Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Studiengang Computer Engineering
Fachbereich 1

Design and implementation of a connectivity manager for virtual scalable network environments

Bachelorarbeit
von

Manuel Bergler

01. Dezember 2014 – 08. Februar 2015

Referent: Herr Prof. Dr. Thomas Baar
Betreuer: Herr Benjamin Reichel M. Sc.

Manuel Bergler
Urbanstr. 26
10967 Berlin

Hiermit versichere ich, dass ich die von mir vorgelegte Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen und Internet-Quellen vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Berlin, den 08. Februar 2015

(Unterschrift)

_____

Manuel Bergler

Contents

List of figures **6**

List of tables **7**

List of algorithms **8**

1 Introduction **9**
1.1 Motivation . . . 9
1.2 Network Architecture . . . 9
1.3 Objective . . . 10
1.4 Scope . . . 10
1.5 Overview . . . 10

2 Fundamentals and related work **12**
2.1 Software-Defined Networking . . . 12
2.1.1 Motivation . . . 12
2.1.2 Software-Defined Networking concept . . . 14
2.1.3 SDN architecture . . . 14
2.1.4 OpenFlow . . . 15
2.1.5 Open vSwitch . . . 20
2.2 Cloud computing infrastructures . . . 23
2.2.1 OpenStack . . . 23
2.2.2 OpenStack Compute (Nova) . . . 24
2.2.3 OpenStack Orchestration (Heat) . . . 24
2.2.4 OpenStack Neutron . . . 24
2.3 Conclusion . . . 25

3 Requirements **26**
3.1 Functional requirements . . . 26
3.2 Non-functional requirements . . . 26

4 State of the art **27**
4.1 Overview . . . 27
4.2 OpenDaylight SDN controller . . . 27
4.3 Ryu SDN controller . . . 27
4.4 OpenStack Neutron - QoS Extension . . . 27

5 Design **28**

5.1 Architecture overview . . . 28
5.2 Design of Connectivity Manager . . . 28
5.3 Design of Connectivity Manager Agent . . . 28
5.4 Conclusion . . . 28

6 Implementation **29**
6.1 Environment . . . 29
6.2 Connectivity Manager components and operations . . . 29
6.3 OpenStack Neutron configuration . . . 29
6.4 Conclusion . . . 29

7 Evaluation **30**
7.1 Feature analysis . . . 30
7.2 Nubomedia use-case . . . 30
7.3 Conclusion . . . 30

8 Conclusion **31**
8.1 Summary . . . 31
8.2 Problems encountered . . . 31
8.3 Future work . . . 31

A List of source codes **32**

B Glossar **33**

Literatur **34**

Sachverzeichnis **35**

List of Figures

2.1 "Classical" switch components 13
2.2 Software-Defined Network architecture 14
2.3 OpenFlow Network Architecture 15
2.4 OpenFlow Switch components 16
2.5 OpenFlow pipeline processing 17
2.6 Packet flow through an OpenFlow switch 18
2.7 OpenFlow QoS as a meter 19
2.8 Architecture of Open vSwitch: divided into kernelspace and userspace 21
2.9 Visualization of the interaction of the ovs-vsctl tool 22
2.10 Interaction among OpenStack services 23
2.11 OpenStack Compute service 24

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

The demands on networks have changed dramatically in the past two decades, with an ever-growing number of people and devices relying on interconnected applications and services. The underlying infrastructure has been left mostly unchanged and is approaching its limits. In order to resolve this, Software Defined Networking (SDN) is going to be extending and replacing parts of traditional networking infrastructures. SDN separates the network into control and forwarding planes and therefore allows a more efficient orchestration and automation of network services.

The use of cloud-based services, with not only competitive pricing but also high-availability and fast network access, is taking over the traditional self-hosted data centers. The ease of administration and the deployment of new Virtual Machines (VMs) on the fly make it possible to create a Topology of Computers with no effort.

Network services have different requirements, depending on the type of data and their importance. The classification of network traffic can be done through Quality of Service (QoS). A new approach has to be made to enable the use of QoS in virtualized cloud infrastructures like OpenStack, to achieve controlled traffic from the deployment of Virtual Machines on.

1.2 Network Architecture

Today's traffic patterns, the rise of cloud computing and "big data" to only name a few examples, are exceeding the capacity of classic network architectures. With scalable computing and storage the common-place tree-structured network infrastructure with Ethernet switches are not efficient and manageable enough.

The increasing complexity of problems that have to be faced in networks and the need to control network traffic through software, are only a selection of the reasons why the Open Networking Foundation (ONF) developed an approach called Software-Defined Networking (SDN).

SDN is a leading-edge approach where the network control is separated from the forwarding

functions. The centralized network intelligence allows programming the network, without a need to access the underlying infrastructure. Therefore a shift of today's networks to more flexibility, programmability and scalability is going to take place.

1.3 Objective

The primary objective of this work is the development of a network orchestrator which is able to apply Quality of Service to the network interfaces of Virtual Machines. These Virtual Machines are deployed with OpenStack Nova and connected to an OpenVSwitch, which uses OpenFlow. Another task of the Connectivity Manager is to select which OpenStack hypervisor new VMs should be running on, which takes different runtime parameters into account. The CM should be able to be applied in environments with scalable hypervisors and VMs.

1.4 Scope

The scope of this work includes a Connectivity Manager which will have a Connectivity Manager Agent running on the cloud controller within the OpenStack infrastructure, to provide access to the hypervisors of OpenStack Nova. These two components have to be implemented and integrated with the existing OpenVSwitches. As a reference for a cloud infrastructure, multimedia communications like the Nubomedia project will be used. The deployment of this cloud is then tested on different performance characteristics like network bandwidth, latency, CPU utilization and memory usage.

In virtualized cloud infrastructure like OpenStack, the placement of Virtual Machines (VMs) on a particular compute node can be decided on by comparing different run-time parameters. The network connectivity between those VMs has to be prioritized and classified into different classes, depending on the service that are running on it.

Currently there are a number of solutions for managing network connectivity between VMs. A comparison and their current limitations follows in the next section. The chosen approach is to extend the existing network control and management services with Quality of Service (QoS) capabilities. In support of the thesis the Connectivity Manager will be implemented and the differences in bandwidth usage will be shown in one use-case.

1.5 Overview

Chapter 1 begins with the motivation for this thesis and gives a brief introduction into the objectives and the scope.

Chapter 2 gives an overview of traditional network concepts and a introduction to SDN and its components. Furthermore the different services that make up OpenStack will be described.

Chapter 3 conceptualizes the state-of-the-art solutions that are currently available and evaluates their implementation and limitations.

Chapter 4 contains an analysis of requirements and an architectural overview of the Connectivity Manager. Moreover design aspects are introduced and illustrated according to their requirements.

Chapter 5 examines the implementation of the Connectivity Manager and Agent.

Chapter 6 evaluates the network performance tests on the basis of a particular use-case.

Chapter 7 summarizes the results of this work and gives an overview on possible future work.

Chapter 2

Fundamentals and related work

2.1 Software-Defined Networking

The origin of Software-Defined Networking (SDN) began already in 1995, however the first use cases were only developed in 2001 and the promotion of SDN only began with the foundation of the non-profit industry consortium Open Networking Foundation (ONF) in 2011. The ONF is dedicated to push and adapt open standards like the OpenFlow into the industry. In this following section a brief overview of the SDN architecture and concepts, including the OpenFlow protocol is given.

2.1.1 Motivation

Today's internet is part of the modern society, be it for private users, enterprises or vital infrastructure services. Networks are required to evolve in order to address the challenges that are entailed with new applications, services and a growing number of end-users.

With a more detailed view on the challenges of current networks one comes to see the following limitations:

- **Inability to scale**: With the expansion of data centers, networks must grow too. Configuring and managing these additional network devices comes at a high administrative effort. With the virtualization of data centers network traffic patterns becomes more and more dynamic and unpredictable. With multi-tenancy a further complication is introduced, because different end-users and services need different network performance and might require traffic steering. Such scaling and network management cannot be done with a manual configuration of the underlying infrastructure.
- **Complexity**: In the past decades new networking protocols have been adapted by the industry. To add or move any device, multiple existing switches, routes, firewalls must be touched in order to manage protocol-based mechanisms on a device-level. With the virtualization of servers the amount of interfaces that need network connectivity and the distribution of applications over a number of virtual machines (VMs) are another demand that the current fairly static networks cannot dynamically adapt to.

- **Inconsistent policies**: For IT to apply a network- or data center-wide policy a lot of devices and mechanisms may need to be reconfigured. Virtual Machines are created and rebuilt within no time, but if for example access or security needs to be updated, the benefits of this dynamic are subverted(?).
- **Vendor dependence**: Standards are needed to match the requirements of the markets with the capabilities of networks and enable network operators to customize the network to specific environments.

Traditionally decisions about traffic flowing through the network are made directly by each network device, because the control logic and forwarding hardware are tightly coupled.

2.1.1.1 Classical switches & routers

Packet forwarding (data plane) and routing decisions (control plane) in classical switching and routing are both within one device. In figure .. the main components that are depicted have the following functions:

1. The **forwarding path** typically handles data path operations for each packet. It generally consists of Application-Specific Integrated Circuits (ASIC), network-processors or general-purpose processors that forwards frames and packets at wire speed (line-rate). Their lookup functions can be further increased with memory resources like Content Addressable Memory (CAM) or Ternary Content Addressable Memory (TCAM) to contain the forwarding information.
2. The elements in the **control plane** are based on general-purpose processors that provide services like routing and signaling protocols, including ARP, MAC Learning and forwarding tables.

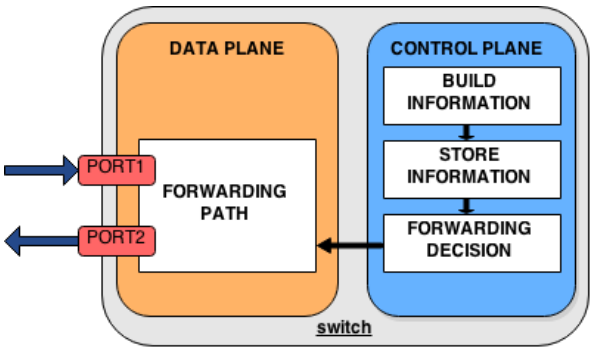

Figure 2.1: "Classical" switch components

A switch consists of multiple ports for incoming and outgoing data. Internal forwarding tables classify the packets and forward them to one or many specific ports. It does so by collecting MAC addresses and storing their corresponding port in specific tables. Layer 2 switches also support the segregation into virtual LANs (VLAN), which enables the network operator to logically isolate networks that share a single switch.

Routers forward packets on the Network layer (Layer 3) and routing-decisions are made based on IP addresses. They contain a routing table where paths to neighbour networks are stored,

so that packets can be forwarded to their destination IP address. Other features that can be configured with routers are Quality of Service (QoS), Network Address Translation (NAT) and packet filtering.

The main differences between the classical architecture and SDN will be further described in the coming sections.

2.1.2 Software-Defined Networking concept

SDN represents a new dynamic, manageable, cost-effective and adaptable architecture that is built to serve the dynamic infrastructures that are needed as a backbone for today's data centers. Opposed to the traditional approach, network control and forwarding functions are decoupled and thus can be programmed and divided into different applications and network services. The work of the Open Networking Foundation laid out the OpenFlow protocol as the base for modern SDN solutions.

2.1.3 SDN architecture

SDN separates the architecture into three distinct layers that communicate with each other through different APIs. In figure .. this separation is shown.

- **Infrastructure Layer:** here all physical and virtual devices (e.g. switches and routers) that are capable of the OpenFlow Protocol provide forwarding mechanisms on different Network Layers.
- **Control Layer:** represents the 'network intelligence' and collects global view of the network, by communicating with the switching elements through the so called South-bound API.
- **Application Layer:** consists of business applications that allow the network operator to extend the SDN controller on an abstracted level, without being tied to the actual details of the implementation of the infrastructure. This communication with the Control Layer

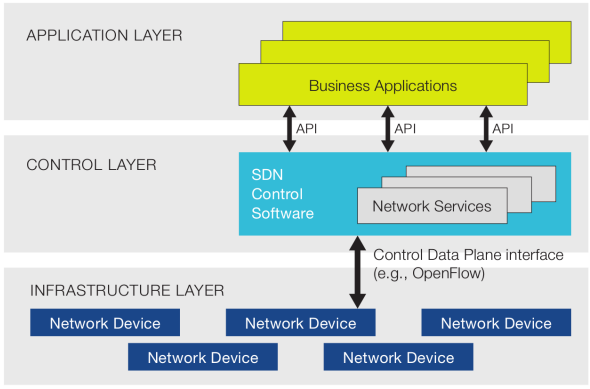

Figure 2.2: Software-Defined Network architecture

2.1.4 OpenFlow

With OpenFlow the Open Networking Foundation defined the first standard communications interface between the SDN architecture's control and forwarding layers. It enables manipulation and direct access to the forwarding plane of physical as well as virtual (hypervisor-based) network devices such as switches and routers.

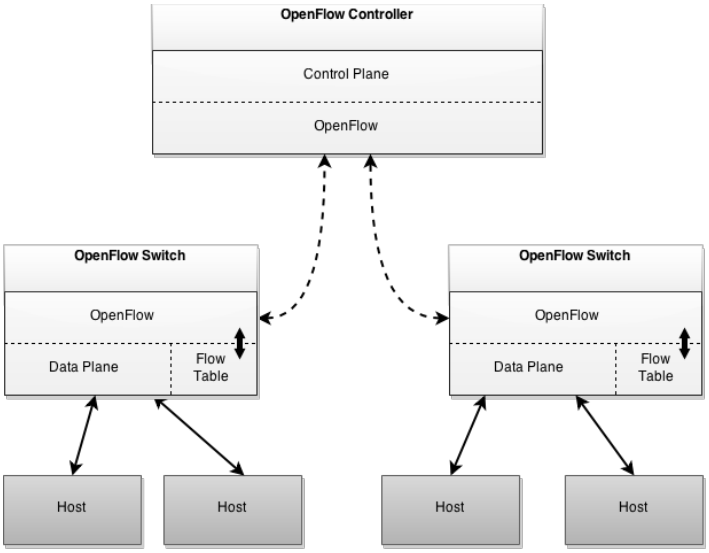

Figure 2.3: OpenFlow Network Architecture

OpenFlow first of all stands for the communications protocol that is used by SDN controllers to fetch information and configure switches. Additionally it is a switch specification that defines its minimum capabilities in order to support OpenFlow.

Most of the OpenFlow-enabled switches and controllers currently still only support the OpenFlow version 1.0 (released in December 2009). The newest version at this date is 1.4, however this explanation of OpenFlow will be focussed on version 1.3 since that is the most recent specification which is supported by OpenVSwitch. The main features added since version 1.0 are among others support for VLANs, IPv6, tunnelling and per-flow traffic meters.

Generally the switches are backwards-compatible down to version 1.0. In the following description the focus lies on the required features of all OpenFlow capable devices, however it has to be mentioned that there is also a set of optional features.

2.1.4.1 OpenFlow Controller

The OpenFlow controller is separated from the switch and has two interfaces. The northbound interface is an API to the application layer for implementing applications that control the network. The southbound interface connects with the underlying switches using the OpenFlow protocol.

2.1.4.2 OpenFlow Switch

There are two varieties of OpenFlow-compliant switches:

- **OpenFlow-only:** in these switches all packets are processed by the OpenFlow pipeline and they have no legacy features.
- **OpenFloy-hybrid:** support OpenFlow and normal Ethernet switching (including traditional L2 Ethernet switching, VLAN isolation, L3 routing, ACL and QoS). Most of the commercial switches that are available on the market today are this type.

An OpenFlow switch includes one ore multiple flow tables and a group table, which have the function of carrying out packet lookups and forwarding. Another component is the OpenFlow channel to the external controller.

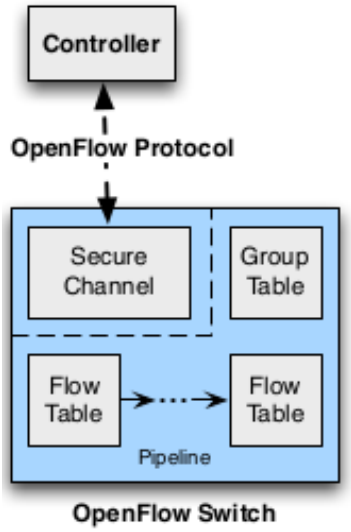

Figure 2.4: OpenFlow Switch components

Through the connection using the OpenFlow protocol, it is possible for the controller to add, update and delete flow entries in flow tables. This action can be performed either reactively or proactively. Sets of flow entries are stored in each flow table and each flow entry consists of *match fields*, *counters*, and a set of *instructions* used for matching packets. (see OF Tables section)

The matching of flow entries begins at the first flow table, however it may continue to additional flow tables, and it uses the first matching entry from each table and performs the instruction that is linked with that specific entry. For packets without any matches a table-miss flow entry can be configured. Flow entries are usually forwarded to a physical port.

The instructions can either include actions or modify pipeline processing. Packet forwarding, packet modification and group table processing are the possible actions. With pipeline processing packets can be permitted to be sent to other tables for further processing and metadata can be exchanged between tables.

Packets can also be directed to a group, which contains a set of actions for flooding and more complex forwarding semantics (e.g. multipath, fast reroute and link aggregation).

2.1.4.3 OpenFlow Ports

OpenFlow ports are the network interfaces used for passing packets between OpenFlow processing and the rest of the network. There are various types of ports that are supported by OpenFlow. This section will give a short overview about this port abstraction. Incoming OpenFlow packets enter the switch on an ingress port, are then processed by the OpenFlow pipeline and forwarded to an output port. (See OF Tables figure for processing).

There are three types of OpenFlow ports that must be supported by an OpenFlow switch:

- **Physical ports:** are hardware interfaces on a switch.
- **Logical ports:** don't directly interact with a hardware interface.
- **Reserved ports:** contain generic forwarding actions (e.g. sending to the controller, flooding or forwarding using traditional switch processing)

2.1.4.4 OpenFlow Tables

Pipeline Processing

The OpenFlow pipeline defines specifies how packets correspond with each of the flow tables.

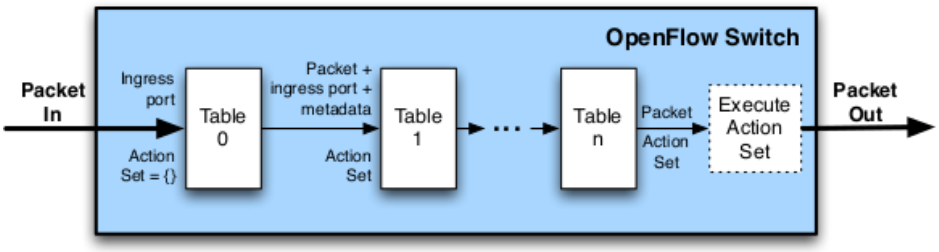

Figure 2.5: OpenFlow pipeline processing

As illustrated in the figure, each packet is matched against the flow entries starting at the first flow table, called flow table 0. The outcome of the match then decides if other of the sequentially numbered tables may be used. In the following sections the components of the Flow table, the matching procedures and different instructions will be described.

Flow Table

A flow table contains flow entries which consist of the following fields:

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

- **match fields:** ingress port, packet headers and optionally metadata
- **priority:** set the priority of the flow entry
- **counters:** is updated for matching packets
- **instructions:** to alter the action set or pipeline processing
- **timeouts:** set maximum amount of time or idle time before expiration of the flow

- **cookie:** is a opaque data value chosen by the controller

Each flow table entry is uniquely identifiable by its match fields and priority.

Packet Matching

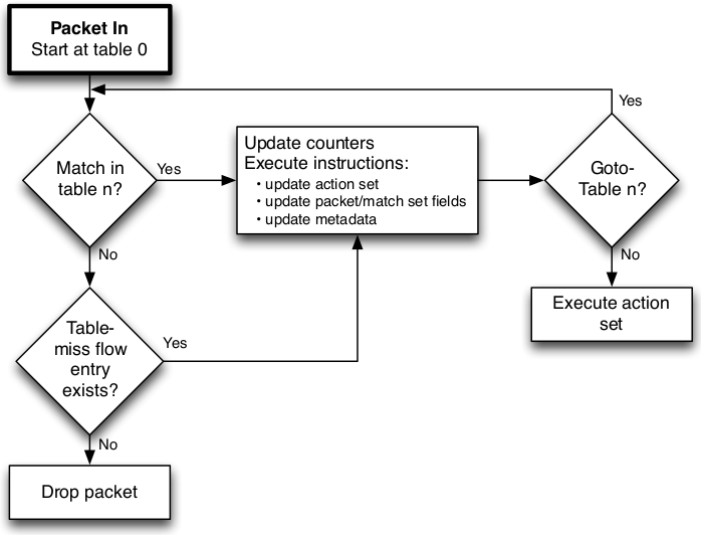

Figure 2.6: Packet flow through an OpenFlow switch

On a packet's arrival at the Flow Table, the packet match fields are extracted and used for the table lookup. They include different packet header fields. Additionally matches can be made against the ingress port and metadata fields. If the values in the packet match fields equate only the flow entry with the highest priority is selected. The associated counters are updated and the instruction set applied.

When the instruction set associated with a matching flow entry does not specify a next table, the pipeline processing stops. Only then the packet is processed with it's action set and in most cases forwarded. as shown in Figure 2.6. However, if the lookup phase does not match any of the entries, a table-miss event occurs.

Table-miss

Each flow table must support a table-miss flow entry which specifies how to process packets that are unmatched by other flow entries. The instructions associated with this entry are very alike to any other flow entries, packets are either forwarded to other controllers, dropped or it is continued with the next flow table. In case the table-miss flow entry is non-existent unmatched packets are dropped by default.

Group Tables

A group table consists of group entries and it provides a way to direct the same set of actions as part of action buckets to multiple flows. A flow entry is pointed to a group and enables additional methods of forwarding (e.g. broadcast or multicast).

Meter Tables

Meters are on a per-flow level and allow OpenFlow to implement various QoS operations,

such as rate-limiting, but it can also be combined with per-port queues to implement more complex QoS like DiffServ.

The main components of a meter entry in the meter table are:

Meter Identifier	Meter Bands	Counters

- **meter identifier:** a 32 bit unsigned integer uniquely identifying the meter
- **meter bands:** each meter band specifies the rate of the band and the action that is triggered by exceeding the limit
- **counters:** is updated when packets are processed by the meter

The rate of packets assigned to a meter are measured and controlled. Meters are directly attached to flow entries, as opposed to queues that are attached to ports. A meter is able to have one or more meter bands, each of which specifies the rate and the way packets should be handled. If the current measured meter rate reached the rate-limit, the band applies an action.

A meter band is identified by its rate and consists of the following fields:

Band Type	Rate	Counters	Type specific arguments

- **band type:** defines how the packets are processed
- **rate:** selects the meter band for the meter and defines the lowest rate at which the band can apply
- **counters:** is updated when packets are processed by the meter
- **type specific arguments:** some band have optional arguments

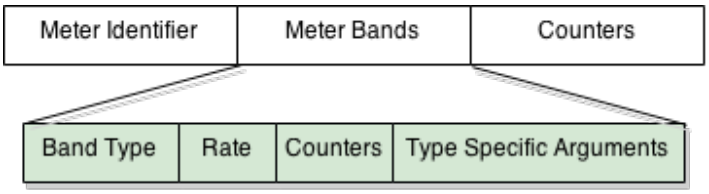

Figure 2.7: OpenFlow QoS as a meter

Instructions

Instructions are executed when a packet matches the flow entry. Their result is either a change to the packet, action set and/or pipeline processing. There are different instruction types and some of them are required for an OpenFlow-enabled switch whereas others are optional:

- **Meter *meter_id*:** direct packet to the specified meter. The packet may be discarded as the result of the metering.
- **Apply-Actions *action(s)*:** Applies the specific action(s) instantly, without any change to the Action Set.
- **Clear-Actions:** Immediately clears all the actions in the action set.

- **Write-Actions *action(s)*:** Merges the specified action(s) into the current action set.
- **Write-Metadata *metadata / mask*:** Writes the masked metadata value into the metadata field.
- **Goto-Table *next-table-id*:** Indicates the next table in the processing pipeline.

A maximum of one instruction of each type is associated with a flow entry and they are executed in the order as specified by the given list. Flow entries can also be rejected if the switch is not able to execute its instruction.

Action Set

An action set is associated with each packet, which is empty by default. The action set can be modified using a *Write-Action* or a *Clear-Action* instruction. If there is no *Goto-Table* instruction within the instruction set of a flow entry the pipeline processing is halted and the actions in the action set of the packet are executed.

Actions

The following action types are available on OpenFlow-enabled switches:

- **Output:** A packet is forwarded to a specified OpenFlow port.
- **Set-Queue:** Sets the queue id for a packet. This id helps determining which queue attached to this port is used for scheduling and forwarding the packet when the packet is forwarded to a port using the output action. This forwarding behaviour allows to enable basic QoS support.
- **Group:** Process the packet through the specified group.
- **Push-Tag/Pop-Tag:** The ability to push/pop tags such as VLAN.
- **Set-Field:** Modifies the values of header fields in a packet.
- **Change-TTL:** Set the values of IPv4 TTL, IPv6 Hop Limit or MPLS TTL in a packet.

2.1.5 Open vSwitch

2.1.5.1 Concept & Functionality

Open vSwitch (OVS) is open source software switch that is used in virtualized server environments. It is able to forward traffic traffic between Virtual Machines (VMs) and the physical network, as well as between different VMs on the same physical host. It can be controlled using OpenFlow and the OVSDB management protocol. It can run on any Linux-based virtualization platform i.e. KVM, VirtualBox, XEN, ESXi and is part of the mainline kernel as of Linux 3.3 but can run on kernel 2.6.32 and newer.

OVS supports the following features:

- 802.1Q VLAN model
- Link Aggregation Control Protocol (LACP)
- GRE and VXLAN tunneling

- fine-grained QoS control
- OpenFlow
- per VM interface traffic policing
- High-performance forwarding using a Linux kernel module

The internals of OVS are as follows:

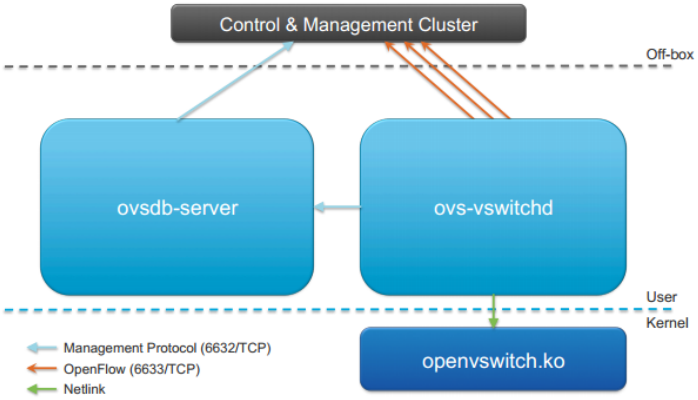

Figure 2.8: Architecture of Open vSwitch: divided into kernelspace and userspace

ovs-vswitchd, a daemon that implements the switch, along with a companion Linux kernel module for flow-based switching. ovsdb-server, a lightweight database server that ovs-vswitchd queries to obtain its configuration.

The daemon which implements the switch is *ovs-vswitchd* and it is shipped with an additional Linux kernel module for flow-based switching that it communicates with using the netlink protocol. The configuration for the switch is queried from a lightweight database server named *ovsdb-server*. Generally the decision about how a packet is processed is made in userspace, yet all following packets hit the cached entry in the kernel.

It is also possible to run it completely in userspace, but it decreases the performance drastically.

2.1.5.2 OVSDB

Each Open vSwitch daemon has a database (OVSDB) that holds it's configuration. The database is divided into multiple different tables with different purposes, with the ones related to this project outlined below:

- **Open_vSwitch**: Open vSwitch configuration
- **Port**: Port configuration
- **Interface**: A physical network device within a Port
- **QoS**: Quality of Service configuration
- **Queue**: QoS output queue

- **Controller**: OpenFlow controller configuration
- **Manager**: OVSDB management connection

2.1.5.3 OpenVSwitch Management

Multiple different configuration utilities exist for OVS, however only ovs-vsctl is explained in this section. It is used for querying and updating the configuration of the switch through interaction with the ovsdb-server.

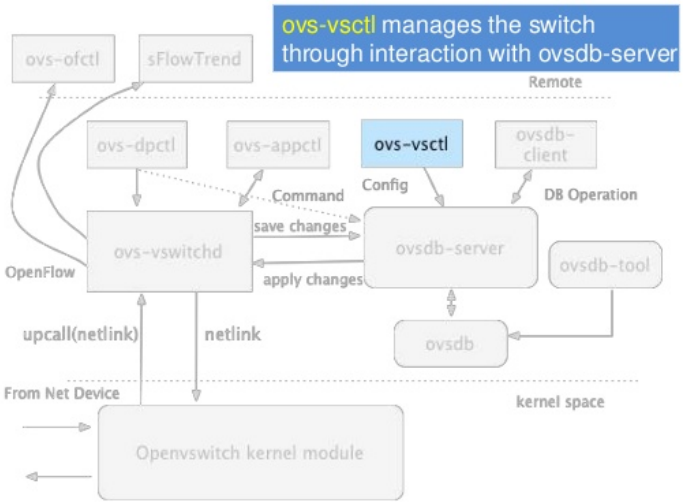

Figure 2.9: Visualization of the interaction of the ovs-vsctl tool

Even the tool configures ovs-vswitchd, it can be seen as a high-level interface for the database. The commands below are available for the basic OVS configuration that is needed to get it running for virtual network services:

- ovs-vsctl add-br %bridge%
- ovs-vsctl list-br
- ovs-vsctl add-port %bridge% %port%
- ovs-vsctl list-ports %bridge%
- ovs-vsctl get-manager %bridge%
- ovs-vsctl get-controller %bridge%
- ovs-vsctl list %table%

2.1.5.4 QoS

With Open vSwitch QoS can be configured for ports or the so-called virtual network interfaces that virtual machines get when they are connected to the internal bridge of the switch. The minimal and maximal rate-limits are defined in bytes and applied to a queue, which operates

as an egress filter. The QoS port policies make use of the 'tc' implementation that is included in the Linux kernel.

Traffic control (tc) uses 'queueing discipline' (qdisc) for configuring the network interface. When a packet is sent, it is enqueued to the qdisc for the interface and shortly after the kernel is trying to get as many packets as it can from the qdisc, so they can be forwarded to the network adaptor driver. By default the 'pfifo_fast' qdisc is set in the kernel, which is a pure First In, First out queue.

In OVS a so-called classful qdisc is used for QoS: HTB. Hierarchy Token Bucket allows guaranteeing bandwidth to classes with the possibility to define upper limits to inter-class sharing. Classes can also be prioritized.

2.1.5.5 GRE

Generic Routing Encapsulation (GRE) is used in OpenStack to tunnel the traffic between multiple nodes. It provides a private and secure path by encapsulating data packets.

2.2 Cloud computing infrastructures

2.2.1 OpenStack

The OpenStack project was founded by Rackpace Cloud and NASA, however currently more than 200 companies are contributing. With OpenStack one is able to design, deploy and maintain a private or public cloud. It is a flexible, scalable and open-source approach that combines multiple technologies into one Infrastructure-as-a-Service (IaaS). All of the inter-related services include an API that offers administrators different ways of controlling the cloud, be it through a web interface, a command-line client or a software development kit. All of the core components that come with OpenStack are implemented in Python.

Conceptual architecture

The following graph shows the interaction between different OpenStack services that are involved in launching a virtual machine.

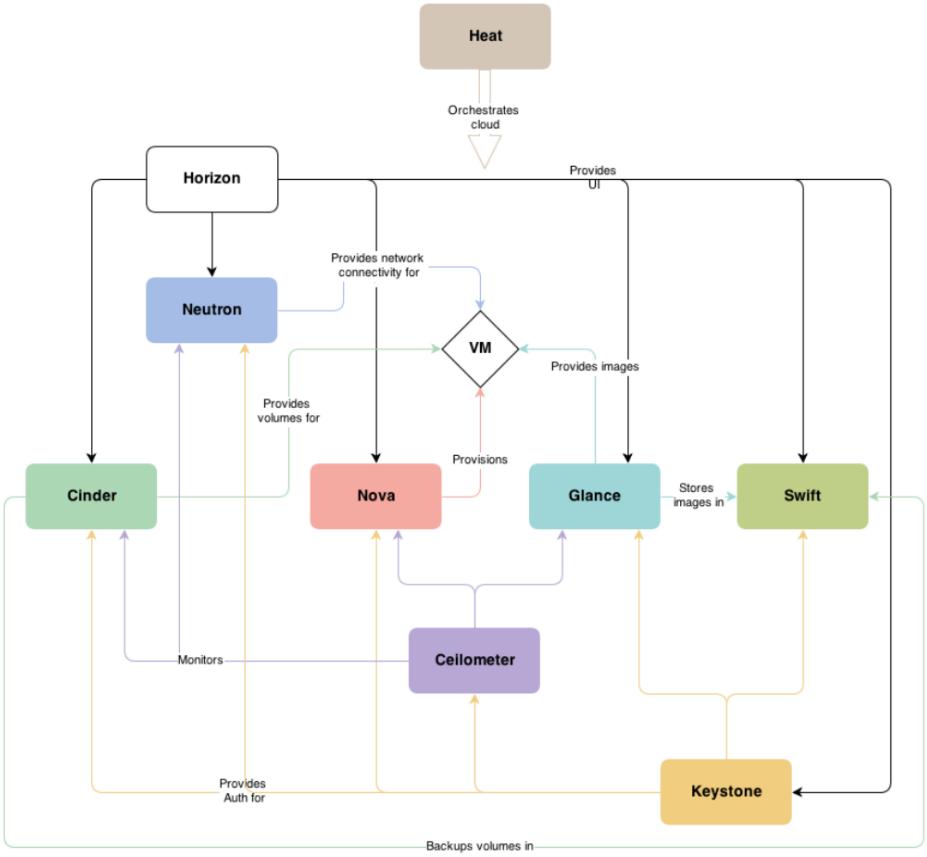

Figure 2.10: Interaction among OpenStack services

2.2.2 OpenStack Compute (Nova)

Nova is used to host and manage cloud computing systems. It supports different hypervisors and the number of physical hosts running the compute services can be scaled horizontally with no requirement of particular hardware resources.

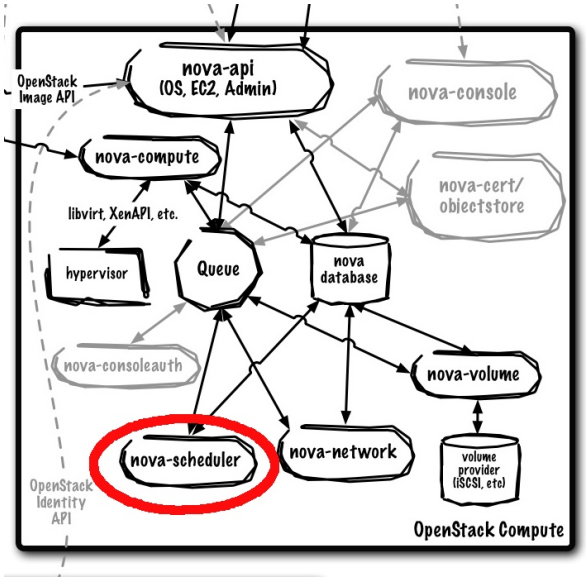

Figure 2.11: OpenStack Compute service

API ??

2.2.3 OpenStack Orchestration (Heat)

Heat provides a template-based orchestration service for creating and managing cloud resources. This means multiple OpenStack resource types (such as virtual machines, floating IP addresses, volumes, security groups and users) can be generated and also maintained with additional functionality like auto-scaling.

2.2.4 OpenStack Neutron

2.2.4.1 Networking concepts

Network types

Namespaces

Metadata

OVS Bridges

ML2 Mechanism drivers Neutron agent API

2.3 Conclusion

Chapter 3

Requirements

3.1 Functional requirements

3.2 Non-functional requirements

Chapter 4

State of the art

4.1 Overview

4.2 OpenDaylight SDN controller

4.3 Ryu SDN controller

4.4 OpenStack Neutron - QoS Extension

Chapter 5

Design

5.1 Architecture overview

5.2 Design of Connectivity Manager

5.3 Design of Connectivity Manager Agent

5.4 Conclusion

Chapter 6

Implementation

6.1 Environment

6.1.0.2 Software development approach

6.1.0.3 Project structure

6.1.0.4 Devstack

What it does and benefits

6.2 Connectivity Manager components and operations

6.2.0.5 Selection of best-matching Hypervisor

6.2.0.6 Enabling QoS for VM

6.3 OpenStack Neutron configuration

6.4 Conclusion

Chapter 7

Evaluation

7.1 Feature analysis

7.2 Nubomedia use-case

7.3 Conclusion

Chapter 8

Conclusion

8.1 Summary

8.2 Problems encountered

8.3 Future work

Appendix A

List of source codes

Appendix B

Glossar

SDN → Software-Defined Networking.

Bibliography

[Abdel-Aziz 71] Y. I. Abdel-Aziz and H. M. Karara, „Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry“, in: Symposium on Close-Range Photogrammetry, issue 11, pp. 1–18, University of Illinois at Urbana-Champaign, 1971.

[AutTech 07] Firma Automation Technology GmbH in 22946 Trittau, Produktübersicht, Downloads und Datenblätter. URL: `http://www.automationtechnology.de`