco, Inc.

The reason as to why SRS is used, is because the time to carry out this project and the budget for this project are both limited, and because SRS avoids introducing bias. There are more advanced types of sampling, one of them is stratified sampling. In stratified sampling, the population of available potential participants would be divided (on number of years of managing networks, or on other differentiators) into groups, or strata. These strata are then each sampled.

### 3.4.6 Sample Size

When collecting data from the installation time experiment, the test is repeated ten times to give sufficiently reliable results. The experiments designed to provision VMs are significantly more time consuming to carry out, therefore each test (there are twenty tests in total) is repeated only three times. However, three times is expected to be enough to yield relatively stable results, since the variations between tests are low.

The author was assigned eight network operators to volunteer in solving assignments. These network operators are employed by Cisco, Inc. In order to determine the significance level and power when given a population of eight participants, the R programming language [38] was used to perform a power analysis.

In the context of a power analysis in the R programming language, *effect size*, *significance level*, and *power* are relevant concepts. Significance level and power describe the probability of errors in the sampling. Significance level is the probability of finding an effect that does not exist. Power is the probability of finding an effect that does exist. Effect size [39] is a measure of how strong a measurement is.

Listing 3.1 shows the results of a power analysis. The sample size is four (four participants in each groups, since eight were available), when using two groups, a significance level of 95% and a power of 16%. Effect size is 0.8, which is weak. A stronger effect size would have been preferable.

```
> library(pwr)
> pwr.anova.test(k=2,f=.40,n=4)

     Balanced one-way analysis of variance power calculation

              k = 2
              n = 4
              f = 0.4
      sig.level = 0.05
          power = 0.1601134

NOTE: n is number in each group
```

Listing 3.1: Sample size calculation results.

Listing 3.2 shows a more desirable sample size, with a significance level of 95%. This requires the sample size to be 26 participants, in each group (where two groups would be used). However, in the scope of this research, gathering 51 volunteers is not possible.

```
> pwr.anova.test(k=2, f =.40 , sig.level =.05 , power =.8)

     Balanced one-way analysis of variance power calculation

              k = 2
              n = 25.52457
              f = 0.4
      sig.level = 0.05
          power = 0.8

NOTE: n is number in each group
```

Listing 3.2: Desirable sample size calculation results.

### 3.4.7 Target Population

The target population is network engineers and operators, individuals maintaining and developing network architectures and services. These persons are likely employed by large or medium sized network operators.

## 3.5 Test Environment

The tests assigned to participants are a set of text files. Participants are free to use the environment they are most comfortable with. Participants are not able to realize their models in the actual systems, instead they have to ask the investigator to validate and accept their solutions, when they think they are ready to do so. Hardware specifications differ between tests, thus they are described where each part of the evaluation is presented in Section 5.1. During the evaluation stages, a set of VNFs was provided by Cisco, Inc., these VNFs are:

- **Cisco Adaptive Security Virtual Appliance (ASAv)**, a firewall and content filter,
- **Cisco Cloud Series Router 1000v**, a router, and

- **Ubuntu 12.04 Server AMD64 (Cloud)**, a version of Ubuntu that is minimal in size and has the ability to load specified configurations when booting.

## 3.6 Assessing Reliability & Validity of the Data Collected

Reliability and validity both depend on how well the researcher manages to collect data during interviews and observations [35]. It also depends on how many times tests are repeated to reach statistically significant results.

### 3.6.1 Reliability

The reliability of the results of measuring the amount of time required to install the evaluated systems on a host computer, is the most reliable data collected. This is because each test is repeated ten times.

In the case of evaluation using volunteering participants, if the difference in the results collected from different volunteers is high, then this indicates that the collected data has low reliability [35]. It means that a larger sampling size participants would have been needed in order to reach a point where the median results data stabilize.

### 3.6.2 Validity

Validity is used to measure the truthfulness of collected data [35]. The results gathered when researching a phenomenon should be repeatable. If the same data is collected several times, then the results of these collections should be similar [35]. Important aspects of validity are; integrity, authenticity, credibility, and criticality [40].

Criticality and integrity are connected to topics such as investigator bias, and validity of the results acquired [40]. The investigator has to be self-critical in how data is collected, the importance of how the assignments are designed, and how this design affects the data collected.

## Chapter 4:

# Gabbleduck

This chapter introduces the system that was implemented as a part of this master's thesis project, the Gabbleduck. Section 4.1 explores the architecture of the system, to give the reader an idea of what the system does. Section 4.2 presents examples to give a more applied understanding of it.

## 4.1 Architecture

Gabbleduck is a software artefact that is designed to fill the gap between cloud management systems and SDN tools, as described in Section 1.2. It can provision VNFs based on machine images and connect to these VNFs over SSH to configure them. However, this requires that YANG models and controllers are defined for these VNFs.

Figure 4.1 depicts the architecture of the Gabbleduck system. The network operator can connect either to a service named *Weaver* or to a service called *Erlvirt*. Weaver can also connect to Erlvirt in order to control this service. Typically, one Erlvirt instance is executing on each host that the operator wants to control.

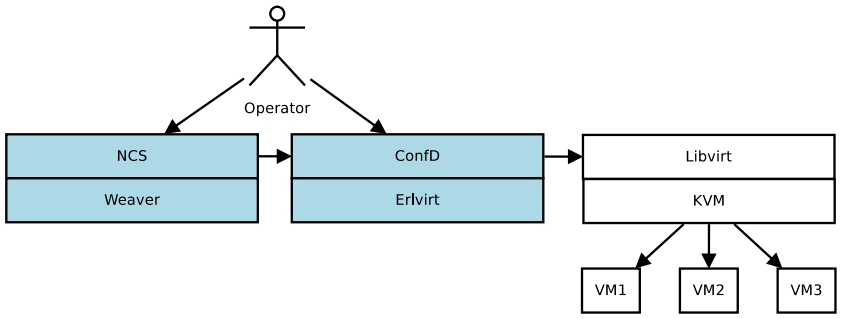


Figure 4.1: Gabbleduck architecture overview.

### 4.1.1 Erlvirt

Erlvirt is a ConfD subscriber application. It subscribes to changes in the CDB, and then iterates through each received set of changes. Erlvirt also connects to Libvirt, through Libvirt's remote API. The main purpose of Erlvirt is to transform CDB changes to Libvirt XML definitions and Libvirt API calls. Since Erlvirt can do this, it is possible to specify YANG instance documents describing what kind of VM to provision along with its configurations. Then, Erlvirt will realize these models.

When provisioning a VNF, such as a virtual router, it is common to specify a Day0 configuration. This Day0 configuration will be attached to the VNF, and is used in part to set up a management IP address in the VNF. This management address is later used by configuration tools to connect to the VNF, so that it can be configured further.

### 4.1.2 Weaver

The Weaver is a set of NCS applications, used to control one or more Erlvirt instances. NCS communicates with ConfD over the NETCONF protocol.

When Erlvirt has successfully provisioned a VNF, it sends a northbound NETCONF notification informing the Weaver about this life cycle event. Weaver then proceeds to load the device's configuration into its device tree. From this point and onwards, the Weaver can connect to the VNF's management address over SSH and configure it.

As described in Section 2.2.5, Weaver can add VMs directly to its device tree. This makes it possible for Weaver to have a complete overview of all managed VNFs (and these VNFs' configurations) in one central system. This also removes the need of communicating with VNFs through Erlvirt, instead Weaver can connect directly to VNFs.

## 4.2 Examples

This section shows the functionality and some use cases of the Gabbleduck system. It does this by introducing some example YANG models, and YANG instance documents.

### 4.2.1 Domain

The YANG model *libvirt-domain* defines all configuration data that the Gabbleduck supports, in terms of provisioning domains. An outline of the structure of this model is shown in Listing A.4.

Since Gabbleduck is a layer on top of Libvirt, the libvirt-domain model builds upon the configuration capabilities of Libvirt. When Erlvirt receives a change notification from the CDB, it queries CDB asking to get the entire domain instance document formatted as XML. Then, Erlvirt transforms this XML to the format that Libvirt expects, and passes this transformed XML to Libvirt. In this process of transforming the instance document, Erlvirt might also download VM image files from a web server hosted by Weaver and upload these image files to Libvirt's volume pool. In these cases, Erlvirt will then proceed to change the URL from the instance documents to the local address the image file got in the host machine.

Listing C.2 shows an example of an instance document used to provision a simple Ubuntu cloud machine. This Ubuntu machine is connected to a virtual network that will be presented below. The instance document specifies how much virtual RAM the VM should allocate, as well as how many virtual CPUs to make use of. This VM will mount a hard drive named *boot-disk*, this hard drive contains a machine image with an OS (contained in the ".img" file). Additionally a virtual CD-ROM containing initial configuration for the OS is mounted. This configuration is defined as *ubuntu-day0* in the instance document.

### 4.2.2 Monitoring

The Gabbleduck system includes a software component that allows monitoring of VMs, VNFs, and physical machines. This monitoring feature currently supports ping and SSH probing of VMs. Monitoring allows the network operator to receive notifications about the life cycle of VNFs. Normally, a VNF is reachable by ping requests before it is reachable over SSH. This component will be mentioned again in Section 5.1.2 when discussing and presenting analysis of the system. Listing 4.1 features an instance document, showing how a monitor can be created and configured to monitor the IP address *192.168.0.2*, by pinging this address every 50 milliseconds. SSH monitoring can be added by copying the *monitor* XML tag and replace *ping-monitor* with *ssh-monitor*.

```
<config xmlns="http://tail-f.com/ns/config/1.0">
  <lwmon xmlns="http://www.cisco.com/lwmon/lwmon">
    <notification-stream>gabbleduck</notification-stream>
    <nodes>
      <node>
        <name>Monitor</name>
        <addresses>
          <address>
            <address>192.168.0.2</address>
            <monitors>
              <monitor>
                <label>ping</label>
                <actions>
                  <action>
                    <label>ping-monitor</label>
                    <notif/>
                  </action>
                </actions>
                <ping-monitor>
                  <enabled>true</enabled>
                  <poll_interval>50</poll_interval>
                  <active_threshold>1</active_threshold>
                  <inactive_threshold>8</inactive_threshold>
                </ping-monitor>
              </monitor>

              ...

            </monitors>
          </address>
        </addresses>
      </node>
    </nodes>
  </lwmon>
</config>
```

Listing 4.1: Gabbleduck monitor example.

### 4.2.3 Network

The YANG model *libvirt-network*, defines the available configuration parameters of virtual networks within the Gabbleduck system. An outline of the structure of this model is shown in Listing A.5. Listing 4.2 shows a YANG instance document defining a virtual network configuration.

This virtual network is connected to the host machine's NIC through a Linux bridge named *virbr0*. Gabbleduck does not yet have a binding to OVS, and therefore Linux bridges are used. NAT, between the NIC's CIDR and *192.168.122.0/24* is performed on all ports between 1024 and 65535.

The gateway of the virtual network is defined as *192.168.122.1*. A certain sub space of the network is allocated to a DHCP server, so that connecting VMs can be assigned IP addresses. Finally, one IP address (192.168.122.29) is preallocated to a VM with a certain MAC address.

```
<config xmlns="http://tail-f.com/ns/config/1.0">
  <network>
    <name>default</name>
    <bridge>
      <name>virbr0</name>
      <stp>on</stp>
      <delay>0</delay>
    </bridge>
    <forward>
      <mode>nat</mode>
      <nat>
        <port>
          <start>1024</start>
          <end>65535</end>
        </port>
      </nat>
    </forward>
    <ip>
      <name>default</name>
      <address>192.168.122.1</address>
      <family>ipv4</family>
      <netmask>255.255.255.0</netmask>
      <dhcp>
        <range>
          <start>192.168.122.100</start>
          <end>192.168.122.254</end>
        </range>
        <host>
          <mac>00:0a:95:9d:68:12</mac>
          <name>VNF1</name>
          <ip>192.168.122.29</ip>
        </host>
      </dhcp>
    </ip>
  </network>
  </networks>
  </libvirt>
</config>
```

Listing 4.2: Gabbleduck network definition example.

Listing 4.3 shows the result of provisioning the above model using Gabbleduck. A Linux bridge is implemented, bridging the virtual network with the host machine's NIC. This listing shows the bridge as seen by the UNIX *ifconfig* command.

```
> ifconfig
virbr0 Link encap:Ethernet HWaddr 52:54:00:cd:96:5e
       inet addr:192.168.122.1 Bcast:192.168.122.255
            Mask:255.255.255.0
       UP BROADCAST MULTICAST MTU:1500 Metric:1
       RX packets:0 errors:0 dropped:0 overruns:0 frame:0
       TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:0
       RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Listing 4.3: Linux bridge implementation.

### 4.2.4 CSR1000v VNF

The Cisco Cloud Series Router 1000v (CSR1000v) is an example of a VNF. It is a router that supports BGP, VPN tunnels, etc. An instance document expressing how to provision a CSR1000v VNF is shown in Listing A.6. This model defines a number of virtual networks, a number of virtual NICs, and the domain that will execute the actual CSR1000v software. The purpose is to define networks, and interconnect these networks through the router.

Mounted to the domain, is an initial router configuration, embedded as *iosxe_config.txt*. This configuration is applied before the router provides a management IP address that network operators can connect to. The purpose of this initial configuration is to create such a management address. The content of this configuration file is illustrated in Listing A.8.

Once a management IP address has been configured, and this interface listens to traffic, it is possible to connect NCS to the router. NCS can then configure more advanced options within the VNF. An example of how to set up NAT, routes, and access lists, is provided in Listing A.7. Listings 4.4 shows a key part of this listing, specifically, a list of route forwards that can be specified. Prefixes and masks correspond to default gateways. The variables, such as DEFAULT_GATEWAY are replaced by Weaver.

```
<route>
  <ip-route-forwarding-list>
    <prefix>0.0.0.0</prefix>
    <mask>0.0.0.0</mask>
    <forwarding-address>{$DEFAULT_GATEWAY}</forwarding-address>
  </ip-route-forwarding-list>
</route>
```

Listing 4.4: CSR1000v route definitions.

Listing 4.5 shows how NICs can be defined and configured. In this instance document, two NICs are defined. Together with them, groups defining inbound and outbound access control lists are defined. These interfaces are used when performing NAT for the Gigabit Ethernet interface "2".

```
<interface xmlns="urn:ios"  tags="merge">
  <GigabitEthernet>
    <name>1</name>
    <ip>
      <access-group>
        <direction>in</direction>
        <access-list>101</access-list>
      </access-group>
      <nat>
        <outside/>
      </nat>
    </ip>
  </GigabitEthernet>
  <GigabitEthernet>
    <name>2</name>
    <ip>
      <nat>
        <inside/>
      </nat>
    </ip>
  </GigabitEthernet>
</interface>
```

Listing 4.5: CSR1000v network interfaces.

## 4.3 Summary

The Gabbleduck system can provision and configure VNFs. Network operators need to provide instance documents defining single VNFs and complete services including several VNFs, in order for Gabbleduck to create and configure network services. Initially, VNFs are provisioned by Erlvirt and Libvirt, then NCS can connect to these VNFs to configure them. NCS can also connect to physical network devices that are part of network services, to configure these network devices too.

## Chapter 5:

# Analysis

This chapter discusses how Gabbleduck and OpenStack were analyzed. It presents major results in Section 5.1. Reliability analysis and validity analysis is presented and motivated in 5.2.

## 5.1 Major Results

This section presents major results collected from the analysis of the Gabbleduck system, OpenStack, and the work conducted as part of this master's thesis project. Section 5.1.1 analyzes the time required to install Gabbleduck, and OpenStack. Section 5.1.2 presents some provisioning specifications and analyzes the amount of time required to provision and connect to different sets of VMs, defined by these specifications. Section 5.1.3 analyzes a minor internal Cisco project that makes use of the Gabbleduck system. Finally, Section 5.1.4, presents some assignments designed to measure the time taken to understand, design, and redesign some provisioning specifications.

### 5.1.1 Time to Install

A global metric to consider when evaluating how productive network operators can be while using a system, is how quickly they can install that system on a host machine. In an NFV scenario, typically the NFV management system has to be installed on all servers that can accommodate VNFs. If the installation process is time consuming, then this slows down network operators in the long run. However, the most important aspect of this, is that if it takes too long, then the operators will not even want to install the system, hence they will look for another way to achieve their goals.

It should be noted that it is difficult to give an estimate of the installation time of these systems. As this installation time depends on the participants' experience in debugging such systems and their experience of the platforms these systems are based on.

The installation process was observed by the author when six Cisco employees installed the Gabbleduck software on their work computers. These observations were not performed in a group, but rather they were performed individually as these employees were interested in using the Gabbleduck system, and also interested in using OpenStack. These employees were not assigned by their manager to carry out these tests, instead they did it out of personal interest. Before starting, the author was notified so that measuring of the time could be done.

Observing participants one by one, has the advantage that the investigator can focus on observing each individual participant. Additionally, as the participants did not communicate with each other, they could not help each other with difficulties. The exception is participant F, who was helped by participant E, since they worked on the same project, and had very similar environments. This explains why participant F had the fastest installation time out of all participants. The drawback of observing participants one by one, is that it is time consuming. Table 5.1 presents the results of measuring installation times.

Table 5.1: Installation time measurements.

| **Participant** | **Gabbleduck (m)** | **OpenStack (m)** |
|---|---|---|
| Participant A | 24 | 15 |
| Participant B | 32 | 261 |
| Participant C | 41 | – |
| Participant D | 52 | – |
| Participant E | 75 | 15 |
| Participant F | 21 | 16 |
| Average | 40.8 | 76.8 |
| Median | 36.5 | 15.5 |
| St dev | 18 | 106 |

The Gabbleduck installation procedure on Ubuntu systems proved to be difficult. On one occasion (while observing participant E), one of the build tools failed to download the correct dependencies from a web server. This caused the build tool to believe the task was finished, when it was not, and some of the components were never compiled. The time taken for this participant to debug and discover the source of the problem, contributed to the far above average installation time.

One problem was encountered by all participants. This was that, when installing Libvirt, the UNIX user of the participant had to be added to a new UNIX group, named libvirtd. To be added properly, the UNIX user has to log out and log in again. The error message from Gabbleduck was not helpful, hence the participants were all spending time on troubleshooting this issue.

Another problem was that participants occasionally forgot to execute a script that sets important environment variables, both of ConfD/NCS and of the Gabbleduck system. These are needed when the Gabbleduck system accesses compiled binary files of software libraries. The median time to install the Gabbleduck system was 36.5 minutes, with an average of ~41 minutes.

When installing OpenStack, all participants choose to use *Devstack*. Devstack is a set of installation scripts designed to simplify the process of installing and configuring OpenStack and its components. This does not correspond to a production environment, where operators would perhaps install different OpenStack components on different physical machines.

Compared to Gabbleduck, OpenStack is a more well tested project, with a more developed installation procedure (Devstack). A major issue encountered while installing Devstack, was failing to provide a proper configuration file that controls which OpenStack components to install. An example of such a configuration file is depicted in Listing 5.1. This configuration specifies some required passwords, sets up a network bridge between the host computer's NIC and the default OpenStack virtual network, enables a certain subset of Neutron's components, and activates the Heat component.

```
> cat local.conf
[[local|localrc]]
ADMIN_PASSWORD=somepassword
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
SERVICE_TOKEN=a682f596-76f3-11e3-b3b2-e716f9080d50

PUBLIC_INTERFACE=eth0
Q_USE_PROVIDERNET_FOR_PUBLIC=True
OVS_PHYSICAL_BRIDGE=br-ex
PUBLIC_BRIDGE=br-ex
OVS_BRIDGE_MAPPINGS=public:br-ex

disable_service n-net
enable_service q-svc
enable_service q-agt
enable_service q-dhcp
enable_service q-l3
enable_service q-meta

# Heat components
enable_service heat
enable_service h-api
enable_service h-api-cfn
enable_service h-api-cw
enable_service h-eng
```

Listing 5.1: Devstack configuration.

Research participant B experienced problems with Python dependencies, and resolved these within 4 hours. Participants C and D failed to install Devstack within a day, hence the observation was aborted. This was likely due to inconsistencies between the different components of Devstack, at the time of these installation attempts. The median amount of time required to install Devstack was 15.5 minutes. The average amount of time required was 76.8 minutes.

#### 5.1.1.1 Assuming No Problems Encountered

After observing research participants install the systems investigated, the evaluation turned to investigating how much time it takes to install the systems, assuming the network operators encounter no problems during the installation procedure. These tests were all performed on the same computer, which has an Intel Core i7-4600U CPU @ 2.10GHz x 4 CPUs, with 8GB

RAM. The same tests were performed (in sequence, not in parallel) ten times to increase the reliability of the estimated time for an installation. Table 5.2 shows the results of this investigation. The median time to start the Gabbleduck system is ~59 seconds, while the median time to start OpenStack using Devstack is ~603 seconds. The standard deviation is 2.38 seconds in the case of starting Gabbleduck, and 11.77 seconds in the case of starting OpenStack. This suggests that the measured installation times varied only slightly.

Table 5.2: Start time measurements.

| **Round** | **Gabbleduck (s)** | **OpenStack (s)** |
|---|---|---|
| 1 | 62.863 | 638 |
| 2 | 56.896 | 607 |
| 3 | 57.332 | 603 |
| 4 | 58.411 | 604 |
| 5 | 59.605 | 601 |
| 6 | 58.973 | 597 |
| 7 | 58.755 | 601 |
| 8 | 55.273 | 598 |
| 9 | 61.361 | 602 |
| 10 | 55.637 | 605 |
| Average | 58.511 | 605.6 |
| Median | 58.583 | 602.5 |
| St dev | 2.387 | 11.77 |

The reason that the time to install the Gabbleduck system is an order of magnitude lower than OpenStack through Devstack is likely that OpenStack is a much larger (in terms of number of components and functionality) system than the Gabbleduck system. Data gathered from RAM allocation measurements of the systems is discussed in Section 5.1.2. When starting, both systems utilize all four CPUs available on the machine, as shown in Listing B.2.

### 5.1.2 Time to Provision VMs

In the future, when implementing auto scaling of network services that include VNFs, the amount of time required to provision VMs will be a critical measurement. Because of this, the time to provision groups of VMs on one host machine, so that they all respond to ping and SSH connection attempts, was measured. When provisioning many VMs, one would typically spread them across several host machines. However, these measurements focus on the time to start VMs, rather than the time to send creation notifications to a set of host machines.

To perform these tests, a machine with a Intel® Core™ i5-4570 CPU @ 3.20GHz CPU was used. This machine was controlled remotely, thus acting as a server. The CPU has four cores, with one thread in each core (the CPU does not support Intel® Hyper-Threading Technology) [41]. The amount of RAM is 16 GB. This machine had no Internet connection during the experimentation, it was connected through an Ethernet switch to the controlling computer. The switch used was a 5 port NETGEAR 10/100 Mbps Switch FS605 v3. The MTU was 1500 both of the machine running ConfD, and of the machine used to control the aforementioned machine.

The measurements were performed by handing instance documents of the YANG domain and network models described earlier to the Gabbleduck system. Heat YAML templates were given to OpenStack's Heat component. The Gabbleduck network models created a virtual network with the CIDR *10.0.0.0/24*. The VMs were created, booted, and connected to this virtual network. The first VM in each set of VMs was assigned (by static assignment rather than DHCP) the IP address *10.0.0.5*, and the consecutively booted machines were assigned the following addresses up until *10.0.0.14* (in the case of provisioning ten VMs).

The largest set of VMs provisioned, was ten Ubuntu 12.04 Cloud images, with 2GB of virtual memory each. RAM measurements are presented later in this section, and can also be seen in Listing B.1.

The virtual OpenStack network was realized automatically when starting OpenStack, hence there was no need to provide a Heat specification document to achieve this. Figure 5.1 shows the topology created by the Gabbleduck system. A Linux bridge is connected to the host's LAN network to allow the host computer to connect to the VMs (remember that Gabbleduck does not make use of OVS at the time of writing this thesis).

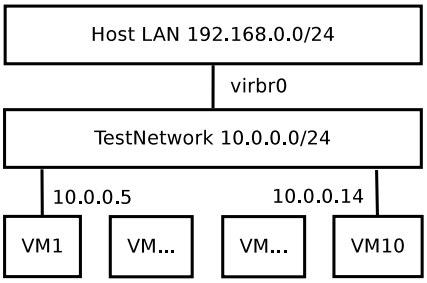


Figure 5.1: Gabbleduck evaluation network topology.

Figure 5.2 shows the slightly more complicated topology realized by OpenStack. OpenStack by default has a public network that interfaces with the host computer, then machines are connected to a private subnet that is connected through a logical router.

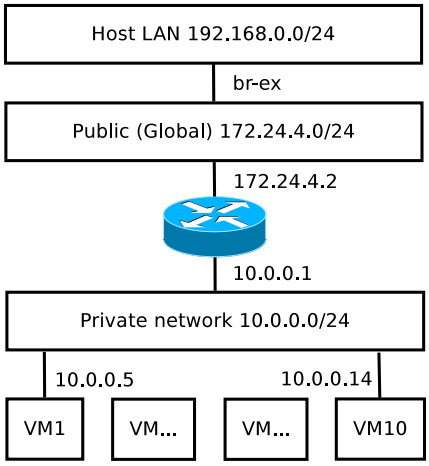


Figure 5.2: OpenStack evaluation network topology.

Listing 5.2 shows the actual route to the private OpenStack subnet *10.0.0.0/24*, implemented in the host machine's routing table. This is observed by issuing the UNIX *route* command on the host machine.

```
Destination  Gateway     Genmask         Flags  Metric  Ref  Use  Iface
10.0.0.0     172.24.4.2  255.255.255.0  U      0       0      0  br-ex
```

Listing 5.2: Route list at host computer using OpenStack.

In these experiments, the KVM hypervisor was used below Libvirt, both by OpenStack and by Gabbleduck. In the Gabbleduck system, this is specified for each domain, by the domain instance document. OpenStack determines the underlying hypervisor based on a configuration file located on the host computer. Part of the specific configuration used in these experiments is shown in Listing 5.3.

```
> cat /etc/nova/nova.conf
...
[libvirt]
vif_driver = nova.virt.libvirt.vif.LibvirtGenericVIFDriver
inject_partition = -2
live_migration_uri = qemu+ssh://stack@%s/system
use_usb_tablet = False
cpu_mode = none
virt_type = kvm
...
```

Listing 5.3: OpenStack virtual type settings.

Listing 5.4 shows the YANG instance document of the virtual network that was created by Gabbleduck during the measurements. This example is truncated after showing only one VM's IP address. The domain specifications are similar to the one shown in Listing C.2. However, note that in the experiments, each VM will need its own unique name and network interface configurations. This is because we need to be able to connect to each machine with a unique address.

```
<networks xmlns="http://tail-f.com/ns/gabbleduck/libvirt-network
    ">
  <network>
    <name>gabblemeasure</name>
    <bridge>
      <name>virbr0</name>
      <stp>on</stp>
      <delay>0</delay>
    </bridge>
    <forward>
      <mode>route</mode>
    </forward>
    <ip>
      <name>default</name>
      <address>192.168.0.0</address>
      <family>ipv4</family>
      <netmask>255.255.0.0</netmask>
      <dhcp>
        <host>
          <mac>52:54:00:fa:41:03</mac>
          <name>vm1</name>
          <ip>192.168.0.2</ip>
        </host>
      </dhcp>
    </ip>
  </network>
</networks>
```

Listing 5.4: Gabbleduck measuring network.

Listing 5.5 shows how the OpenStack VM instances were defined. Note that the Heat template (OS::Nova::Server) for specifying VMs, is on a more abstract, higher level than the Gabbleduck specification documents. For instance, as a result it is more difficult to specify the amount of RAM to allocate to VMs via Heat, than it would be using Gabbleduck. The network operator would have to first create a *flavor* specification with this information, and then reference this specification in the Nova instance specification.

```
HeatTemplateFormatVersion: '2012-12-12'

Description: Ubuntu machine

Resources:
  port1:
    Type: OS::Neutron::Port
    Properties:
      network: private
      fixed_ips:
      - subnet: private-subnet
        ip_address: 10.0.0.5

  instance1:
    Type: OS::Nova::Server
    Properties:
      name: test-01
      image: "ubuntuImage"
      flavor: m1.small
      networks:
      - network: public
        port: {Ref: port1}
      user_data: "#cloud-config
                  password: mysecret
                  chpasswd: { expire: False }
                  ssh_pwauth: True
                  #meta-data
                  network-interfaces: |
                    auto eth0
                    iface eth0 inet static
                    address 10.0.0.5
                    netmask 255.255.255.0
                    gateway 10.0.0.1"
      user_data_format: RAW
```

Listing 5.5: OpenStack measuring network.

Monitors attempting to connect to the provisioned machines using ping and SSH were created. They all constantly attempted to connect, with a 50 ms interval. To avoid excessive ping and SSH connection attempts, only as many monitors as there are VMs were created in each test.

In order to save time when developing the test environment, Gabbleduck's monitoring system was used to measure the provisioning time of VMs created by Openstack. However, this overhead should be negligible, since the memory and CPU footprint of the Gabbleduck monitor is low (around 110MB RAM). This can be seen from Listing B.1. Note that when performing OpenStack tests, Gabbleduck did not connect to Libvirt, it only spawned and

hosted monitors.

The wall clock time at which a set of VMs started booting, and the time at which the monitors succeeded in connecting to their assigned VM (with ping and SSH), was logged to a file. This file was then used to calculate the time required to provision the VM. Both the amount of time required for each VM to start responding to ping requests, and the amount of time required for the SSH server to start, was measured this way.

Each set of VMs (one to ten per set), was tested three times. Listing 4.1 (presented in Section 4.2.2) showed an example of how monitors can be defined using Gabbleduck.

Figure 5.3 shows that the time to provision and start a set of VMs, so that they all respond to ping requests, is consistently lower using the Gabbleduck system, than using the OpenStack system. When provisioning ten VMs with Gabbleduck, the median amount of time required to ping all VMs is ~1m 41s. When provisioning ten VMs with OpenStack, a median time of ~2m 35s was measured.

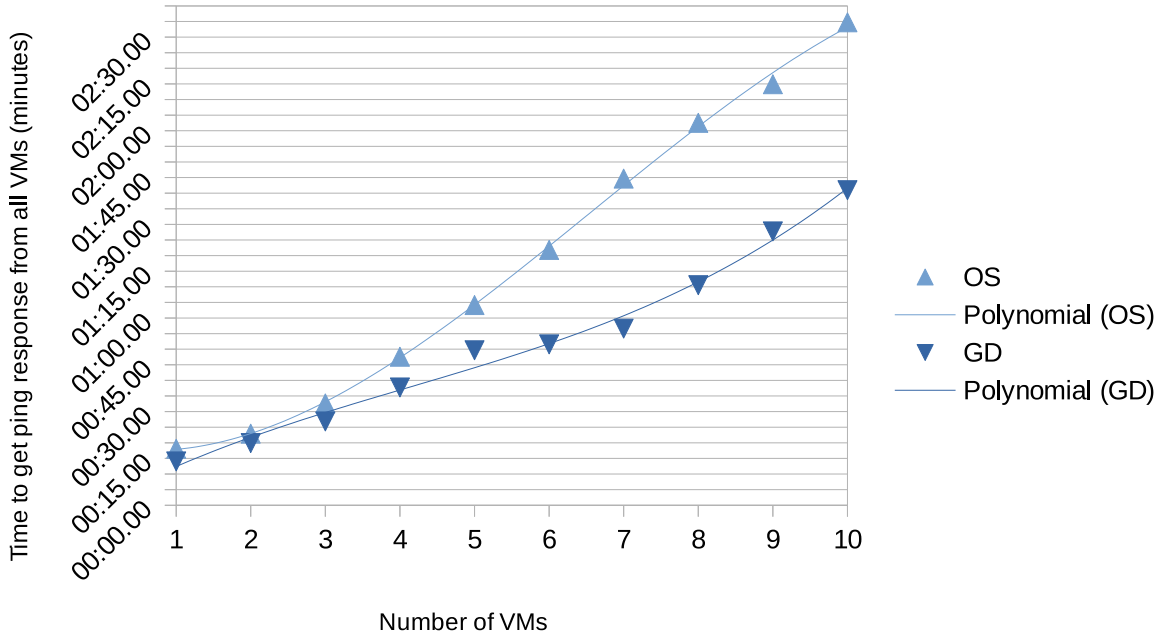


Figure 5.3: VM ping time results.

In the figure, OS is used to abbreviate OpenStack, and GD is used to abbreviate Gabbleduck. Table B.2 presents median time ping values for all sets of VMs. Table B.4 reports the coefficients and $R^2$ values for the fitted lines in Figure 5.3.

Figure 5.4 shows the amount of time required for all SSH servers in a set of VMs to start listening to connection attempts. This amount of time is higher when VMs are provisioned by OpenStack, than when VMs are provisioned by Gabbleduck. An exception to this is observed when starting only one or two VMs. In the latter case, VMs provisioned by OpenStack respond earlier to SSH connections. Provisioning was done serially, in the sense that Libvirt received instructions serially. However, once the initial creation of VMs had been performed, then VMs booted in parallel.

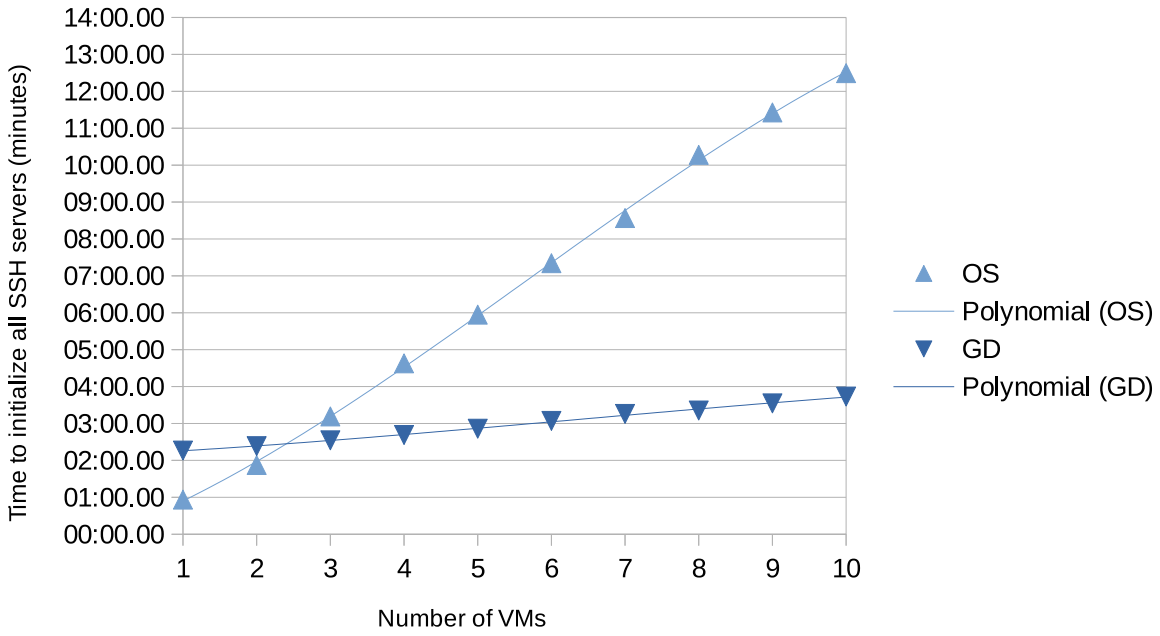


Figure 5.4: VM SSH time results.

When provisioning a set of ten VMs using OpenStack, the median amount of time required for all VMs to respond to SSH connections, was 12.5 minutes. The median amount of time time required to connect over SSH to all VMs, when provisioning these VMs with Gabbleduck, was ~3 minutes 44 seconds. Table B.3 shows the median time values to connect over SSH, for all sets of VMs. Table B.5 reports the coefficients and $R^2$ values for the fitted lines in Figure 5.4.

From the RAM traces in Listing B.1, we can see that the RAM used by OpenStack is roughly 4600MB. The RAM used by Gabbleduck was only 110 MB. Both systems utilize all four available CPUs of the host machine. This is shown in Listing B.2. CPU load tends to be high initially when provisioning a set of VMs and then decreases as the VMs boot. Gabbleduck uses 13.3% of total CPU resources and later 8.7% of total CPU resources. OpenStack initially uses 21.7% of CPU resources, while later it uses only 2.8% of total CPU resources.

The author believes that there is a correlation between the lower CPU utilization of OpenStack (after the initially high CPU utilization) when booting, and the greater amount of time required to ping (and SSH) to all VMs. This might be controlled by some Libvirt parameter that the author has not discovered, that might have to be set through OpenStack. The CPU utilization was measured with the *mpstat* UNIX command. An average of 50 measurements, with a one second delay between measurements was calculated.

### 5.1.3 Cluster Tool

In order to collect thoughts and opinions on the Gabbleduck system, an interview with two participants was conducted. These participants are the ones labeled E and F earlier, in Section 5.1.1. Participants E and F are employees of Cisco, Inc., and they were aware that their opinions and the information they shared was collected and potentially included in this master's thesis report. They were selected for an interview, because they evaluated the Gabbleduck system in a minor company internal system. This section presents the learning outcomes of this interview. The questions asked were the following:

- Why did you choose to use Gabbleduck? What other management systems did you consider?
- What is the most important aspect of a system assisting in managing networks containing VNFs?
- In what ways did Gabbleduck help you with your task?
- In what ways did Gabbleduck system did not help?
- What features were you missing in your management system?

In the internal company project, there was a need to provision a number of VMs, and install certain software on predefined nodes. A couple of these VMs act as routers (in that respect they are defined as VNFs). However, VNFs were not used as described previously in this master's thesis. The CSR1000v was not used to provision routers. Instead, Debian VMs were used with routes set up within these VMs.

These Debian VMs were pre-installed into a qcow2 file, thus the initial configuration was specified only once for all VM instances. Because of this, there was no need for the Day0 configuration functionality of Erlvirt. The topology that was provisioned is illustrated in Figure 5.5.

The participants had previously defined scripts that implemented the desired topology using Vagrant [42] together with VirtualBox [43]. However,

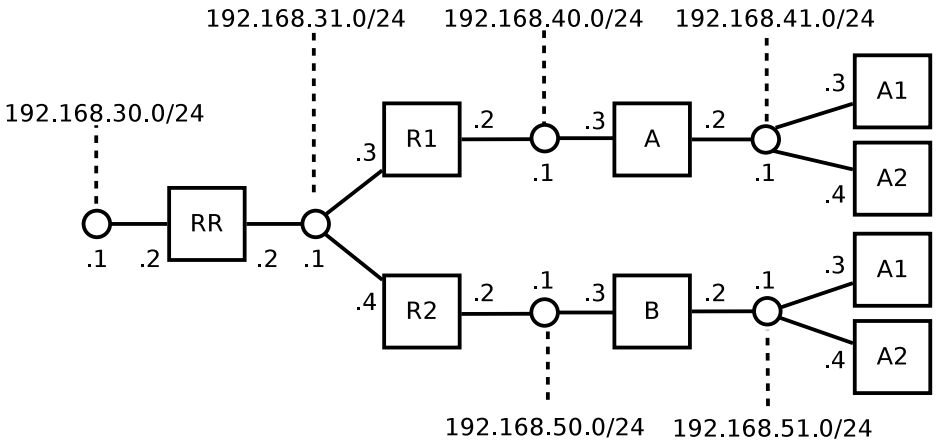


Figure 5.5: The network topology described by the participants in the interview. (Adapted from sketches drawn during the interview.)

they found Vagrant to be less stable than expected. Frequently, VMs would be created and not started, or deleted according to Vagrant – but still continue executing.

Because of this, the participants decided to evaluate Libvirt. Participant E had previously used Libvirt. Participant E had previously also used OpenStack and found it difficult to set OpenStack up properly, and to debug it when errors occurred. This previous difficulty was explained to result from the many different components of OpenStack.

Given this background, the participants were choosing between using Libvirt without any overlay, and using the Erlvirt part of the Gabbleduck system. Since the participants were comfortable using YANG models to express VMs and networks, they initially chose to use Erlvirt. After some time, when they had iterated their models, and decided upon a topology, they exported all XML documents that had been generated by Erlvirt and could then hand them directly to Libvirt.

This meant that they could circumvent Erlvirt, using only Libvirt, rather than Erlvirt and Gabbleduck, which has the advantage of not needing to execute ConfD and Erlvirt on the host machine. According to participant F, when this step was taken, they missed Gabbleduck's features of downloading and caching of machine images, and uploading these images to Libvirt's

volume pool.

The participants both said the assistance of a reliable tool is important, as such a tool gives the operator detailed control over both network configuration and settings in the hypervisor. It is important to avoid loosing control of the details, to make it easier to tune performance and perform other operations.

Participant E stated that an initial problem was that all VMs' default gateways would be the network bridge created by Gabbleduck. This means that all VMs were accessible directly from the host machine, even though they were located in subnets that should only be accessible via the routers (R1 and R2 in Figure 5.5).

Scripts connecting (over SSH) to the VMs to set up routes, and to remove the old network configuration connecting VMs directly to the host computer, had to be implemented in order to force traffic to go through the designed topology. Another issue, was that SSH connections to VMs deeper in the network topology proved to be slow.

Participant F would like to see a connection between Gabbleduck and OVS, similarly to how OpenStack is connected to OVS. With this connection, it would be possible to create more advanced virtual networks, including tunnels between networks.

### 5.1.4 Model Assignments

Perhaps the most important measure of the usefulness of an NVF management system, is the amount of time needed to design network services and the difficulty of modeling these network services. Therefore, three assignments were designed by the author and given to two groups of research participants: *group X* and *group Y*.

Since we are comparing Gabbleduck with OpenStack, each assignment consists of two parts: *part A* and *part B*. For group X, part A is an assignment related to Gabbleduck and part B is related to OpenStack. For group Y, the reverse applies. The reason to divide the participants in two groups, is to lower the learning effects of which system is used first. Since the sub parts of

the assignments are very similar, the first part would likely take much longer time than the second one, had we not have divided participants into these two groups we would have ended up with biased results.

In the assignments, the participants were asked to complete unfinished models (or modify finished models), the participants were also asked to measure the time they needed to complete these tasks. The idea is that in this way, the complexity of using the systems to design network services can be quantified.

The assignments were packed into two tarballs (depending on the participant's designated group), and given to the participants. These designated groups were randomized based on network operators that were tasked by their manager to participate in this evaluation. None of these participants were the same as those mentioned earlier, i.e. participants (A-F). Instead, these participants are identified by a number from 1-8. The reasons for this is both practical (many network operators were absent due to summer holidays) and because the investigator wanted to have fresh eyes for this investigation.

After the assignments were finished, participants were asked to provide their thoughts and comments on any problems they experienced. This makes it possible to capture opinions and thoughts that are not easy to capture with the earlier quantitative measurements. To help participants, a number of files were provided in a directory, these are largely the same as those listed in Appendix A.

Note that these assignments focused on provisioning and providing initial configurations to VNFs. Assignments providing subsequent configurations, were not tested, since OpenStack's Neutron components does not have support to configure the CSR1000v VNF.

#### 5.1.4.1 Assignment One

Assignment one, shown in Listing C.1, is designed as an introduction to the evaluated systems' templates. Participants are asked to modify instance documents that currently represent a Ubuntu VM, (shown in Listing C.2 and Listing C.3), to instance documents representing a CSR1000v VNF.

This assignment requires participants to read and understand how to provision a single VM, so that they can then modify the instance documents. The participants will want to look at the YANG model *libvirt-domain*, to learn the capabilities of a domain and also the Heat YAML template. Moreover, participants will need to look at the CSR1000v Day0 configuration file (shown in Listing C.4) to see how the CSR router will be initially configured.

Note that, with the current provisioning flow of OpenStack, it is not possible to provide a valid initial configuration to the CSR router. A virtual CD-ROM has to be mounted to the CSR, providing these configurations as a text file. This is something that Gabbleduck does. However, this issue is ignored in these assignments, as it would likely be a simple task to further develop OpenStack to support this.

#### 5.1.4.2 Assignment Two

Assignment two, shown in Listing C.5, is designed to evaluate each systems' approach to defining virtual networks. Participants are asked to define a VLAN with the CIDR *10.0.0.0/24*, and connect this VLAN to the host computer's NIC through a bridge. This assignment requires participants to read and understand the YANG model representing *libvirt-network* (the outline of this YANG model is shown in Listing A.5). They also need to understand the Neutron concepts *Net* and *Subnet*. Participants are provided with stub models that they can use in solving the assignments. These stub models are shown in Listing C.6 and Listing C.7.

#### 5.1.4.3 Assignment Three

Assignment three, shown in Listing C.8, is designed to evaluate how difficult it is to connect a VM to a network. The participants need to combine the learning outcomes of the previous two assignments in order to solve this assignment. The networks that participants were supposed to implement in assignment two, are provided (these are shown in Listing C.9 and Listing C.10) to the participants. Also, the Ubuntu VM specification from assignment one is provided. Assignment three asks the participants to connect the given VM, with the given network.

#### 5.1.4.4 Results and Analysis

This section presents the results from the model assignments performed by the volunteer participants. Figure 5.6, presents the median amount of time required by the combined groups (group X and group Y) of participants to solve their assignments. Assignments labeled A (and mentioned as GD in the legend) in this figure, are associated with Gabbleduck, while assignments B (mentioned as OS in the legend) are associated with OpenStack.

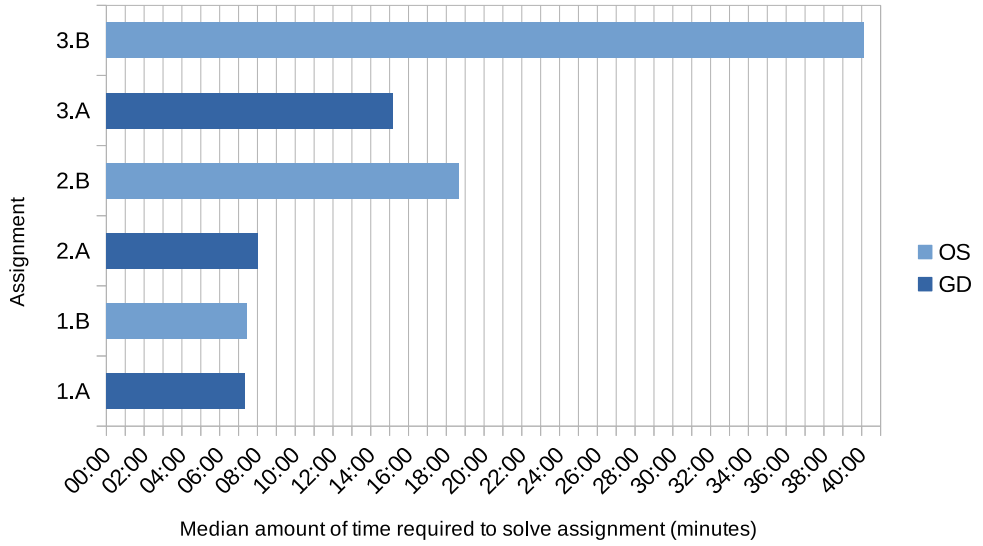


Figure 5.6: Median results of model assignments.

In addition to the figure, Tables B.6, B.7, and B.8 present this data in detail. Table B.8 also shows the standard deviation for each assignment. The standard deviation is high for most tests, for example with ~23 minues 56 seconds for assignment 2.A. This suggests that the amount of time required for participants to solve the assignment varies to a high degree.

Figure 5.7 shows the average amount of time required for participants to solve the assignments. The average amount of time required to solve 2.A was high due participants 1 and 2 needing a larger amount of time to solve this assignment, compared to the median.

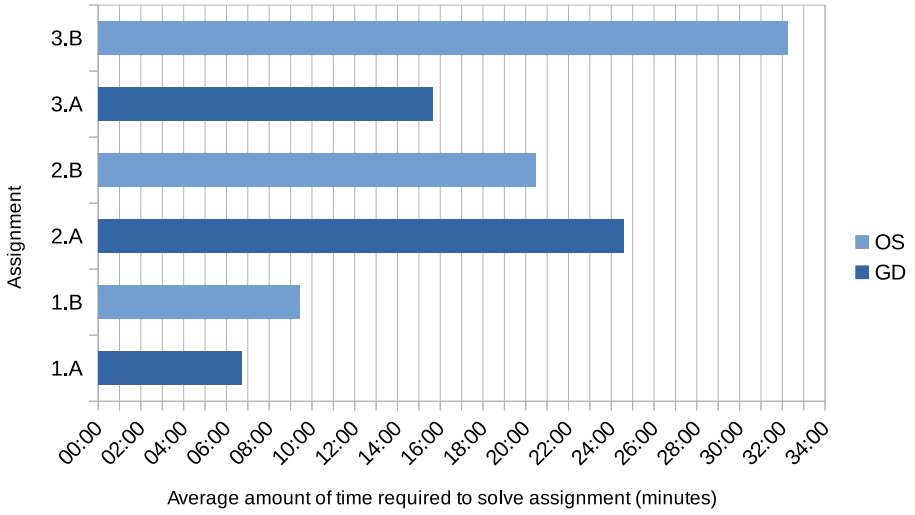


Figure 5.7: Average results of model assignments.

The median amount of time required to modify the given Ubuntu model so that it instead provisions a CSR VNF, is ~7 minutes 21 seconds using Gabbleduck, and ~7 minutes 26 seconds using OpenStack. In a small assignment such as this, consulting and understanding a rather long (over 1500 lines) YANG model such as *libvirt-domain*, can be a major overhead. The fact that the median amount of time required to solve 1.A is lower than 1.B, suggests that most participants solved the assignment without looking at the YANG model in detail. The Gabbleduck approach proved to be a little bit faster. However, the difference is a minor.

Assignment one requires the participants to allocate a certain amount of RAM to the CSR1000v VNF. In the Gabbleduck provisioning model, it is easy to find where to change this. However, participants were not told the unit of memory until they consulted the YANG model. From the YANG model, it is clear that the unit is *kilobytes*. The OpenStack approach requires participants to find out how much memory is allocated to the *m1.small* flavor (a flavor defines how much RAM and how many virtual CPUs a certain type of virtual machine is allocated).

Gabbleduck allows operators to define Day0 configurations with a URL to a text file, while OpenStack requires operators to define this information directly in the provisioning model, as a property named *user_data* [44].

Assignment two requires the participants to complete an instance document that defines how to provision a virtual network. The median amount of time required to complete this assignment, is ~8 minutes 2 seconds using Gabbleduck, and ~18 minutes 41 seconds using OpenStack. The results show that the participants could perform better in this assignment when using the Gabbleduck approach to defining networks.

The YANG model *libvirt-network* is relatively short (280 lines), which makes it fairly easy to read and understand, for someone who is acquainted with YANG. The Heat model requires the participants to find information about Neutron *Net* and *Subnet* in the OpenStack documentation (links to these are provided in the assignment description).

Participant 7 did not manage to solve assignment 2.A (which to this participant was presented as 2.B since participant 7 was a member of group Y). This means that this particular set of measurements only has seven samples, instead of eight.

Assignment three requires the participants to provide network connectivity to the VM defined in assignment one. The median amount of time required to complete this assignment, is ~15 minutes 11 seconds using Gabbleduck, and ~40 minutes 9 seconds using OpenStack. These results suggest that, having looked at the YANG models already, the participants could solve the Gabbleduck related assignment much faster than the OpenStack related assignment.

OpenStack, in this scenario, requires a Neutron Port, to connect the network and the VM. This is likely a major contributor to the much higher amount of time required for participants to solve the assignment using OpenStack, than using Gabbleduck. There was no stub Neutron Port, to guide participants that this was needed. Instead, this was a challenge that participants needed to overcome. This may have been a bad design choice when constructing the assignments or it may simply be an example of the difficulties caused by a more complex (but more flexible) tool.

## 5.2 Reliability & Validity Analysis

Since the Devstack installation procedure downloads software from a fairly frequently updated repository, it is possible that there will occasionally be bugs (and inconsistencies between the different parts of the system). Unfortunately, this proved to significantly slow the installation process down. Sometimes, these bugs are simply too difficult and time consuming to track down. In these cases, installing OpenStack through Devstack is simply not viable within a reasonable time.

Installing release versions of OpenStack is not easy either, as the installation process can for instance fail due to currently unavailable web resources. This suggests that (1) there should be a way to ensure that a given release of Devstack is consistent or (2) that there should be an alternative source for installing a consistent version of OpenStack.

This problem was experienced by participants C and D (as shown in Table 5.1). In these cases, the participants had to wait several days until the OpenStack developers detected and fixed the bugs.

The data of the experiments presented in Table 5.2 was measured ten times for each experiment, thus this data has a relatively high level of confidence. A decision to stop testing this measurement was taken when the result did not seem to change much between the rounds. The twenty separate experiments of Section 5.1.2 were done in three rounds each. A larger number of rounds (ten or higher) would give a higher level of confidence in the data. However, due to the limited time available when performing these experiments – this was not done.

## Chapter 6:

# Conclusions & Future Work

This chapter concludes this master's thesis. Section 6.1 presents conclusions drawn from the results and analysis presented in Chapter 5. Section 6.2 presents and discusses factors limiting the results. Section 6.3 presents future work in the field of this master's thesis. Finally, Section 6.4 discusses required reflections.

## 6.1 Conclusions

As shown in Chapter 4, a system capable of provisioning and configuring VNFs has been designed and implemented. This system, is capable of not only creating VMs, but also of connecting to these VMs and configuring them based on YANG models serving as contracts (or specifications) of which settings can be configured. This is essential in NFV scenarios. The prototype system can be used in real projects, as shown by Section 5.1.3.

According to measurement data presented in Section 5.1.1, installing Gabbleduck requires less time than installing OpenStack. This is useful in cases where network operators want to install the software to new machines, or reinstall systems in order to upgrade versions.

Gabbleduck proved to be more efficient than OpenStack when provisioning VMs, as shown in Section 5.1.2. Currently, this difference is not critical. To use cases such as dynamically adding routers in VPN networks, differences in minutes are negligible. However, in the future, when implementing automatic scaling of VNF systems, this will become important.

From Section 5.1, we can see that the goal of making network operators more productive, was met in the context of the designed assignments. This result however, assumes that network operators are already acquainted with the YANG based approach to provisioning services.

## 6.2 Limitations

Putting together a sufficiently large group of participants to observe was a major limitation of this project. A larger sample size would give greater statistical power. Collecting a group of participants, has a number limitations in itself.

Informed consent has to be considered, as participants need to know they are going to take part in a study and they must be willing to do so (without coercion). The overall experience of participants varied to a high degree, both in using/reading YANG models and of cloud systems in general. For instance, participants who had used both Libvirt and OpenStack before, solved the designed assignments very quickly.

There is an inherent risk that participants are biased towards one of the systems evaluated. This may occur because they have used OpenStack before, and because of that, they like it or do not like it. There may also exist bias as a result of the order of learning the different systems that are to be evaluated.

For instance, a participant might find the first system encountered difficult to comprehend. Then, when using another similar system, the participant might experience this as being less difficult, because he or she has already become familiarized with the concepts and ideas as introduced by the first system. This particular bias was considered and attempts were made to reduce it, by providing assignments in different order to different participants, as was explained in Section 5.1.4.

Another problem, that was unfortunately not considered when designing assignments, is that participants have different strategies when solving assignments. When asked to solve the assignments, some participants spend time early on to understand provided YANG models. These participants

require a high amount of time to solve the first assignment, but a lower amount of time to solve later assignments. Other participants solved assignments without first studying the YANG models, and then later on had to go back to do that. This made these participants require a higher amount of time when solving later assignments.

The assignments did not specify what precision to measure the amount of time required to solve assignments in. This lack of specification, lead some of the participants to measure their performance in precision of seconds, while others counted minutes only.

My academic adviser pointed out the problems of recruiting participants and the probability of learning effects based upon the order of learning these two different systems. However, I choose to carry out these experiments (referring to Section 5.1.4) even though I was warned about these problems, since the time and complexity for network operators to design network services including VNFs was the motivation for the project of which this master's thesis project was a part.

While conducting experiments provisioning times of sets of VMs, the computer used in these experiments had only 16GB RAM and one CPU with four cores, this was a limitation. In contrast, an example Cisco blade server [45] supports up to 6TB RAM and four CPU's with up to 15 cores each.

Instead of using release versions of OpenStack, the installation system Devstack was used. This should have no impact on the performance of OpenStack itself (KVM was already installed and executing on the host computer), but this did probably decrease the amount of time required by network operators to install OpenStack.

## 6.3 Future Work

There are a number of possible extensions and improvements available in the area of this master's thesis project. The research efforts concerning NVF are growing quickly.

OpenStack is continuously being developed. Support for NFV use cases is in the process of being planned or proposed [46]. Future work could compare the Gabbleduck approach with OpenStack's future NFV support. In addition, Ericsson is adding NFV support to OpenStack through a proprietary plug-in [5], this could also be included in evaluations.

One of the major contributions to NFV that is needed in the future is to introduce elastic scaling of VNFs. Elastic scaling requires network operators to carefully design scalable network architectures. This will mean that there is a need for appropriate tools which can provision and dynamically configure these VNFs. This elastic scaling is very important, since it could bring the same kinds of functionality to networking, as cloud systems brought to computing.

If, for instance, a major sports event is hosted in a certain city, then the network infrastructure in that city could be reinforced (scaled out) during this event so that visitors can use their smart phones without network outages caused by system overload. Then, when the sports event has finished, the network infrastructure could be automatically scaled in.

Some VNFs require several minutes to boot up and to be of any value to services and users. Automatic scaling would likely require an image cache containing preconfigured (already booted once) versions of VNFs, these could then very quickly be configured into the desired state. This area of how much to configure and when is an important topic for future research.

Automatic placement of VNFs in available hosts, would be a useful addition to the Gabbleduck. Scheduling of jobs in a distributed network of compute hosts is a well known and well understood problem [47]. However, the requirements for where to place VNFs are significantly different from the requirements of where to place computation jobs in clusters. These differences in requirements could be investigated.

Some well known cloud management systems were left out of the scope of this thesis because of time constraints. These could be taken into account in future research. Additionally, several different VM hypervisors can be investigated.

Different hypervisors have different characteristics, and can potentially be run below Libvirt with very little change to OpenStack or Gabbleduck. For instance, a certain hypervisor might be designed to be very secure, while another hypervisor might be designed to offer maximal CPU performance to managed VMs. Yet another hypervisor might be designed to offer maximal network throughput. Several papers have presented and compared hypervisors, two of these are: *Quantitative comparison of Xen and KVM* [48] and *Analysis of virtualization technologies for high performance computing environments* [49].

An interesting idea for future research, would be to replace the *Switches* layer of Figure 2.1 with FBOSS (as was introduced in Section 2.3.5). This could extend the NFV concept to be even more programmable, as it would allow network operators to treat the physical switches in data centers as UNIX servers rather than just switches. Future work could research how this would affect the NFV concept as a whole and the switch layers of data centers in particular.

Finally, it could be investigated whether a translation from YAML to YANG would be possible. This way, OpenStack could instruct ConfD or NCS to configure network devices in a way that is required in order to fully support NFV use cases. Using this approach, instead of developing OpenStack's Neutron component to support configuration of a wide variety of network devices, the existing model driven approach of NETCONF and YANG (and the existing functionality to configure network devices) could be harnessed.

## 6.4 Required Reflections

This section focuses on some of the environmental, ethical, and economic aspects of this master's thesis project. Section 6.4.1 discusses environmental implications. Section 6.4.2 discusses ethical aspects. Section 6.4.3 discusses economic aspects.

### 6.4.1 Environmental & Sustainability Aspects

This master's thesis project concerns the production and evaluation of infrastructure software that potentially other applications will depend upon. Hence, it is important that everything is developed with sustainability in mind.

Using effective cloud and network management systems, in combination with NFV technology, the world's total energy consumption could potentially be lowered by more cleverly utilizing computing power and by scheduling applications (and network functions) to execute on those machines that are currently powered on, while keeping unneeded hardware turned off [4].

There are efforts to reduce energy consumption in physical network equipment too. In his master's thesis, Eric Svensson shows how the power consumption of broadband networks can be reduced by constantly re-configuring networks to make them efficient [50]. The Institute of Electrical and Electronics Engineers (IEEE) presented work on Energy-Efficient Ethernet, that lowers energy consumption in periods of low levels of traffic [51].

### 6.4.2 Ethical Aspects

A possible scenario is that a malicious user or organization gets access to a cloud or network control systems and shuts down important services, steals secret information, or starts a huge number of VMs to cause economic loss. This can harm the environment due to high electricity consumption, while causing large financial losses, as well as damages of the reputation, of the network operator.

Getting access to many devices would unfortunately be easier for a malicious user when the system is controlled by a central manager, provided that the attacker is able to exploit this central manager. In a system where users have to physically connect to network devices in order to configure them, it is more difficult to cause damage on a large scale.

When performing interviews and observations, it is important to protect interviewees' identities in order to preserve their privacy. No names, genders, or other personal details of these interviewees was recorded. Nor are details of the participants mentioned anywhere in this thesis.

A question that springs to mind when thinking about the future of NFV systems is: Will automated systems make network operators obsolete? The author believes that this will not be the case. Potentially, low skilled network operators who manually configure network devices on behalf of high skilled network architects, might be replaced by automated systems.

However, high skilled network architects (or operators) will likely be in greater demand in the future than today. This is since it is difficult to design scalable network topologies. This process of designing scalable networks, is not easily transitioned to algorithms or artificial intelligence, at least not in the near future.

### 6.4.3 Economic Aspects

The major goal of this master's thesis project was to make network operators more productive, if successful this should lead to cost savings. Every step in the chain of consumer applications (such as databases, web back-end, and graphical user interfaces) would benefit from shorter development cycles of network infrastructure.

Both initial investment costs and operational expenses can be lowered by using an NFV model instead of traditional, specialize, on-premises network equipment [2]. Moving network functions from specialized hardware to standard commodity hardware will make these network functions cheaper to design and implement.

When costs of developing applications are sufficiently lowered (by cheaper infrastructure), then more developers can afford to realize their ideas. This is turn, gives consumers access to a wider variety of useful services and applications. In turn, as more and more services are created, the economy hopefully benefits as do the wider variety of users (since it is now feasible to create services even for smaller groups of users).

# Bibliography


[1] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," *IEEE Communications Magazine*, vol. 51, no. 11, pp. 24–31, 2013. doi: 10.1109/MCOM.2013.6658648. [Online]. Available: http://dx.doi.org/10.1109/MCOM.2013.6658648

[2] ETSI GS NFV, *NFV White Paper*, Oct. 2012. [Online]. Available: http://portal.etsi.org/nfv/nfv_white_paper.pdf

[3] J. Smith and R. Nair, "The architecture of virtual machines," *Computer*, vol. 38, no. 5, pp. 32–38, May 2005. doi: 10.1109/MC.2005.173

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010. doi: 10.1145/1721654.1721672. [Online]. Available: http://doi.acm.org/10.1145/1721654.1721672

[5] Ericsson, "Ericsson Review : OpenStack as the API framework for NFV: the benefits, and the extensions needed," Apr. 2015. [Online]. Available: http://www.ericsson.com/res/thecompany/docs/publications/ericsson_review/2015/er-openstack-api-nfv.pdf

[6] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on.* IEEE, 2009, pp. 124–131.

[7] Google, Inc., "Ganeti overview," Jun. 2015. [Online]. Available: https://code.google.com/p/ganeti/

[8] Md. Iqbal Hossain, Md. Iqbal Hossain, "Dynamic scaling of a web-based application in a Cloud Architecture," no. TRITA-ICT-EX-2014:13, 2014. [Online]. Available: http://www.diva-portal.org/smash/record.jsf?pid=diva2:699823

[9] Tail-f Systems, "Deploying virtual network functions: TOSCA & NETCONF/YANG," Feb. 2015. [Online]. Available: http://www.tail-f.com/wordpress/wp-content/uploads/2015/02/HR-Cisco-ALU-TOSCA-YANG-WP-2-17-15.pdf

[10] A. Roozbeh, "Resource monitoring in a Network Embedded Cloud: An extension to OSPF-TE," in *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*. IEEE Computer Society, 2013, pp. 139–146.

[11] ETSI GS NFV, *ETSI GS NFV-MAN 001*, Dec. 2014. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf

[12] ——, *NFV Virtualization Requirements*, Oct. 2013. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/nfv/001_099/004/01.01.01_60/gs_nfv004v010101p.pdf

[13] J. Schoenwaelder, "Overview of the 2002 IAB Network Management Workshop," RFC 3535 (Informational), Internet Engineering Task Force, May 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3535.txt

[14] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "Simple Network Management Protocol (SNMP)," RFC 1157 (Historic), Internet Engineering Task Force, May 1990. [Online]. Available: http://www.ietf.org/rfc/rfc1157.txt

[15] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)," RFC 6241 (Proposed Standard), Internet Engineering Task Force, Jun. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6241.txt

[16] M. Bjorklund, "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)," RFC 6020 (Proposed Standard), Internet Engineering Task Force, Oct. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc6020.txt

[17] S. Wallin and C. Wikström, "Automating network and service configuration using NETCONF and YANG," in *Proceedings of the 25th International Conference on Large Installation System Administration*. Berkeley, CA, USA: USENIX Association, 2011, pp. 22–22. [Online]. Available: http://dl.acm.org/citation.cfm?id=2208488.2208510

[18] K. Dooley and I. Brown, *Cisco IOS Cookbook*. O'Reilly Media, Inc., 2006. ISBN 0596527225

[19] J. Schönwälder, M. Björklund, and P. Shafer, "Network configuration management using NETCONF and YANG," *IEEE Communications Magazine*, vol. 48, no. 9, pp. 166–173, 2010. doi: 10.1109/MC.2005.173

[20] Tail-f Systems, "Tail-f ConfD User Guide : unpublished," Feb. 2015. [Online]. Available: https://developer.cisco.com/site/confD/

[21] ——, "Tail-f NCS User Guide : unpublished," Jun. 2015.

[22] T. Makita, "Virtual switching technologies and linux bridge," Jun. 2014. [Online]. Available: http://events.linuxfoundation.org/sites/events/files/slides/LinuxConJapan2014_makita_0.pdf

[23] L. Foundation, "bridge," Nov. 2009. [Online]. Available: http://www.linuxfoundation.org/collaborate/workgroups/networking/bridge

[24] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, "Extending networking into the virtualization layer," in *Eight ACM Workshop on Hot Topics in Networks (HotNets-VIII), HOTNETS '09, New York City, NY, USA, October 22-23, 2009*, 2009. [Online]. Available: http://conferences.sigcomm.org/hotnets/2009/papers/hotnets2009-final143.pdf

[25] Open vSwitch, "Open vSwitch Frequently Asked Questions," Jun. 2014. [Online]. Available: http://git.openvswitch.org/cgi-bin/gitweb.cgi?p=openvswitch;a=blob_plain;f=FAQ;hb=HEAD

[26] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. doi: 10.1145/1355734.1355746. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[27] OpenStack Foundation, "Layer 3 Networking in Neutron - via Layer 3 agent & OpenVSwitch," May 2015. [Online]. Available: http://docs.openstack.org/developer/neutron/devref/layer3.html

[28] 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, pp. 38–42, October 2012. doi: 10.5120/8738-2991. [Online]. Available: http://www.ijcaonline.org/archives/volume55/number3/8738-2991

[29] M. Bolte, M. Sievers, G. Birkenheuer, O. Niehörster, and A. Brinkmann, "Non-intrusive virtualization management using Libvirt," in *Proceedings of Design, Automation and Test in Europe (DATE)*, Dresden, Germany, 8 - 12 # mar 2010, pp. 574–579.

[30] OASIS TOSCA, "Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0," Jan. 2013. [Online]. Available: http://docs.oasis-open.org/tosca/tosca-primer/v1.0/cnd01/tosca-primer-v1.0-cnd01.pdf

[31] ——, "Topology and Orchestration Specification for Cloud Applications Version 1," Mar. 2013. [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA/v1.0/cs01/TOSCA-v1.0-cs01.pdf

[32] A. Simpkins, "Facebook Open Switching System ("FBOSS") and Wedge in the open," Mar. 2015. [Online].

Available: https://code.facebook.com/posts/843620439027582/facebook-open-switching-system-fboss-and-wedge-in-the-open

[33] A. Håkansson, "Portal of research methods and methodologies for research projects and degree projects," in *Proceedings of the International Conference on Frontiers in Education : Computer Science and Computer Engineering FECS'13*. CSREA Press U.S.A, 2013, pp. 67–73.

[34] M. Myers, *Qualitative Research in Business & Management*. SAGE Publications, 2009. ISBN 9781412921657. [Online]. Available: http://books.google.se/books?id=lpEVEfnbx9cC

[35] N. Golafshani, "Understanding reliability and validity in qualitative research," *The qualitative report*, vol. 8, no. 4, pp. 597–607, 2003. [Online]. Available: http://www.nova.edu/ssss/QR/QR8-4/golafshani.pdf

[36] M. Huo, J. Verner, L. Zhu, and M. A. Babar, "Software quality and agile methods," in *Proceedings of the 28th Annual International Computer Software and Applications Conference - Volume 01*, ser. COMPSAC '04. Washington, DC, USA: IEEE Computer Society, 2004. ISBN 0-7695-2209-2-1 pp. 520–525. [Online]. Available: http://dl.acm.org/citation.cfm?id=1025117.1025549

[37] S. Singh, "Simple random sampling," in *Advanced Sampling Theory with Applications*. Springer Netherlands, 2003, pp. 71–136. ISBN 978-94-010-3728-0. [Online]. Available: http://dx.doi.org/10.1007/978-94-007-0789-4_2

[38] N. Matloff, *The art of R programming: a tour of statistical software design*. No Starch Press, 2011. ISBN 1593273843

[39] S. Nakagawa and I. C. Cuthill, "Effect size, confidence interval and statistical significance: a practical guide for biologists," *Biological Reviews*, vol. 82, no. 4, pp. 591–605, 2007.

[40] R. Whittemore, S. K. Chase, and C. L. Mandle, "Validity in qualitative research," *Qualitative health research*, vol. 11, no. 4, pp. 522–537, 2001. [Online]. Available: http://www.sagepub.com/gray/Website%20material/Journals/whittemore.pdf

[41] Intel Corporation, "Intel® Core™ i5-4570 Processor (6M Cache, up to 3.60 GHz)," Jan. 2013. [Online]. Available: http://ark.intel.com/products/75043/Intel-Core-i5-4570-Processor-6M-Cache-up-to-3_60-GHz

[42] M. Peacock, *Creating Development Environments with Vagrant*. Packt Publishing, 2013. ISBN 1849519188, 9781849519182

[43] V. Community, "Oracle VM VirtualBox," Jun. 2015. [Online]. Available: https://www.virtualbox.orl

[44] OpenStack, "Os::nova::server," Jun. 2015. [Online]. Available: http://docs.openstack.org/hot-reference/content/OS___Nova___Server.html

[45] Cisco, Inc., "Cisco UCS B460 M4 Blade Server (with Intel® Xeon E7 v2 CPU)," Jun. 2015. [Online]. Available: http://www.cisco.com/c/dam/en/us/products/collateral/servers-unified-computing/ucs-b-series-blade-servers/B460M4_SpecSheet.pdf

[46] OpenStack Telco Working Group, "TelcoWorkingGroup mission statement and scope," Apr. 2015. [Online]. Available: https://wiki.openstack.org/wiki/TelcoWorkingGroup

[47] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ser. SOSP '13. ACM, 2013. doi: 10.1145/2517349.2522716. ISBN 978-1-4503-2388-8 pp. 69–84. [Online]. Available: http://doi.acm.org/10.1145/2517349.2522716

[48] T. Deshane, Z. Shepherd, J. Matthews, M. Ben-Yehuda, A. Shah, and B. Rao, "Quantitative comparison of Xen and KVM," *Xen Summit, Boston, MA, USA*, pp. 1–2, 2008.

[49] A. J. Younge, R. Henschel, J. T. Brown, G. Von Laszewski, J. Qiu, and G. C. Fox, "Analysis of virtualization technologies for high performance computing environments," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 2011, pp. 9–16.

[50] E. Svensson, "A First Step Toward Green Wireline Broadband : A tool for systematic measurement of Digital Subscriber Line parameters as input to dynamic power optimization algorithms," Dec. 2011. [Online]. Available: http://urn.kb.se/resolve?urn=urn%3Anbn%3Ase%3Akth%3Adiva-53600

[51] K. Christensen, P. Reviriego, B. Nordman, M. Bennett, M. Mostowfi, and J. A. Maestro, "IEEE 802.3 az: the road to energy efficient ethernet," *Communications Magazine, IEEE*, vol. 48, no. 11, pp. 50–56, 2010.

## Appendix A:

# Example Configuration Files and Models

## A.1 Heat Template of Logical Routers

```
HeatTemplateFormatVersion: '2012-12-12'

Description: Simple Network Topology

Resources:
  network1:
    Type: OS::Neutron::Net
    Properties: {name: vlan1}

  network2:
    Type: OS::Neutron::Net
    Properties: {name: vlan2}

  subnet1:
    Type: OS::Neutron::Subnet
    Properties:
      network_id: {Ref: network1}
      ip_version: 4
      cidr: 10.0.0.0/24
      allocation_pools:
      - {end: 10.0.0.150, start: 10.0.0.20}

  subnet2:
    Type: OS::Neutron::Subnet
    Properties:
      network_id: {Ref: network2}
      ip_version: 4
      cidr: 10.0.1.0/24
      allocation_pools:
      - {end: 10.0.1.150, start: 10.0.1.20}

  port1:
    Type: OS::Neutron::Port
    Properties:
      name: port1
      network: public
      fixed_ips:
      - subnet: public-subnet
        ip_address: 172.24.4.123

  port2:
    Type: OS::Neutron::Port
    Properties:
      name: port2
      network: public
      fixed_ips:
```

```
47        - subnet: public-subnet
48          ip_address: 172.24.4.127
49
50    rout1:
51      Type: 'OS::Neutron::Router'
52
53    rout2:
54      Type: 'OS::Neutron::Router'
55
56    router_interface1:
57      Type: OS::Neutron::RouterInterface
58      Properties:
59        router_id: {Ref: rout1}
60        subnet_id: {Ref: subnet1}
61
62    router_interface2:
63      Type: OS::Neutron::RouterInterface
64      Properties:
65        router_id: {Ref: rout2}
66        subnet_id: {Ref: subnet2}
67
68    router_pubinterface1:
69      Type: OS::Neutron::RouterInterface
70      Properties:
71        router_id: {Ref: rout1}
72        port_id:   {Ref: port1}
73
74    router_pubinterface2:
75      Type: OS::Neutron::RouterInterface
76      Properties:
77        router_id: {Ref: rout2}
78        port_id:   {Ref: port2}
```

Listing A.1: Heat template of logical routers.

## A.2 OVS Configuration: ifconfig Trace

```
1  br-ex     Link encap:Ethernet  HWaddr 02:c4:39:da:02:4d
2            inet addr:172.24.4.1  Bcast:0.0.0.0
3            Mask:255.255.255.0
4            inet6 addr: fe80::ec47:82ff:feaa:2d5f/64 Scope:Link
5            UP BROADCAST RUNNING  MTU:1500  Metric:1
6            RX packets:49 errors:0 dropped:0 overruns:0 frame:0
7            TX packets:45 errors:0 dropped:0 overruns:0 carrier:0
8            collisions:0 txqueuelen:0
9            RX bytes:3660 (3.6 KB)  TX bytes:3069 (3.0 KB)
10
11 br-int    Link encap:Ethernet  HWaddr 86:ba:3e:50:57:48
12           inet6 addr: fe80::201b:65ff:fe99:571f/64 Scope:Link
13           UP BROADCAST RUNNING  MTU:1500  Metric:1
14           RX packets:133 errors:0 dropped:0 overruns:0 frame:0
15           TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
16           collisions:0 txqueuelen:0
17           RX bytes:8702 (8.7 KB)  TX bytes:648 (648.0 B)
18
19 br-tun    Link encap:Ethernet  HWaddr 66:b2:9e:da:0b:40
20           inet6 addr: fe80::8c53:14ff:fea4:c230/64 Scope:Link
21           UP BROADCAST RUNNING  MTU:1500  Metric:1
22           RX packets:0 errors:0 dropped:0 overruns:0 frame:0
23           TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
24           collisions:0 txqueuelen:0
25           RX bytes:0 (0.0 B)  TX bytes:648 (648.0 B)
```

Listing A.2: OVS Configuration: ifconfig trace.

# A.3 OVS Configuration: Logical Routers

```
computer: location$ sudo ovs-vsctl show
9c023c27-1cf7-46f2-a7c8-a8d32bbae8cd
    Bridge br-tun
        fail_mode: secure
        Port br-tun
            Interface br-tun
                type: internal
        Port patch-int
            Interface patch-int
                type: patch
                options: {peer=patch-tun}
    Bridge br-ex
        Port br-ex
            Interface br-ex
                type: internal
        Port "qg-053fb28e-fb"
            Interface "qg-053fb28e-fb"
                type: internal
    Bridge br-int
        fail_mode: secure
        Port "tap6ad17169-52"
            tag: 1
            Interface "tap6ad17169-52"
                type: internal
        Port "qr-ee65896a-11"
            tag: 13
            Interface "qr-ee65896a-11"
                type: internal
        Port patch-tun
            Interface patch-tun
                type: patch
                options: {peer=patch-int}
        Port "qr-161c2ef0-3a"
            tag: 3
            Interface "qr-161c2ef0-3a"
                type: internal
        Port "tap9159df83-0c"
            tag: 13
            Interface "tap9159df83-0c"
                type: internal
        Port br-int
            Interface br-int
                type: internal
        Port "qr-6a2c1a94-a5"
            tag: 1
            Interface "qr-6a2c1a94-a5"
                type: internal
    ovs_version: "2.0.2"
```

Listing A.3: OVS configuration: Logical routers.

# A.4 Structure of libvirt-domain Model

```
module: libvirt-domain
augment /c:libvirt:
   +--rw data-sources* [name]
   |  +--rw name?      string
   |  +--rw type?      source-type
   |  +--rw source
   |     +--rw file                 c:absolute-path
   |     +--rw url                  string
   |     +--rw iso-member-url* [url]
   |     |  +--rw url?    string
   |     +--rw iso-member-text* [file-name]
   |     |  +--rw file-name?    string
   |     |  +--rw content?      string
   |     +--rw dev                  c:absolute-path
   |     +--rw dir                  c:absolute-path
   |     +--rw startup-policy?      enumeration
   +--rw domains
   |  +--rw domain* [name]
   |     +--rw name?                string
   |     +---x rebuild
   |     +---x restart-domain
   |     |  +---w input
   |     |     +---w rank?    vir-restart-type
   |     +---x get-domain-xml
   |     |  +--ro output
   |     |     +--ro domain-xml?           string
   |     |     +--ro virt-xml-validate?    string
   |     +---x snapshot
   |     |  +---w input
   |     |     +---w name?    string
   |     +---x snapshot-delete
   |     |  +---w input
   |     |     +---w name     string
   |     +--rw monitoring
   |     |  +--rw monitor-node?    string
   |     |  +--rw start-up* [after]
   |     |  |  +--rw after?          xs:duration
   |     |  |  +--rw action          action-type
   |     |  |  +--rw target          string
   |     |  |  +--rw data-source     -> /c:libvirt/data-sources/name
   |     |  |  +--rw disk            -> ../../../devices/disk/id
   |     |  +--rw failure* [after]
   |     |     +--rw after?          xs:duration
   |     |     +--rw action          action-type
   |     |     +--rw target          string
   |     |     +--rw data-source     -> /c:libvirt/data-sources/name
   |     |     +--rw disk            -> ../../../devices/disk/id
   |     +--rw wanted-state?        vir-domain-state
   |     +--rw hypervisor-type?     hypervisor-type
   |     +--rw description?         string
   |     +--rw uuid?                yang:uuid
   |     +--rw cpu
   |     |  +--rw mode?         enumeration
   |     |  +--rw model?        cpu-model
   |     |  +--rw vendor?       enumeration
   |     |  +--rw topology
   |     |  |  +--rw sockets?    uint32
   |     |  |  +--rw cores?      uint32
   |     |  |  +--rw threads?    uint32
   |     |  +--rw match?        enumeration
   |     |  +--rw feature* [name]
   |     |  |  +--rw name?       cpu-feature
   |     |  |  +--rw policy?     enumeration
   |     |  +--rw numa
```

```
65  |   |   |   +--rw cell* [id]
66  |   |   |      +--rw id?           uint32
67  |   |   |      +--rw cpus?         cpuset
68  |   |   |      +--rw memory?       uint32
69  |   |   |      +--rw mem-access?   enumeration
70  |   |   +--rw cputune
71  |   |   |  +--rw shares?           uint32
72  |   |   |  +--rw period?           uint32
73  |   |   |  +--rw quota?            int64
74  |   |   |  +--rw emulator-period?  uint32
75  |   |   |  +--rw emulator-quota?   int64
76  |   |   |  +--rw vcpupin* [vcpu]
77  |   |   |  |  +--rw vcpu?     uint16
78  |   |   |  |  +--rw cpuset    cpuset
79  |   |   |  +--rw emulatorpin
80  |   |   |     +--rw cpuset?   cpuset
81  |   |   +--rw memory?             uint64
82  |   |   +--rw current-memory?     uint64
83  |   |   +--rw memory-backing
84  |   |   |  +--rw hugepages
85  |   |   |  |  +--rw page* [id]
86  |   |   |  |     +--rw id?        string
87  |   |   |  |     +--rw size?      uint64
88  |   |   |  |     +--rw unit?      string
89  |   |   |  |     +--rw nodeset?   cpuset
90  |   |   |  +--rw nosharepages?   empty
91  |   |   |  +--rw locked?         empty
92  |   |   +--rw vcpu
93  |   |   |  +--rw placement?   enumeration
94  |   |   |  +--rw cpuset?      cpuset
95  |   |   |  +--rw count?       uint16
96  |   |   |  +--rw current?     uint16
97  |   |   +--rw os
98  |   |   |  +--rw type
99  |   |   |  |  +--rw os-type    enumeration
100 |   |   |  |  +--rw arch?      enumeration
101 |   |   |  |  +--rw machine?   union
102 |   |   |  |  +--rw init?      c:absolute-path
103 |   |   |  |  +--rw initarg* [id]
104 |   |   |  |     +--rw id?        string
105 |   |   |  |     +--rw argument?  string
106 |   |   |  +--rw boot* [dev]
107 |   |   |     +--rw dev?   enumeration
108 |   |   +--rw bootloader?        c:absolute-path
109 |   |   +--rw bootloader-args?   string
110 |   |   +--rw features
111 |   |   |  +--rw pae?          empty
112 |   |   |  +--rw apic!
113 |   |   |  |  +--rw eoi?   c:on-off
114 |   |   |  +--rw acpi?         empty
115 |   |   |  +--rw hap?          empty
116 |   |   |  +--rw hyperv!
117 |   |   |  |  +--rw relaxed
118 |   |   |  |  |  +--rw state?   c:on-off
119 |   |   |  |  +--rw vapic
120 |   |   |  |  |  +--rw state?   c:on-off
121 |   |   |  |  +--rw spinlocks
122 |   |   |  |     +--rw state?     c:on-off
123 |   |   |  |     +--rw retries?   uint32
124 |   |   |  +--rw viridian?     empty
125 |   |   |  +--rw privnet?      empty
126 |   |   |  +--rw pvspinlock!
127 |   |   |     +--rw state?   c:on-off
128 |   |   +--rw devices
129 |   |      +--rw emulator?    string
130 |   |      +--rw disk* [id]
131 |   |      |  +--rw id?         string
132 |   |      |  +--rw device?     enumeration
133 |   |      |  +--rw rawio?      enumeration
```

```
134|     |         |  +--rw sgio?        enumeration                                  |
135|     |         |  +--rw type?        source-type                                  |
136|     |         |  +--rw source                                                    |
137|     |         |  |  +--rw file                 c:absolute-path                   |
138|     |         |  |  +--rw url                  string                            |
139|     |         |  |  +--rw iso-member-url* [url]                                  |
140|     |         |  |  |  +--rw url?    string                                     |
141|     |         |  |  +--rw iso-member-text* [file-name]                           |
142|     |         |  |  |  +--rw file-name?    string                              |
143|     |         |  |  |  +--rw content?      string                              |
144|     |         |  |  +--rw dev                  c:absolute-path                   |
145|     |         |  |  +--rw dir                  c:absolute-path                   |
146|     |         |  |  +--rw startup-policy?      enumeration                       |
147|     |         |  +--rw readonly?    boolean                                      |
148|     |         |  +--rw target                                                    |
149|     |         |  |  +--rw dev           string                                 |
150|     |         |  |  +--rw bus?          enumeration                            |
151|     |         |  |  +--rw tray?         enumeration                            |
152|     |         |  |  +--rw removable?    c:on-off                               |
153|     |         |  +--rw address                                                   |
154|     |         |  |  +--rw type?          string                                |
155|     |         |  |  +--rw controller?    uint8                                 |
156|     |         |  |  +--rw bus?           uint8                                 |
157|     |         |  |  +--rw unit?          uint8                                 |
158|     |         |  +--rw driver                                                    |
159|     |         |     +--rw name?     string                                       |
160|     |         |     +--rw type?     disk-type                                    |
161|     |         |     +--rw cache?    string                                       |
162|     |         +--rw interface* [id]                                              |
163|     |         |  +--rw id?             string                                  |
164|     |         |  +--rw type?           enumeration                             |
165|     |         |  +--rw source                                                    |
166|     |         |  |  +--rw bridge?     c:device-name                            |
167|     |         |  |  +--rw network?    -> /c:libvirt/n:networks/network/name    |
168|     |         |  |  +--rw dev         c:device-name                            |
169|     |         |  |  +--rw mode?       enumeration                              |
170|     |         |  |  +--rw socket                                                |
171|     |         |  |     +--rw type     enumeration                              |
172|     |         |  |     +--rw path     c:absolute-path                          |
173|     |         |  |     +--rw mode     enumeration                              |
174|     |         |  +--rw address                                                   |
175|     |         |  |  +--rw type?        string                                  |
176|     |         |  |  +--rw domain?      string                                  |
177|     |         |  |  +--rw bus?         string                                  |
178|     |         |  |  +--rw slot?        string                                  |
179|     |         |  |  +--rw function?    string                                  |
180|     |         |  +--rw model?          string                                  |
181|     |         |  +--rw target?         c:device-name                           |
182|     |         |  +--rw mac-address?    c:unicast-mac-address                   |
183|     |         |  +--rw virtualport                                               |
184|     |         |     +--rw type?          enumeration                             |
185|     |         |     +--rw parameters                                             |
186|     |         |        +--rw managerid?        uint8                             |
187|     |         |        +--rw typeid?           uint32                            |
188|     |         |        +--rw typeidversion?    uint8                             |
189|     |         |        +--rw instanceid?       yang:uuid                         |
190|     |         |        +--rw profileid?        string                            |
191|     |         |        +--rw interfaceid?      yang:uuid                         |
192|     |         +--rw graphics* [type]                                             |
193|     |         |  +--rw type?             enumeration                             |
194|     |         |  +--rw (connection-type)?                                        |
195|     |         |     +--:(port)                                                   |
196|     |         |     |  +--rw port?            uint16                           |
197|     |         |     |  +--rw autoport?        c:yes-no                         |
198|     |         |     |  +--rw websocket?       uint16                           |
199|     |         |     |  +--rw listen?          address-ip-or-name               |
200|     |         |     |  +--rw share-policy?    enumeration                      |
201|     |         |     +--:(socket)                                                 |
202|     |         |        +--rw socket?          string                             |
```

```
203 |    |        +--rw serial* [type]
204 |    |        |  +--rw type?     string
205 |    |        |  +--rw target
206 |    |        |     +--rw type?   string
207 |    |        |     +--rw port?   uint8
208 |    |        +--rw console* [type]
209 |    |           +--rw type?     string
210 |    |           +--rw target
211 |    |              +--rw type?   string
212 |    |              +--rw port?   uint8
213 |    +--ro domain-console* [name]
214 |    |  +--ro name?     string
215 |    |  +--ro buffer?   string
216 |    +--ro domain-status* [name]
217 |    |  +--ro name?         string
218 |    |  +--ro state?        vir-domain-state
219 |    |  +--ro reason?       vir-domain-reason
220 |    |  +--ro uuid?         yang:uuid
221 |    |  +--ro mem-max?      uint32
222 |    |  +--ro mem-used?     uint32
223 |    |  +--ro vcpus?        uint16
224 |    |  +--ro cpu-time?     uint64
225 |    |  +--ro snapshots* [name]
226 |    |     +--ro name?           string
227 |    |     +--ro creationTime?   yang:date-and-time
228 |    +--ro hypervisor
229 |       +--ro type?       string
230 |       +--ro version?    uint64
231 |       +--ro node
232 |          +--ro model?      string
233 |          +--ro memory?     uint64
234 |          +--ro cpus?       uint32
235 |          +--ro mhz?        uint32
236 |          +--ro nodes?      uint32
237 |          +--ro sockets?    uint32
238 |          +--ro cores?      uint32
239 |          +--ro threads?    uint32
```

Listing A.4: Structure of YANG domain model.

# A.5 Structure of libvirt-network Model

```
module: libvirt-network
augment /libvirt-common:libvirt:
  +--rw networks
     +--rw network* [name]
        +--rw name?            string
        +--rw connections?     uint32
        +--rw ipv6?            libvirt-common:yes-no
        +--rw uuid?            yang:uuid
        +--rw bridge
        |  +--rw name?      libvirt-common:device-name
        |  +--rw stp?       libvirt-common:on-off
        |  +--rw delay?     uint32
        +--rw mac
        |  +--rw address?    libvirt-common:unicast-mac-address
        +--rw forward
        |  +--rw dev?          libvirt-common:device-name
        |  +--rw mode?         enumeration
        |  +--rw nat
        |  |  +--rw address
        |  |  |  +--rw start?    inet:ipv4-address-no-zone
        |  |  |  +--rw end?      inet:ipv4-address-no-zone
        |  |  +--rw port
        |  |     +--rw start?    non-zero-port-number
        |  |     +--rw end?      non-zero-port-number
        |  +--rw managed?      libvirt-common:yes-no
        |  +--rw interface* [dev]
        |     +--rw dev?    libvirt-common:device-name
        +--rw domain
        |  +--rw name?    inet:domain-name
        +--rw ip* [name]
        |  +--rw name?       string
        |  +--rw address?    inet:ip-address-no-zone
        |  +--rw family?     enumeration
        |  +--rw (netmask-or-prefix)?
        |  |  +--:(netmask)
        |  |  |  +--rw netmask?    inet:ipv4-address-no-zone
        |  |  +--:(prefix)
        |  |     +--rw prefix?     uint8
        |  +--rw dhcp
        |     +--rw range* [start end]
        |     |  +--rw start?    inet:ip-address-no-zone
        |     |  +--rw end?      inet:ip-address-no-zone
        |     +--rw host* [mac]
        |        +--rw mac?     libvirt-common:unicast-mac-address
        |        +--rw name?    string
        |        +--rw ip?      inet:ip-address-no-zone
        +--rw virtualport
           +--rw type?          enumeration
           +--rw parameters
              +--rw managerid?        uint8
              +--rw typeid?           uint32
              +--rw typeidversion?    uint8
              +--rw instanceid?       yang:uuid
              +--rw profileid?        string
              +--rw interfaceid?      yang:uuid
```

Listing A.5: Structure of YANG network model.

## A.6 CSR Template

```
<config xmlns="http://acme.com/ns/config/1.0">
  <devices xmlns="http://acme.com/ns/ncs">
    <device tags="nocreate">
      <name>{$DEVICE_NAME}</name>
      <config tags="merge">
        <libvirt xmlns="http://acme.com/ns/gabbleduck/
                        libvirt-common">
          <networks xmlns="http://acme.com/ns/gabbleduck/
                           libvirt-network">
            <network>
              <name>wan</name>
              <bridge>
                <name>virbrwan</name>
                <stp>on</stp>
                <delay>0</delay>
              </bridge>
              <ip>
                <name>default</name>
                <address>192.168.9.1</address>
                <netmask>255.255.255.0</netmask>
              </ip>
            </network>

            <network>
              <name>monitor</name>
              <bridge>
                <name>virbrmon</name>
                <stp>on</stp>
                <delay>0</delay>
              </bridge>
              <ip>
                <name>default</name>
                <address>{$MONITOR_IF_ADDRESS}</address>
                <netmask>255.255.255.0</netmask>
              </ip>
            </network>
            <network>
              <name>lan</name>
              <bridge>
                  <name>virbrlan</name>
                  <stp>on</stp>
                  <delay>0</delay>
              </bridge>
            </network>
          </networks>
          <domains xmlns="http://acme.com/ns/gabbleduck/
                          libvirt-domain">
            <domain>
              <name>{$DOMAIN_NAME}</name>
              <wanted-state>running</wanted-state>
              <hypervisor-type>kvm</hypervisor-type>
              <description>{$DESC}</description>
              <monitoring>
                <monitor-node>inside</monitor-node>
                <start-up>
                  <after>PT5M</after>
                  <action>reboot</action>
                </start-up>
                <start-up>
                  <after>PT10M</after>
                  <action>reset</action>
                </start-up>
                <start-up>
                  <after>PT15M</after>
                  <action>reinitialize</action>
```

```
66                    <target>day1</target>
67                  </start-up>
68                  <start-up>
69                    <after>PT25M</after>
70                    <action>reinitialize</action>
71                    <target>day0</target>
72                  </start-up>
73                  <failure>
74                    <after>PT30s</after>
75                    <action>reboot</action>
76                  </failure>
77                  <failure>
78                    <after>PT10M</after>
79                    <action>reset</action>
80                  </failure>
81                  <failure>
82                    <after>PT20M</after>
83                    <action>reinitialize</action>
84                    <target>day1</target>
85                  </failure>
86                </monitoring>
87                <cpu>
88                  <mode>host-passthrough</mode>
89                </cpu>
90                <memory>2621440</memory>
91                <current-memory>2621440</current-memory>
92                <os>
93                  <type>
94                    <arch>x86_64</arch>
95                    <machine>pc</machine>
96                    <os-type>hvm</os-type>
97                  </type>
98                  <boot>
99                    <dev>hd</dev>
100                 </boot>
101               </os>
102               <features>
103                 <pae/>
104                 <apic>
105                 </apic>
106               </features>
107               <devices>
108                 <emulator>/usr/bin/qemu-system-x86_64</emulator>
109                 <disk>
110                   <id>cd</id>
111                   <device>cdrom</device>
112                   <type>dynamic-iso</type>
113                   <source>
114                     <iso-member-text>
115                       <file-name>iosxe_config.txt</file-name>
116                       <content>{$CSR_DAY0}</content>
117                     </iso-member-text>
118                   </source>
119                   <target>
120                     <dev>hdc</dev>
121                     <bus>ide</bus>
122                   </target>
123                 </disk>
124                 <disk>
125                   <id>disk</id>
126                   <type>url</type>
127                   <source>
128                     <url>{$IMAGE_URL}</url>
129                   </source>
130                   <target>
131                     <dev>hda</dev>
132                     <bus>ide</bus>
133                   </target>
134                   <address>
```

```
135                     <type>drive</type>
136                     <controller>0</controller>
137                     <bus>0</bus>
138                     <unit>0</unit>
139                   </address>
140                   <driver>
141                     <name>qemu</name>
142                     <type>qcow2</type>
143                     <cache>none</cache>
144                   </driver>
145                 </disk>
146                 <interface>
147                   <id>gig1</id>
148                   <type>network</type>
149                   <source>
150                     <network>wan</network>
151                   </source>
152                 </interface>
153                 <interface>
154                   <id>gig2</id>
155                   <type>network</type>
156                   <source>
157                     <network>monitor</network>
158                   </source>
159                 </interface>
160                 <interface>
161                   <id>gig3</id>
162                   <type>network</type>
163                   <source>
164                     <network>lan</network>
165                   </source>
166                 </interface>
167               </devices>
168             </domain>
169           </domains>
170         </libvirt>
171       </config>
172     </device>
173   </devices>
174 </config>
```

Listing A.6: CSR template.

## A.7 CSR NAT Template

```
<config xmlns="http://acme.com/ns/config/1.0">
  <devices xmlns="http://acme.com/ns/ncs">
    <device tags="nocreate">
      <name>{$DEVICE_NAME}</name>
      <config>
        <ip xmlns="urn:ios" tags="merge">
          <dhcp>
            <excluded-address>
              <low-list>
                <low>{$LAN_IP_ADDRESS}</low>
              </low-list>
            </excluded-address>
            <pool>
              <id>BRANCH_DHCP_POOL</id>
              <network>
                <network-number>{$LAN_NET_ADR}</network-number>
                <mask>{$LAN_IP_MASK}</mask>
              </network>
              <dns-server>{$DNS_SERVER}</dns-server>
              <default-router>{$LAN_IP_ADDRESS}</default-router>
              <lease>
                <days>1</days>
                <hours>1</hours>
              </lease>
            </pool>
          </dhcp>
          <nat>
            <inside>
              <source>
                <list>
                  <id>1</id>
                  <interface>
                    <GigabitEthernet>1</GigabitEthernet>
                  </interface>
                  <overload/>
                </list>
              </source>
            </inside>
          </nat>
          <route>
            <ip-route-forwarding-list>
              <prefix>0.0.0.0</prefix>
              <mask>0.0.0.0</mask>
              <forwarding-address>{$DEF_GW}</forwarding-address>
            </ip-route-forwarding-list>
          </route>
        </ip>
        <interface xmlns="urn:ios" tags="merge">
          <GigabitEthernet>
            <name>1</name>
            <ip>
```

```
52                <access-group>
53                  <direction>in</direction>
54                  <access-list>101</access-list>
55                </access-group>
56                <nat>
57                  <outside/>
58                </nat>
59              </ip>
60            </GigabitEthernet>
61            <GigabitEthernet>
62              <name>3</name>
63              <ip>
64                <nat>
65                  <inside/>
66                </nat>
67              </ip>
68            </GigabitEthernet>
69          </interface>
70          <access-list xmlns="urn:ios" tags="merge">
71            <access-list-standard-range>
72              <listnumber>1</listnumber>
73              <std-access-list-rule>
74                <rule>permit {$LAN_NET_ADR} {$LAN_INV_MASK}</rule>
75              </std-access-list-rule>
76            </access-list-standard-range>
77            <access-list-extended-range>
78              <listnumber>101</listnumber>
79              <ext-access-list-rule>
80                <rule>permit ip any host {$WAN_IP_ADDRESS}</rule>
81              </ext-access-list-rule>
82              <ext-access-list-rule>
83                <rule>deny ip {$LAN_NET_ADDRESS}
84                             {$LAN_INV_MASK} any</rule>
85              </ext-access-list-rule>
86              <ext-access-list-rule>
87                <rule>permit icmp any any echo-reply</rule>
88              </ext-access-list-rule>
89              <ext-access-list-rule>
90                <rule>permit icmp any any time-exceeded</rule>
91              </ext-access-list-rule>
92              <ext-access-list-rule>
93                <rule>permit icmp any any unreachable</rule>
94              </ext-access-list-rule>
95              <ext-access-list-rule>
96                <rule>deny ip any any</rule>
97              </ext-access-list-rule>
98            </access-list-extended-range>
99          </access-list>
100       </config>
101   </device>
102   </devices>
103 </config>
```

Listing A.7: CSR NAT template.

# A.8 CSR Day0 Configuration

```
!####_DAY0_HOSTNAME_AND_ENABLE_PASSWORD
!
hostname gorgonzola
!
enable secret admin
!
!####_DAY0_VTY_LINE
!
line con 0
exec-timeout 30 0
!
line vty 0 15
 exec-timeout 30 0
 transport input ssh
 transport output none
!
!####_DAY0_MANAGEMENT_VRF_AND_INTERFACE -
      e.g.172.16.0.21 255.255.255.0 (CSR)
!
interface GigabitEthernet 1
 description wan interface
 ip address $WAN_IP_ADDRESS $WAN_IP_MASK
 no shutdown
!
interface GigabitEthernet 2
 description Monitor interface
 ip address $MONITOR_IP_ADDRESS 255.255.255.0
 no shutdown
!
interface GigabitEthernet 3
 description Lan interface
 ip address $LAN_IP_ADDRESS $LAN_IP_MASK
 no shutdown
!
```

Listing A.8: CSR Day0 configuration.

# Appendix B:

# Measurements and Data

## B.1 VM Ping and SSH Time Stamps

Test specifies how many VMs were started. GD means Gabbleduck, OS means OpenStack. Start, Last ping, and Last SSH refer wall clock times when the last VM in a set of VMs responded to ping and SSH respectively. Ping and SSH is the delta between last ping and Start, and last SSH and Start.

| **Test** | **Start** | **Last ping** | **Last SSH** | **Ping (m)** | **SSH (m)** |
|---|---|---|---|---|---|
| GD1 1 | 12:35:15.70 | 12:35:29.68 | 12:37:31.51 | 00:00:13.98 | 00:02:15.81 |
| GD1 2 | 15:42:02.91 | 15:42:17.03 | 15:44:19.00 | 00:00:14.13 | 00:02:16.10 |
| GD1 3 | 12:41:41.88 | 12:41:55.91 | 12:43:57.85 | 00:00:14.03 | 00:02:15.97 |
| GD2 1 | 12:50:20.96 | 12:50:40.90 | 12:52:44.13 | 00:00:19.93 | 00:02:23.16 |
| GD2 2 | 12:53:36.15 | 12:53:56.40 | 12:55:59.81 | 00:00:20.25 | 00:02:23.67 |
| GD2 3 | 12:58:01.98 | 12:58:21.81 | 13:00:24.58 | 00:00:19.83 | 00:02:22.60 |
| GD3 1 | 13:02:48.37 | 13:03:14.99 | 13:05:20.35 | 00:00:26.62 | 00:02:31.97 |
| GD3 2 | 13:05:43.40 | 13:06:10.24 | 13:08:16.26 | 00:00:26.84 | 00:02:32.86 |
| GD3 3 | 13:09:18.81 | 13:09:45.78 | 13:11:51.35 | 00:00:26.98 | 00:02:32.54 |
| GD4 1 | 13:17:19.87 | 13:17:58.04 | 13:20:00.34 | 00:00:38.17 | 00:02:40.47 |
| GD4 2 | 14:07:29.67 | 14:08:07.41 | 14:10:10.93 | 00:00:37.75 | 00:02:41.27 |
| GD4 3 | 14:13:40.55 | 14:14:15.82 | 14:16:24.74 | 00:00:35.28 | 00:02:44.19 |
| GD5 1 | 14:19:38.82 | 14:20:28.46 | 14:22:30.23 | 00:00:49.64 | 00:02:51.42 |
| GD5 2 | 14:23:32.86 | 14:24:23.29 | 14:26:25.97 | 00:00:50.42 | 00:02:53.10 |
| GD5 3 | 14:33:47.90 | 14:34:32.76 | 14:36:35.90 | 00:00:44.86 | 00:02:48.00 |
| GD6 1 | 14:39:54.53 | 14:40:45.98 | 14:42:59.07 | 00:00:51.45 | 00:03:04.54 |
| GD6 2 | 14:44:44.53 | 14:45:38.58 | 14:47:48.76 | 00:00:54.04 | 00:03:04.23 |

| | | | | | |
|---|---|---|---|---|---|
| GD6 3 | 14:49:48.07 | 14:50:39.60 | 14:52:48.60 | 00:00:51.53 | 00:03:00.53 |
| GD7 1 | 15:05:46.71 | 15:06:43.31 | 15:09:02.29 | 00:00:56.60 | 00:03:15.58 |
| GD7 2 | 15:11:54.50 | 15:12:50.12 | 15:15:07.93 | 00:00:55.61 | 00:03:13.43 |
| GD7 3 | 15:16:29.20 | 15:17:28.90 | 15:19:44.74 | 00:00:59.70 | 00:03:15.54 |
| GD8 1 | 15:20:45.61 | 15:21:56.14 | 15:24:06.43 | 00:01:10.53 | 00:03:20.82 |
| GD8 2 | 15:30:32.24 | 15:31:36.84 | 15:33:53.26 | 00:01:04.60 | 00:03:21.03 |
| GD8 3 | 15:37:11.07 | 15:38:28.51 | 15:40:34.60 | 00:01:17.45 | 00:03:23.53 |
| GD9 1 | 15:47:30.77 | 15:48:58.53 | 15:51:03.47 | 00:01:27.76 | 00:03:32.71 |
| GD9 2 | 15:57:48.23 | 15:59:01.03 | 16:01:20.56 | 00:01:12.81 | 00:03:32.34 |
| GD9 3 | 16:12:24.58 | 16:13:53.83 | 16:15:57.39 | 00:01:29.26 | 00:03:32.81 |
| GD10 1 | 16:28:46.93 | 16:30:27.89 | 16:32:29.56 | 00:01:40.95 | 00:03:42.63 |
| GD10 2 | 16:43:11.59 | 16:44:51.40 | 16:46:55.38 | 00:01:39.81 | 00:03:43.79 |
| GD10 3 | 16:48:48.58 | 16:50:31.93 | 16:52:38.29 | 00:01:43.34 | 00:03:49.70 |
| OS1 1 | 18:38:20.00 | 18:38:42.73 | 18:39:18.72 | 00:00:22.73 | 00:00:58.72 |
| OS1 2 | 18:41:48.00 | 18:42:04.38 | 18:42:40.92 | 00:00:16.38 | 00:00:52.92 |
| OS1 3 | 18:54:31.00 | 18:54:49.10 | 18:55:27.35 | 00:00:18.10 | 00:00:56.35 |
| OS2 1 | 18:58:33.00 | 18:59:04.75 | 19:00:34.31 | 00:00:31.75 | 00:02:01.31 |
| OS2 2 | 19:01:58.00 | 19:02:18.74 | 19:03:47.63 | 00:00:20.74 | 00:01:49.63 |
| OS2 3 | 19:06:14.00 | 19:06:36.94 | 19:08:06.49 | 00:00:22.94 | 00:01:52.49 |
| OS3 1 | 19:20:17.00 | 19:20:41.72 | 19:22:40.85 | 00:00:24.72 | 00:02:23.85 |
| OS3 2 | 19:24:27.00 | 19:24:59.64 | 19:27:48.35 | 00:00:32.64 | 00:03:21.35 |
| OS3 3 | 19:31:45.00 | 19:32:21.85 | 19:34:56.49 | 00:00:36.85 | 00:03:11.49 |
| OS4 1 | 19:41:11.00 | 19:41:57.30 | 19:45:53.14 | 00:00:46.30 | 00:04:42.14 |
| OS4 2 | 19:47:20.00 | 19:48:07.58 | 19:51:57.76 | 00:00:47.58 | 00:04:37.76 |
| OS4 3 | 19:53:46.00 | 19:54:40.46 | 19:58:19.34 | 00:00:54.46 | 00:04:33.34 |
| OS5 1 | 20:01:29.00 | 20:02:41.15 | 20:07:36.43 | 00:01:12.15 | 00:06:07.43 |
| OS5 2 | 20:10:56.00 | 20:11:40.17 | 20:16:28.24 | 00:00:44.17 | 00:05:32.24 |
| OS5 3 | 20:22:46.00 | 20:23:50.15 | 20:28:43.06 | 00:01:04.15 | 00:05:57.06 |
| OS6 1 | 20:32:59.00 | 20:34:29.19 | 20:40:32.13 | 00:01:30.19 | 00:07:33.13 |
| OS6 2 | 20:42:27.00 | 20:43:46.79 | 20:49:47.66 | 00:01:19.79 | 00:07:20.66 |
| OS6 3 | 13:10:32.00 | 13:11:53.82 | 13:17:29.26 | 00:01:21.82 | 00:06:57.26 |

| | | | | | |
|---|---|---|---|---|---|
| OS7 1 | 13:21:00.00 | 13:22:32.91 | 13:29:33.04 | 00:01:32.91 | 00:08:33.04 |
| OS7 2 | 13:36:25.00 | 13:38:09.68 | 13:45:03.14 | 00:01:44.68 | 00:08:38.14 |
| OS7 3 | 13:47:23.00 | 13:49:10.77 | 13:55:56.98 | 00:01:47.77 | 00:08:33.98 |
| OS8 1 | 14:00:21.00 | 14:02:23.54 | 14:10:37.72 | 00:02:02.54 | 00:10:16.72 |
| OS8 2 | 14:12:19.00 | 14:14:29.81 | 14:22:51.60 | 00:02:10.81 | 00:10:32.60 |
| OS8 3 | 14:25:11.00 | 14:27:10.32 | 14:35:24.08 | 00:01:59.32 | 00:10:13.08 |
| OS9 1 | 14:39:26.00 | 14:41:31.41 | 14:50:51.22 | 00:02:05.41 | 00:11:25.22 |
| OS9 2 | 14:54:37.00 | 14:56:51.90 | 15:06:02.73 | 00:02:14.90 | 00:11:25.73 |
| OS9 3 | 15:07:34.00 | 15:09:49.25 | 15:19:02.70 | 00:02:15.25 | 00:11:28.70 |
| OS10 1 | 15:30:49.00 | 15:33:23.73 | 15:42:58.19 | 00:02:34.73 | 00:12:09.19 |
| OS10 2 | 16:00:42.00 | 16:03:19.75 | 16:13:31.42 | 00:02:37.75 | 00:12:49.42 |
| OS10 3 | 16:19:21.00 | 16:21:55.05 | 16:31:50.54 | 00:02:34.05 | 00:12:29.54 |

Table B.1: VM ping and SSH time stamps.

Table B.2: VM ping median time (minutes).

| **Amount of Machines** | **OpenStack** | **Gabbleduck** |
|---|---|---|
| 1 | 00:18.10 | 00:14.03 |
| 2 | 00:22.94 | 00:19.93 |
| 3 | 00:32.64 | 00:26.84 |
| 4 | 00:47.58 | 00:37.74 |
| 5 | 01:04.15 | 00:49.64 |
| 6 | 01:21.82 | 00:51.53 |
| 7 | 01:44.68 | 00:56.60 |
| 8 | 02:02.54 | 01:10.53 |
| 9 | 02:14.90 | 01:27.76 |
| 10 | 02:34.73 | 01:40.95 |

Table B.3: VM SSH median time (minutes).

| **Amount of Machines** | **OpenStack (m)** | **Gabbleduck (m)** |
|---|---|---|
| 1 | 00:56.35 | 02:15.97 |
| 2 | 01:52.49 | 02:23.16 |
| 3 | 03:11.49 | 02:32.54 |
| 4 | 04:37.76 | 02:41.27 |
| 5 | 05:57.06 | 02:51.42 |
| 6 | 07:20.66 | 03:04.23 |
| 7 | 08:33.98 | 03:15.54 |
| 8 | 10:16.72 | 03:21.03 |
| 9 | 11:25.73 | 03:32.71 |
| 10 | 12:29.54 | 03:43.79 |

Table B.4: VM ping median time fitted line.

| | $\mathbf{R^2}$ | **Equation** |
|---|---|---|
| OS fitted line | 0.9988610626 | $f(x) = -0.000002153x^3 + 4.23994215660883E\text{-}005x^2 - 5.34519428616658E\text{-}005x + 0.0002203858$ |
| GD fitted line | 0.9901573996 | $f(x) = 1.26921368240812E\text{-}006x^3 - 1.56727531727531E\text{-}005x^2 + 0.0001462297x + 0.000011902$ |

Table B.5: VM SSH median time fitted line.

| | $\mathbf{R^2}$ | **Equation** |
|---|---|---|
| OS fitted line | 0.9993992512 | $f(x) = -4.76717405536852E\text{-}006x^3 + 8.11772236251406E\text{-}005x^2 + 0.000533181x + 0.000015179$ |
| GD fitted line | 0.9978693691 | $f(x) = -4.46208883708874E\text{-}007x^3 + 8.40314874169023E\text{-}006x^2 + 6.93028508913935E\text{-}005x + 0.0014923746$ |

## B.2 RAM Traces

```
OpenStack with 10 VMs, and the Gabbleduck monitor executing:
> free -m
             total       used       free     shared    buffers     cached
Mem:         15960      14990        970          2        249       7039
-/+ buffers/cache:       7701       8259
Swap:        16291          0      16291

OpenStack with 10 VMs, Gabbleduck monitor shut down:
> free -m
             total       used       free     shared    buffers     cached
Mem:         15960      14880       1080          2        249       7046
-/+ buffers/cache:       7585       8375
Swap:        16291          0      16291

OpenStack with no VMs:
> free -m
             total       used       free     shared    buffers     cached
Mem:         15960      11369       4590          1        249       6677
-/+ buffers/cache:       4443      11517
Swap:        16291          0      16291

Neither OpenStack or Gabbleduck executing:
> free -m
             total       used       free     shared    buffers     cached
Mem:         15960       6734       9226          1        249       5774
-/+ buffers/cache:        711      15249
Swap:        16291          0      16291

Running Gabbleduck with 10 VMs:
> free -m
             total       used       free     shared    buffers     cached
Mem:         15960      13111       2849          1        249       9941
-/+ buffers/cache:       2920      13040
Swap:        16291          0      16291
```

Listing B.1: RAM traces.

## B.3 CPU Utilization Traces

```
(mpstat -P ALL 1 50) is used (50 times 1 second pause)

Freshly rebooted host machine:
Average: CPU %usr %nice %sys %iowait %irq %soft %steal %guest %gnice %idle
Average: all 0.33  0.00 0.23 0.15    0.00 0.00  0.00   0.00   0.00   99.30
Average:   0 0.40  0.00 0.00 0.10    0.00 0.00  0.00   0.00   0.00   99.50
Average:   1 0.40  0.00 0.30 0.00    0.00 0.00  0.00   0.00   0.00   99.30
Average:   2 0.40  0.00 0.30 0.10    0.00 0.00  0.00   0.00   0.00   99.20
Average:   3 0.20  0.00 0.20 0.30    0.00 0.00  0.00   0.00   0.00   99.30

During start phase of Devstack:
Average: CPU %usr  %nice %sys %iowait %irq %soft %steal %guest %gnice %idle
Average: all 13.09 0.00  2.42  8.00    0.00 0.06  0.00   0.00   0.00   76.42
Average:   0 12.08 0.00  2.17  5.03    0.00 0.07  0.00   0.00   0.00   80.65
Average:   1 16.09 0.00  2.83  7.37    0.00 0.09  0.00   0.00   0.00   73.61
Average:   2 13.51 0.00  2.30 11.20    0.00 0.07  0.00   0.00   0.00   72.91
Average:   3 10.70 0.00  2.38  8.39    0.00 0.05  0.00   0.00   0.00   78.48

OpenStack with no VMs:
Average: CPU %usr %nice %sys %iowait %irq %soft %steal %guest %gnice %idle
Average: all 3.10 0.00  0.90 1.46    0.00 0.01  0.00   0.00   0.00   94.53
Average:   0 3.11 0.00  1.01 0.23    0.00 0.00  0.00   0.00   0.00   95.65
Average:   1 3.34 0.00  0.88 4.01    0.00 0.01  0.00   0.00   0.00   91.76
Average:   2 3.21 0.00  0.89 1.28    0.00 0.03  0.00   0.00   0.00   94.60
Average:   3 2.75 0.00  0.82 0.30    0.00 0.00  0.00   0.00   0.00   96.13

OpenStack during start phase of 10 VMs (initially):
Average: CPU %usr  %nice  %sys %iowait %irq %soft %steal %guest %gnice %idle
Average: all 21.71 0.00   10.44 24.87   0.00 0.17  0.00   15.19  0.00   27.63
Average:   0 22.60 0.00    9.75 13.09   0.00 0.18  0.00   14.52  0.00   39.87
Average:   1 19.77 0.00   13.38 23.14   0.00 0.18  0.00   15.06  0.00   28.48
Average:   2 22.07 0.00    9.99 33.42   0.00 0.14  0.00   15.44  0.00   18.95
Average:   3 22.36 0.00    8.65 29.84   0.00 0.16  0.00   15.79  0.00   23.21

OpenStack during start phase of 10 VMs (later):
Average: CPU %usr %nice %sys %iowait %irq %soft %steal %guest %gnice %idle
Average: all 2.80 0.00  1.26 82.40   0.00 0.02  0.00   0.13   0.00   13.40
Average:   0 2.63 0.00  1.21 86.79   0.00 0.00  0.00   0.08   0.00    9.29
Average:   1 2.70 0.00  1.09 84.66   0.00 0.02  0.00   0.10   0.00   11.43
Average:   2 2.62 0.00  1.11 80.27   0.00 0.02  0.00   0.24   0.00   15.75
Average:   3 3.28 0.00  1.63 77.88   0.00 0.02  0.00   0.08   0.00   17.11

top:
25881 libvirt+ 20 0 5034512 226704 16588 S 0.3 1.4 0:08.75 qemu-system-x86
26436 libvirt+ 20 0 5041300 223660 16348 S 0.3 1.4 0:06.99 qemu-system-x86
26675 libvirt+ 20 0 5073572 219232 16376 S 0.3 1.3 0:07.03 qemu-system-x86
27723 libvirt+ 20 0 5018164 209716 16420 S 0.3 1.3 0:18.20 qemu-system-x86
27914 libvirt+ 20 0 5041300 211644 16620 S 0.3 1.3 0:16.77 qemu-system-x86

OpenStack with 10 VMs:
Average: CPU %usr %nice %sys %iowait %irq %soft %steal %guest %gnice %idle
Average: all 3.01  0.00 1.00    1.72 0.00  0.03   0.00   0.02   0.00 94.23
Average:   0 2.93  0.00 1.01    4.79 0.00  0.04   0.00   0.02   0.00 91.21
Average:   1 2.93  0.00 1.07    1.19 0.00  0.00   0.00   0.02   0.00 94.79
Average:   2 3.05  0.00 0.83    0.20 0.00  0.04   0.00   0.04   0.00 95.84
Average:   3 3.08  0.00 1.07    0.67 0.00  0.02   0.00   0.02   0.00 95.14

Gabbleduck with no VMs:
Average: CPU %usr %nice %sys %iowait %irq %soft %steal %guest %gnice %idle
Average: all 0.76  0.00 0.30    0.12 0.00  0.01   0.00   0.00   0.00 98.81
Average:   0 0.74  0.00 0.24    0.00 0.00  0.00   0.00   0.00   0.00 99.02
Average:   1 0.52  0.00 0.16    0.24 0.00  0.00   0.00   0.00   0.00 99.08
Average:   2 1.15  0.00 0.48    0.16 0.00  0.00   0.00   0.00   0.00 98.21
Average:   3 0.66  0.00 0.32    0.08 0.00  0.00   0.00   0.00   0.00 98.94

Gabbleduck during start phase of 10 VMs (initially):
```

```
Average: CPU  %usr %nice  %sys %iowait %irq %soft %steal %guest %gnice %idle
Average: all 13.28  0.00 31.87   17.76 0.00  0.34   0.00  25.71   0.00 11.04
Average:   0 14.14  0.00 30.72   15.99 0.00  0.36   0.00  18.86   0.00 19.93
Average:   1 15.75  0.00 37.21   19.01 0.00  0.56   0.00  19.41   0.00  8.05
Average:   2 12.96  0.00 27.62   28.20 0.00  0.18   0.00  24.33   0.00  6.71
Average:   3 10.24  0.00 31.96    7.84 0.00  0.24   0.00  40.20   0.00  9.52

Gabbleduck during start phase of 10 VMs (later):
Average: CPU %usr %nice  %sys %iowait %irq %soft %steal %guest %gnice %idle
Average: all 8.74  0.00 16.37   56.00 0.00  0.20   0.00  7.45   0.00 11.23
Average:   0 8.60  0.00 15.70   48.55 0.00  0.22   0.00  8.13   0.00 18.79
Average:   1 8.12  0.00 16.60   61.72 0.00  0.26   0.00  8.64   0.00  4.66
Average:   2 9.11  0.00 17.24   61.36 0.00  0.22   0.00  5.64   0.00  6.43
Average:   3 9.13  0.00 15.95   52.41 0.00  0.10   0.00  7.41   0.00 15.00

Gabbleduck with 10 VMs:
Average: CPU %usr %nice %sys %iowait %irq %soft %steal %guest %gnice %idle
Average: all 5.70  0.00 9.34   8.93 0.00  0.34   0.00 15.24   0.00 60.45
Average:   0 5.62  0.00 9.38  10.07 0.00  0.38   0.00 15.43   0.00 59.11
Average:   1 5.79  0.00 9.13   9.25 0.00  0.43   0.00 14.95   0.00 60.45
Average:   2 5.64  0.00 9.41   7.48 0.00  0.26   0.00 14.71   0.00 62.50
Average:   3 5.73  0.00 9.42   8.94 0.00  0.28   0.00 15.87   0.00 59.76
```

Listing B.2: CPU utilization traces.

## B.4 Model Assignment Results

Table B.6: Group X, test results.

| Participant | 1.A | 1.B | 2.A | 2.B | 3.A | 3.B | Total |
|---|---|---|---|---|---|---|---|
| 1 | 00:11:15 | 00:11:20 | 00:48:54 | 00:29:56 | 00:09:50 | 00:42:16 | 02:33:31 |
| 2 | 00:11:04 | 00:09:22 | 01:05:34 | 00:39:13 | 00:05:21 | 00:44:33 | 02:55:07 |
| 3 | 00:00:50 | 00:05:30 | 00:05:00 | 00:35:00 | 00:21:47 | 00:45:19 | 01:53:26 |
| 4 | 00:01:35 | 00:04:21 | 00:06:18 | 00:07:26 | 00:26:42 | 00:31:55 | 01:18:17 |

Table B.7: Group Y, test results.

| Participant | 1.A | 1.B | 2.A | 2.B | 3.A | 3.B | Total |
|---|---|---|---|---|---|---|---|
| 5 | 00:30:00 | 00:10:00 | 00:04:34 | 00:08:02 | 00:06:32 | 00:20:00 | 01:19:08 |
| 6 | 00:02:37 | 00:04:42 | 00:06:53 | 00:03:44 | 00:07:09 | 00:10:21 | 00:35:26 |
| 7 | 00:01:23 | 00:02:43 | 00:05:23 | - | 00:41:21 | 00:21:41 | 01:12:31 |
| 8 | 00:10:57 | 00:11:32 | 00:35:19 | 00:34:41 | 00:38:56 | 00:09:34 | 02:20:59 |

Table B.8: Both groups, median time and standard deviation.

| Test | Median Time | Std Dev | Average |
|---|---|---|---|
| 1.A | 00:07:21 | 00:04:42 | 00:06:43 |
| 1.B | 00:07:26 | 00:09:07 | 00:09:26 |
| 2.A | 00:08:02 | 00:25:10 | 00:24:36 |
| 2.B | 00:18:41 | 00:15:37 | 00:20:28 |
| 3.A | 00:15:11 | 00:07:45 | 00:15:40 |
| 3.B | 00:40:09 | 00:16:13 | 00:32:15 |
| Total | 01:36:49 | - | 01:49:08 |

# Appendix C:

# Model Assignments

## C.1 Assignment 1

```
~~~~~~~~~~~~
Assignment 1
~~~~~~~~~~~~

*** Informed consent: By taking part in this assessment, the data you provide
    will be analyzed and published. No names will not be mentioned anywhere
    in the resulting document. ***

==== HELP ====

* Assume that there is a webserver listening on port 4000,
    containing the following files:
    csr1000v.qcow2   - A image file that will boot a
                       Cisco CSR1000v instance.
    ubuntu-day0.txt  - A Day0 configuration for the
                       Ubuntu VM.
                       * Available in this directory
    iosxe_config.txt - A Day0 configuration for the
                       Cisco CSR1000v router.
                       * Available in this directory

* Assume that there is a virtual LAN already set up on the host computer,
  with the following CIDR: 10.0.0.0/24. This is the VLAN that the Ubuntu
  machine is added to.

* Please, allocate at least 2 GB of RAM to the CSR1000v VM.

* ubuntu-day0.txt is an initial configuration that is given to the Ubuntu VM
  (and received by cloud-init) that is already defined. The CSR1000v requires
  another initial configuration, this one is given in iosxe_config.txt.

File list: iosxe_config.txt, ubuntu-day0.txt,
           ubuntu.xml, ubuntu.yaml

==============

We would like to change the ubuntu provisioning
templates to instead provision a CSR1000v.

(a) Modify ubuntu.xml, so that it instead of specifying
    an Ubuntu VM, specifies a Cisco CSR1000v VM.
    * Please measure how much time you need to solve
      this assignment.

(b) Modify ubuntu.yaml, so that it instead of specifying
    an Ubuntu VM, specifies a Cisco CSR1000v VM.
    * Please measure how much time you need to solve
      this assignment.
```

Listing C.1: Assignment 1 description.

```
<config xmlns="http://acme.com/ns/config/1.0">
  <libvirt xmlns="http://acme.com/ns/anvil/libvirt-common">
  <domains xmlns="http://acme.com/ns/anvil/libvirt-domain">
  <domain>
    <name>ToModify</name>
    <wanted-state>running</wanted-state>
    <hypervisor-type>kvm</hypervisor-type>
    <description>
      Provisioning specification for a Ubuntu VM.
    </description>
    <cpu>
      <mode>host-passthrough</mode>
    </cpu>
    <memory>1097152</memory>
    <vcpu>
      <count>2</count>
    </vcpu>
    <os>
      <type>
        <arch>x86_64</arch>
        <machine>pc</machine>
        <os-type>hvm</os-type>
      </type>
      <boot>
        <dev>hd</dev>
      </boot>
    </os>
    <features>
      <pae/>
      <apic>
      </apic>
    </features>
    <devices>
      <emulator>/usr/bin/qemu-system-x86_64</emulator>
      <disk>
        <id>ubuntu-day0</id>
        <device>cdrom</device>
        <type>dynamic-iso</type>
        <source>
          <iso-member-url>
            <url>http://localhost/ubuntu-day0.txt</url>
          </iso-member-url>
        </source>
        <target>
          <dev>hdc</dev>
          <bus>ide</bus>
        </target>
      </disk>
      <disk>
        <id>boot-disk</id>
        <type>url</type>
        <source>
          <url>http://localhost:4000/ubuntu-12.04- \
                server-cloudimg-amd64-disk1.img</url>
```

```
55          </source>
56          <target>
57            <dev>hda</dev>
58            <bus>ide</bus>
59          </target>
60          <address>
61            <type>drive</type>
62            <controller>0</controller>
63            <bus>0</bus>
64            <unit>0</unit>
65          </address>
66          <driver>
67            <name>qemu</name>
68            <type>qcow2</type>
69            <cache>none</cache>
70          </driver>
71        </disk>
72        <interface>
73          <id>gig1</id>
74          <type>network</type>
75          <source>
76              <network>
77                 private
78              </network>
79          </source>
80          <mac-address>52:54:00:fa:41:03</mac-address>
81          <address>
82            <type>pci</type>
83            <domain>0x00</domain>
84            <bus>0x00</bus>
85            <slot>0x03</slot>
86            <function>0x0</function>
87          </address>
88        </interface>
89      </devices>
90    </domain>
91    </domains>
92    </libvirt>
93  </config>
```

Listing C.2: Assignment 1 Ubuntu VM XML.

```
HeatTemplateFormatVersion: '2012-12-12'

Description: Provisioning specification for a Ubuntu VM.

Resources:
  ubuntuimage:
    Type: OS::Glance::Image
    Properties:
      name: "ubuntuImage"
      container_format: bare
      disk_format: qcow2
      is_public: True
      location: "http://localhost:4000/ubuntu-12.04- \
                 server-cloudimg-amd64-disk1.img"
      protected: False

  instance1:
    Type: OS::Nova::Server
    Properties:
      name: ToModify
      image: {Ref: ubuntuimage}
      flavor: m1.small
      networks:
      - network: private
        port: someport
      user_data: |
        #cloud-config
        password: passw0rd
        chpasswd: { expire: False }
        ssh_pwauth: True
      user_data_format: RAW
```

Listing C.3: Assignment 1 Ubuntu VM YAML.

```
hostname roadrunner
!
enable secret admin
!
!####_DAY0_VTY_LINE
!
line con 0
exec-timeout 30 0
!
line vty 0 15
 exec-timeout 30 0
 transport input ssh
 transport output none
!
!####_DAY0_MANAGEMENT_VRF_AND_INTERFACE - e.g.10.0.0.5
    255.255.255.0 (CSR)
!
interface GigabitEthernet 1
 description wan interface
 ip address 192.168.0.100 255.255.255.0
 no shutdown
!
interface GigabitEthernet 2
 description Lan interface
 ip address 10.0.0.5 255.255.255.0
 no shutdown
!

!####_DAY0_LOCAL_USER
!
aaa new-model
!
username admin privilege 15 password admin
!
!####_DAY0_SSH
!
ip domain lookup
ip domain-name acme.com
ip tftp source-interface GigabitEthernet 1
ip ssh version 2
ip ssh source-interface GigabitEthernet 1
no ip ssh stricthostkeycheck
!
!####_DAY0_GENERATE_CRYPTO_KEY
!
crypto key generate rsa modulus 2048 general-keys
!
do write memory
```

Listing C.4: Assignment 1 CSR1000v Day0 configuration.

## C.2 Assignment 2

```
~~~~~~~~~~~~
Assignment 2
~~~~~~~~~~~~

*** Informed consent: By taking part in this assessment,
    the data you provide will be analyzed and published.
    No names will not be mentioned anywhere in the
    resulting document. ***

==== HELP ====

* "less ../help/libvirt-network.yang"
   *** This is a schema model, that defines
       the structure of network.xml ***
* http://docs.openstack.org/hot-reference/ \
  content/OS__Neutron__Net.html
* http://docs.openstack.org/hot-reference/ \
  content/OS__Neutron__Subnet.html

File list: network.xml, network.yaml

==========

In the previous assignment, a VLAN with the CIDR of
10.0.0.0/24 was assumed. Here, we would like to
implement this VLAN, and create a bridge to the
host computer's NIC.

(a) In network.xml, fill out the <ip> tag as you
    think it should be. Please measure how much
    time you need to solve this assignment.

(b) In network.yaml, fill out the Properties of
    publicsub, as you think it should be done.
    Please measure how much time you need to
    solve this assignment.
```

Listing C.5: Assignment 2 description.

```
<config xmlns="http://acme.com/ns/config/1.0">
  <libvirt xmlns="http://acme.com/ns/acme/libvirt-common">
  <networks xmlns="http://acme.com/ns/acme/libvirt-network">
  <network>
    <name>Assignment 2 network</name>
    <bridge>
      <name>virbr0</name>
      <stp>on</stp>
      <delay>0</delay>
    </bridge>
    <forward>
      <mode>route</mode>
    </forward>
    <ip>

    </ip>
  </network>
  </networks>
  </libvirt>
</config>
```

Listing C.6: Assignment 2 network XML.

```
HeatTemplateFormatVersion: '2012-12-12'

Description: Simple Network

Resources:
  mypublic:
    Type: OS::Neutron::Net
    Properties: {name: mypublic}

  publicsub:
    Type: OS::Neutron::Subnet
    Properties:
```

Listing C.7: Assignment 2 network YAML.

## C.3 Assignment 3

```
~~~~~~~~~~~~
Assignment 3
~~~~~~~~~~~~

*** Informed consent: By taking part in this assessment,
    the data you provide will be analyzed and published.
    No names will not be mentioned anywhere in the
    resulting document. ***

==== HELP ====

* "less ../help/libvirt-network.yang"
* http://docs.openstack.org/hot-reference/content/ \
  OS__Neutron__Port.html

File list: network.xml, network.yaml, ubuntu.xml
           (ubuntu.yaml from asn1 is embedded in network.yaml)

==============

In this assignment, we would like to connect the Ubuntu VM
given in the first assignment, with the network constructed
in the second assignment.

(a) In network.xml, add to the file what you think is
    needed. Please measure how much time you need to
    solve this assignment.

(b) In network.yaml, add to the file what you think
    is needed. Please measure how much time you need
    to solve this assignment.
```

Listing C.8: Assignment 3 description.

```
<config xmlns="http://acme.com/ns/config/1.0">
  <libvirt xmlns="http://acme.com/ns/acme/libvirt-common">
  <networks xmlns="http://acme.com/ns/acme/libvirt-network">
  <network>
    <name>Assignment 2 network</name>
    <bridge>
      <name>virbr0</name>
      <stp>on</stp>
      <delay>0</delay>
    </bridge>
    <forward>
      <mode>route</mode>
    </forward>
    <ip>
      <name>default</name>
      <address>10.0.0.0</address>
      <netmask>255.255.255.0</netmask>
      <dhcp>
        <range>
          <start>10.0.0.2</start>
          <end>10.0.0.254</end>
        </range>

      </dhcp>
    </ip>
  </network>
  </networks>
  </libvirt>
</config>
```

Listing C.9: Assignment 3 network XML.

```
HeatTemplateFormatVersion: '2012-12-12'

Description: Simple Network

Resources:
  ubuntuimage:
    Type: OS::Glance::Image
    Properties:
      name: "ubuntuImage"
      container_format: bare
      disk_format: qcow2
      is_public: True
      location: "http://localhost:4000/ubuntu-12.04 \
                 -server-cloudimg-amd64-disk1.img"
      protected: False

  mypublic:
    Type: OS::Neutron::Net
    Properties: {name: mypublic}

  publicsub:
    Type: OS::Neutron::Subnet
    Properties:
      network_id: {Ref: mypublic}
      ip_version: 4
      cidr: 10.0.0.0/24
      allocation_pools:
      - {start: 10.0.0.2, end: 10.0.0.254}

  instance1:
    Type: OS::Nova::Server
    Properties:
      name: ToModify
      image: {Ref: ubuntuimage}
      flavor: m1.small
      networks:
      - network: private
        port:
      user_data: |
        #cloud-config
        password: passw0rd
        chpasswd: { expire: False }
        ssh_pwauth: True
      user_data_format: RAW
```

Listing C.10: Assignment 3 network YAML.

TRITA-ICT-EX-2015:159