Software Defined Networking based Data-Center Services

vorgelegt von
Marc F. Körner
MEng
geb. in Berlin

Von der Fakultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Manfred Hauswirth
Gutachter:	Prof. Dr. habil. Odej Kao
	Prof. Dr. César De Rose
	Prof. Dr.-Ing. habil. Thomas Magedanz

Tag der wissenschaftlichen Aussprache: 2. Juli 2015

Berlin 2015

Acknowledgments

First of all, I would like to thank my ad- and supervisor Prof. Dr. habil. Odej Kao, for the opportunity to work for the complex and distributed IT systems department, which was the best work environment I have ever found. He gave me support in all work related areas and words of solace during hard times, like rejected project proposals. I further want to thank all my colleagues from the CIT department for being nearly good friends and sharing every time their honest opinion concerning my work.

Moreover, the OFELIA project team who introduced me to the network research community and gave incitations due fruitful and intensive discussions. In this context I also want to thank Mr. Herbert Almus for his guidance through the regarding project time and the rich administrative knowledge transfer as well as the open ear for ever issue.

I further want to thank some special tubIT colleagues, where I completely spend the first year in this job to profit from their rich knowledge and close cooperation. They introduced me to the design of large scaled campus and datacenter networks and shared their practical insights and experiences, which are beyond any theory and highlight the best practice procedures and daily problems in these areas, namely Michael Flachsel, Markus Hohenhaus, Thilo Wetzel and Timo Riger.

At last, I would like to thank my girlfriend Viktoria-Anne for giving me the support to finalize this thesis. Also special thanks to my dearly beloved parents Kurt and Francine, who formed me to the person I am and will stay forever in my heart. Finally, I want to thank also my beloved grandmother Anni, she was some sort of a nanny for me and would be unbelievable proud, I still miss you so much, rest in peace.

Zusammenfassung

Diese Dissertation beschäftigt sich mit der Frage wie Rechenzentrum Netzwerke von der Integration von Software Defined Networking profitieren können. Hierfür werden einige für Rechenzentren essentielle Netzwerkkomponenten mit einem neuen OpenFlow Ansatz konzipiert, mit der Absicht sie in Netzwerk-Grundfunktionen zu überführen, basierend auf dem entkoppelten Steuerungs- und Daten-Konzept von Software Defined Networking.

Die untersuchten Netzwerk-Grundfunktionen sind typische Komponenten wie sie in nahezu jedem Rechenzentrum vorkommen, dies zeigt den praktischen Hintergrund für die durchgeführte Betrachtung und Analyse. Die in dieser These betrachteten Netzwerk-Grundfunktionen, im speziellen Lastbalancierung, QoS Überlagerungen und Weiterleitung sowie eine Firewall, werden separat untersucht und evaluiert basierend auf dem Netzwerk Anwendungs-Konzept laufend auf einem Netzwerk-Betriebssystem. Der spezifische Rahmen ist isolierte Leistungsergebnisse zu sammeln und zu verifizieren ob dieser Ansatz zu einer grundsätzlichen Verbesserung des heutzutage vorherrschenden Rechenzentrum Netzwerk Designs führt.

Die vorgestellten Komponenten sind prototypisch entwickelt und implementiert, um Messwerte auf einem OpenFlow Prüfstand mit Namen OFELIA TUB Insel, zu sammeln. Die Leistungs-Ergebnisse der Messungen werden benutzt um die Machbarkeit dieses Ansatzes für die Nutzung in einem realen Rechenzentrum zu vergleichen und zu bewerten.

Abstract

This dissertation deals with the question how data-centers networks can benefit from the integration of Software Defined Networking. Therefore some data-centers essential network components are revised with an new OpenFlow approach in order to transfer them to network primitives based on the decoupled control- and data-plan paradigm introduced by Software Defined Networking.

The investigated network primitives are typical components how they appear in nearly every data-center, this builds the practical background for the conducted consideration and analysis. The in this thesis covered network primitives, in particular load balancing, QoS overlays and forwarding, and a firewall, will be explored and evaluated separately based on the network application concept running on top of a network operating system. The particular scope is to collect isolated performance results and verify if this approach in general can lead to an enhancement of the todays predominated data-center network designs.

The introduced components are prototypical developed and implemented to collect measurements on an OpenFlow networking testbed called OFELIA TUB island. The performance results of the measured samples are used to compare and rate the feasibility of this approach for an application in a real data-center environment.

Contents

Chapter 1

Introduction

"The future belongs to those who prepare for it today."
- Malcolm X

Contents

Since the birth of the Internet with the first interconnects of mainframe computers, the ARPANET [61, 62], till today's data-centers and networks related technology has significantly changed [44]. The amount of data and devices, the processing speed, and available bandwidth has dramatically increased, but local area networks are still built with concepts and restrictions of the beginning of Carrier Sence Multiple Access with Collision Detection (CSMA/CD) [1]. All achieved improvements concerning the forwarding process and the forwarding decisions use existing or additional information which is added to the packet header [33] and special decentralized equipment which is able to deal with this information. These standards evolve Ethernet and address a particular problem: in the case of the Spanning Tree Protocol (STP) this is a loop free redundant physical topology.

In contrast, a new promising approach called Software Defined Networking (SDN) [27] came into play. SDN opens up a completely new perspective on networks and the way they are able to work. Furthermore, this technology delivers the opportunity to completely evolve the way of data forwarding by providing a standardized interface for an external control logic to specialized network management control applications, which allow several new use-cases without impact on any legacy technology. It e.g. provides networking substrates with the feasibility to act flexible and dynamically on the requirements of processing or storage elements connected to them. Moreover, it promises to deliver an innovative networking backbone architecture for data-centers, campus-networks and probably also carrier-networks with the aim to provide fast, dynamic and application specific effective packet forwarding. SDN is the key component for an intelligent and on demand network management on top of the physical substrate without required changes of the existing Open Systems Interconnection (OSI) layers [73] or the associated protocols.

1.1 Motivation and Scope

Basic network innovations in general and particularly in data-centers are difficult due to the closed nature of proprietary switches and routers. They are usually closed boxes with vendor specific features and management. In contrast, the idea of SDN is to bring todays networks to a software-defined application platform. This concept and the OpenFlow [46] protocol, as open-source implementation in particular, build the background for the developed new data-center networking approach.

Data-centers rely on high-speed networks with fail-over mechanisms and avoidance of bottlenecks, to provide availability and the best possible reliability. These aspects are essential requirements for hosting services and data with a guaranteed service performance and scalability to provide the best possible service related end-user experience. Other services like the Storage Area Network (SAN) require special Quality of Service (QoS) parameters to realize a high throughput and/or a low latency. Summarizing this requirements data center networks are staying on the following three pillars:

- Availability
- Scalability
- Reliability

These pillars build the parameters for the physical network design. The SDN technology offers the opportunity to enhance and unify all these processes. It acts as a central instance for network allocation and processing compared to the current decentralized method. This simplifies the entire process from the configuration until the management, and delivers alternative approaches to e.g. increase the network performance on the same physical topology. In other words, using SDN does not require a completely new design of the physical infrastructure or technology. It evolves the forwarding process and makes it smart by using a comprehensive and outsourced control logic with knowledge about the entire network, its state and loads for providing the most reasonable forwarding path to the corresponding application or communication partner. The benefits for data-centers are customized network services and applications, are optimized for the required features and available topology. Moreover, quick network improvements and innovations are no longer a problem and can be deployed in a very short time period due to the centralized nature of the overall management and control design.

This thesis postulates an approach for an alternative data-center network concept based on SDN. It is focused on SDN data-center network services. These services will cover and improve the currently predominated management and operational design without the need of rearrangements in the physical topology. This approach reduces the operational complexity by introducing the concerning OpenFlow based network services which are covering the further explained networking task. These tasks are transfered into network primitives to improves the forwarding process itself and dispose of conventional proprietary legacy equipment. In particular, to provide and supply clients with services and a proper service experience, the afore mentioned requirements (availability, scalability, and reliability) have to be enforced in data-centers. This is usually covered by three main components, a Firewall (FW), a Load Balancer (LB) and a QoS related forwarding and network overlay. This is realized as depict in figure 1.1. The figure shows all these components and where they usually located in the data-center forwarding path. The scope of this thesis is to develop and implement a revised and unified OpenFlow approach of the introduced components in particular and the concept in general.

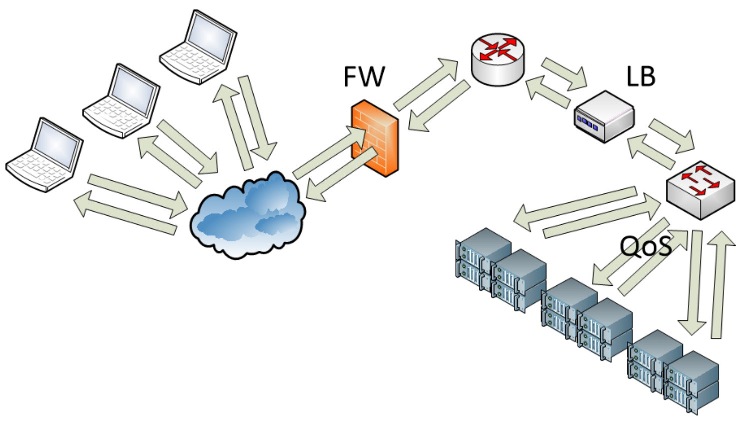

Figure 1.1: Schematic data-center traffic forwarding

The basic idea of this approach is to cover a typical data-center use-case and to demonstrate how this can be migrated, realized, and improved based on the SDN concept. So this thesis is basically a proof of concept work where all afore mentioned parts are explored, investigated and evaluated separately. Nevertheless, as far as possible all evaluation results are based on a special testbed with OpenFlow enabled switches which are producing repeatable results with realistic values how they would appear in a data-centers environment. This means that the presented results are transferable concerning their significance.

These SDN application and the resulting impact will be further explained in the respective chapters as summarized in section 1.3. This thesis is mainly about the questions how powerful these solutions are and what this new networking concept can achieve based on common data-centers network topologies. Therefore, every network application will be introduced, scientifically explored and evaluated on real packet forwarding hardware as far as possible.

1.2 Contribution

The legacy forwarding procedure and design which were introduced in section 1.1 are transfered and redesigned. Features of the dedicated network equipment, as depict in figure 1.1, are integrated into a central SDN management concept. Furthermore, redundant and over-provisioned resources are used to optimize scalability tasks. Finally, a global and dynamic QoS management environment guarantees a reliable service experience. This improves the current state of art concept with:

- transfer of specialized networking tasks to network primitives also known as Network Function Virtualization (NFV)
- exemplary implementation and evaluation of SDN services for a data-center Network Service Chaining (NSC) approach
- a comprehensive network management concept for a typical data-center NSC
- avoidance of special and proprietary network equipment
- avoidance of bottlenecks and single point of failure concepts

Multiple parts of this thesis, covering the identified opportunities and introduced subtopics, have been previously published in the following scientific and international peer reviewed publications:

Conference Papers

1. Marc Koerner and Odej Kao. Multiple service load-balancing with openflow. In *Proceedings of the 13th International Conference on High Performance Switching and Routing (HPSR)*, IEEE, pages 210–214. IEEE publishers, 2012
2. Marc Koerner. The ofelia tub-island an europe-wide connected openflow testbed. In *Proceedings of the 38th IEEE Conference on Local Computer Networks (LCN)*, pages 452–455, Sydney, Australia, oct 2013. IEEE publishers

3. Marc Koerner and Odej Kao. Optimizing openflow load-balancing with l2 direct server return. In *Fourth International Conference on Network of the Future (NoF'13)*, IEEE, pages 1–5, Pohang, Korea, oct 2013

4. Marc Koerner and Herbert Almus. Hla - a hierarchical layer application for openflow management abstraction. In *Fourth International Conference on Network of the Future (NoF'13)*, IEEE, pages 1–4, Pohang, Korea, oct 2013

5. Marc Koerner, Alexander Stanik, and Andreas Kliem. An approach for QoS constraint networks in cloud environments. In *Fourth International Conference on Network of the Future (NoF'13) (NoF'13)*, IEEE, pages 1–3, Pohang, Korea, Oct 2013

6. Marc Koerner and Odej Kao. Oftables: A distributed packet filter. In *The 6th International Conference on Communication System and Networks (COMSNETS)*, pages 1–4, Bangalore, India, jan 2014

7. Marc Koerner and Odej Kao. Evaluating sdn based rack-to-rack multi-path switching for data-center networks. volume 34C, pages 118–125. Elsevier, 2014

8. Alexander Stanik, Marc Koerner, and Leonidas Lymberopoulos. Sla-driven federated cloud networking: Quality of service for cloud-based software defined networks. volume 34, pages 655–660. Elsevier, 2014

9. Marc Koerner, Alexander Stanik, and Odej Kao. Applying qos in software defined networks by using ws-agreement. In *Cloud Computing Technology and Science (CloudCom), Proceedings of the 2014 IEEE 6th International Conference on*, volume 2, pages 893–898. IEEE Computer Society, December 2014

10. Thomas Renner, Alexander Stanik, Marc Koerner, and Odej Kao. Portable sdn applications on the paas layer. In *Utility and Cloud Computing (UCC), Proceedings of the 2014 IEEE/ACM 7th International Conference on*, pages 497–498. IEEE Computer Society, December 2014

11. Constantin Gaul, Marc Koerner, and Odej Kao. Design and implementation of a cloud-federation agent for software defined networking. In *Proceedings of 2015 IEEE International Conference on Cloud Engineering (IC2E 2015)*, pages 323–328, Tempe, AZ, USA, March 2015. IEEE

Journal Papers

12. Marc Koerner, Herbert Almus, Hagen Woesner, and Tobias Jungel. Metrics and measurement tools in openflow and the ofelia testbed. In *Lecture Notes in Computer Science (LNCS) 7586*, pages 123–134, Heidelberg, 2013. Springer

13. Marc Sune, Leonardo Bergesio, Hagen Woesner, Tom Rothe, Andreas Koepsel, Didier Colle, Bart Puype, Dimitra Simeonidou, Reza Nejabati, Mayur Channegowda, Mario Kind, Thomas Dietz, Achim Autenrieth, Vasileios Kotronis, Elio Salvadori, Stefano Salsano, Marc Koerner, and Sachin Sharma. Design and implementation of the ofelia fp7 facility: The european openflow testbed. *Computer Networks (COMNET)*, 61:132 – 150, 2014. Special issue on Future Internet Testbeds - Part I

14. Alexander Stanik, Marc Koerner, and Odej Kao. Service-level agreement aggregation for quality of service-aware federated cloud networking. *IET Networks Journal*, -:1–6, 2015

1.3 Outline of the Thesis

This section gives a brief overview about the structure of this thesis. It also reflects and introduces the three main coverage areas of the OpenFlow management- or rather control-applications required in data-centers as explained in section 1.1.

Every chapter explores one distinguished area and describes the concerning approach as well as opportunities for improvements with the corresponding OpenFlow solution, as shown in figure 1.2. The remaining chapters are structured as follows:

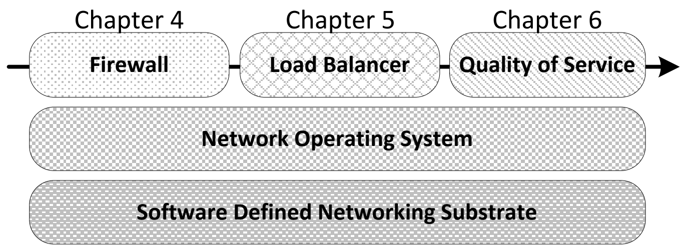

Figure 1.2: Typical data-center network service chain

Chapter 2: Background provides a state of the art analysis for networking in data-centers and the service related requirements. The SDN paradigm is explained and the OpenFlow protocol and other used technologies will be introduced.

Chapter 3: Testbed for Evaluation describes the detailed structure, organization, and composition of the local OpenFlow testbed, called OFELIA TUB island. This testbed was used to evaluated the exemplary implementations of the network applications which are presented in this thesis in order to verify their feasibility on real OpenFlow enabled network equipment. This chapter gives a detailed explanation about the used hard- and software components and builds the basis for the classification of the outcomes.

Chapter 4: Packet Filtering points out opportunities for packet filtering in OpenFlow networks. Further an approach for an innovative distributed packet filter tool with iptables similar interface is introduced. The alpha version of this innovative decentralized application for distributed packet filtering is evaluated, to show the impact and benefits of traffic pre-filtering and fine granular perimeter networks.

Chapter 5: Load Balancing introduces novel load balancing procedures and describes the demonstrator implementation for a SDN based LB solution. A common load balancing concept is realized on the OpenFlow concept. In particular, network service based load balancing concepts are introduced and samples based on the implemented network application prototypes are evaluated and compared. Especially the first commonly host based load balancing approach used in data-centers is further revised in this chapter, in order to address the identified performance issues of the underlying switching devices and their capabilities.

Chapter 6: Quality of Service introduces some QoS design aspects and realization concepts for QoS constraint SDN and the herewith connected Quality of Experience (QoE). The chapter starts with new opportunities for QoS enforcements in OpenFlow networks and investigates realization concepts and challenges, which will finally be applied and exemplary evaluated. Moreover, this chapter goes deeper into a Private Network-to-Network Interface (PNNI) related design concept for the OpenFlow control-plane, to increase the scalability and separate responsibilities under technical and non-technical aspects for large scaled SDN networks. In particular, this addresses the fundamental problems concerning the centralized SDN based control-plan concept for large scaled OpenFlow environments.

Chapter 7: Conclusion will provide a final summary of the used methods. The chapter is closing this thesis with a prospect on further research in the area of OpenFlow data-center networks.

Chapter 2

Background

"We can draw lessons from the past, but we cannot live in it."
- Lyndon B. Johnson

Contents

This chapter provides an overview with regards to the applied technology and further introduces some basics for chapter 4, 5, and 6, where conceptual opportunities to improve or at least migrate data-center networks and network services to SDN are proposed. Moreover, the testbed installation for the evaluation presented in chapter 3 is based on the technology and concepts briefly introduced in the following sections.

First, a brief definition and explanation of the SDN paradigm itself is given. Then the OpenFlow technology will be explained and considered as enabler for the in sec. 1.1 introduced new approaches. Finally, a brief overview about OpenFlow controllers and network virtualization tools is given. The network virtualization tools are used to develop the network services and create testbed.

2.1 The Software Defined Networking Paradigm

Software Defined Networking basically describes a complete new networking paradigm. The first time it was proposed 2008 in [46] by introducing the idea of OpenFlow. The paper basically describes an innovative new design approach for Ethernet driven networks and can be summarized with two main differences compared to the traditional Ethernet networks.

- decoupled control- and data-plane
- logically centralized network intelligence

The introduced paradigm proposes a separation of the forwarding hardware (data-plane) and the control logic (control-plane) of packet forwarding elements. The control logic is introduced as a centralized software driven control entity. It contains informations about the entire network compose of the many data-planes. This architecture again is later abstracted and condensed in a generalized description of the entire SDN paradigm [27]. The introduced SDN described henceforth a layered model with an additional application layer on top of the control layer/control-plane:

- Application layer
- Control layer
- Infrastructure layer

The proposed changes may seem trivial, still, the consequences for the entire networking area are not conceivable. The proposed paradigm effects the basement of actual network devices architectural design. Further it influences the way the devices (or more precisely the network in general) is supposed to work.

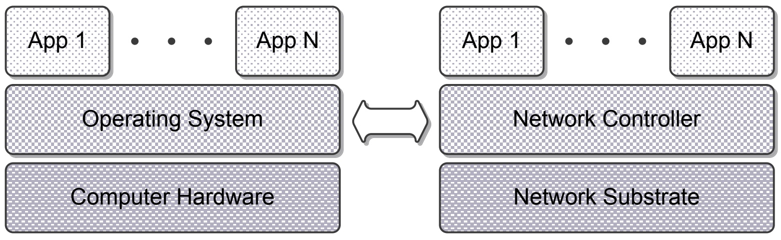

Figure 2.1: Comparison of computer and SDN application abstraction

This is basically comparable to the way of computer platforms and applications had evolved due to the introduction of the Operating System (OS), as depict in figure 2.1. From a specific application created to run on dedicated computer hardware to an Operating System based scheduling and management of applications on top of the OS dependent Input/Output (I/O) system, Hardware Abstraction Layer (HAL), and process model.

2.1.1 OpenFlow

The OpenFlow[22] approach aims to create a vendor independent, standardized interface for adding and removing flow entries, some sort of Forwarding Information Base (FIB) entries, on switches and routers or even more precisely using the terminology of this thesis packet forwarding elements. Thus, the network flows can be defined dynamically based on the current network state and the expected load. The implementation is based on an Application Interface (API), which allows the modification of the flow tables representing e.g. the forwarding decisions on a particular switch. The flow tables consist of flow entries, as depict in figure 2.2. They represent the mapping information between a header information and the action(s) to be executed if a packet matches this particular header. Moreover, the flow entries do also contain different counters which e.g. simplify the statistic analysis. The header matching pattern is basically used for the matching process against the Ethernet packet-header of an incoming packet. Switching or routing operation like forward or rewrite and forward the incoming packet to a specified port or ports are typical actions attached to the flow entries. For every processed packet the flow entry counters are incremented regarding the packets attributes, e.g. byte length. More information are provided in the regarding OpenFlow specification.

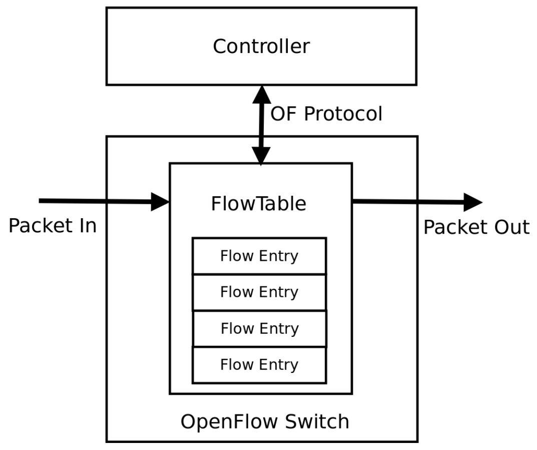

Figure 2.2: Abstract OpenFlow forwarding model

OpenFlow is today's best explored and mostly deployed SDN solution. Moreover, OpenFlow (OF) is a protocol implementation which exactly follows the SDN approach. Meanwhile, from the first research based publication till today's further standardization by the ONF [52], it is more or less the most established SDN solution in the market. Actually it has made its way into switching products from various vendors and is already included to their product portfolio.

2.2 Nomenclature

Since SDN is a relative new technology this section will briefly introduce some nomenclatures used within this thesis to clarify the terminology used in the following chapters. Thus, it's ensured that all readers have a common understanding.

Packet Forwarding Element

Packet forwarding element is a generalized term for any networking device – a piece of physical hardware which is designed to forward packets – which is capable to forward Ethernet based network packets. Thereby it does not matter if the device is also able to process or to modify the packet. Usual devices are repeater, hubs, switches, routers and so on.

Network Appliance

A Network appliance is usually a dedicated piece of hardware. It is designed, constructed and deployed within the network topology to serve a particular purpose and realize a specific network function. Usual network appliances in ever enterprise network are for instance firewalls, deep packet inspection firewalls, routers or load-balancers.

Network Operating System

As introduced in section 2.1 SDN uses a controller which is outsourcing the control intelligence. This controller does not only implement the regarding underlying SDN protocol it further provides an API for network applications.Due to these functions the synonym Network Operating System is widely used. Both terms are usually describing the same software component which implements the SDN protocol, e.g. OpenFlow, and directly communicates with the packet forwarding elements.

Network Functions

A network function describes a particular function to transform or process network packets. It can basically described as a set of orders how to proceed and process a particular packet or flow. These processing orders are encapsulated in some sort of logical block similar and comparable with a software module.

Network Function Virtualization

NFV describes a network function which is implemented and realized as an network application. It's running on a SDN controller or network OS instead of on a network appliance.

Network Service

A network service equals a network appliance when it's considered from the operational perspective. It usually provides a particular network function which is implemented on the principles of NFV. The distinguishing aspect is, that a network service is rather the operator perspective on a network function.

Network Service Chaining

Network service chaining describes a concept for the orchestration of network services as the introduced firewall, load-balancing and QoS forwarding in section 1.1. Network service chaining is a very contemporary topic and is meanwhile also called network function chaining.

Flow

A crucial term for the technical sections is the flow definition. The flow terminology henceforth used is the abstract definition of a vectored point to point connection. This connection is drawn over one or multiple packet forwarding elements and based on parameters specified by the packet header fields and the associated action or actions.

2.3 OpenFlow Control-Plane Tools

This section briefly introduces tools which were used in the development process of the different network services presented in the following chapters and the deployment of the testbed as well as presented in chapter 3.

2.3.1 OpenFlow Controllers

Nowadays several OpenFlow controllers like OpenDaylight, Trema, NOX, POX, Floodlight are available. OpenFlow controllers are basically providing a particular programming language API to wrap the OpenFlow protocol and provide a comfortable enviroment for the rapid network application development. Everything started with NOX [31] donated by Nicira to the research community. The NOX controller provides developers with the opportunity to build C/C++ or phyton applications on top of the framework. It was basically the first available OpenFlow controller which was used in a variety of research evaluations at this time, also for the first implementations and evaluations [38] for this thesis. Often controller do not only wrap the OpenFlow standard[22], they usually provide even an SDN ecosystem. This means they have a multitude of basic functions like the network topology detection. These controllers allow the fast development of general networking applications or even the previously introduced virtualized network functions.

Modern object-oriented programming and design patterns controllers like Floodlight [49] became more convenient. This controller for instance provides a versatile framework for developing OpenFlow applications with Java. Another opportunity to use Floodlight or to build applications with it is to use the Representational State Transfer (REST) API, which is provided by the Static Flow Pusher module. REST is a Hypertext Transfer Protocol (HTTP) based web-service API [60] which provides several advantages, e.g. an easy Command Line Interface (CLI) usable design. This makes it very simple to interact with the controller. For example: if all OpenFlow switches have successfully connected by just sending the concerning REST command, the evaluation becomes very easy.

2.3.2 Network Virtualization

SDN opened the closed and distributed management nature of network forwarding elements and networks in general. It provides a completely new opportunity to consider networks as a software controllable hardware resource. It is the basic enabler for a lot of new and innovative network applications and offers a new approach and understanding of network virtualization. The consequences are very widely spread. They reach from a simple Virtual Local Area Networks (VLAN) similar separation or encapsulation mechanism up to a full virtualization of network resources or even the network in general, which is well known from the cloud computing.

FlowVisor

The FlowVisor tool was the first development in this new direction. It was the logical consequence of the demand or question, how to use different controllers at the same time on the same substrate. So the question was how to share the physically available resources to e.g. test different network applications without larger interference. The idea for a tool called FlowVisor was born. FlowVisor is just an OpenFlow control channel proxy server with an additional administrative interface to define the regarding user policies.

The user is getting a network slice which is connected to his controller. FlowVisor is only evaluating the OpenFlow commands according to the previously defined slice policies and enforces them. In detail, it forwards OpenFlow control messages which are not violating the slice policies. It drops messages and send back an error message for commands which are doing this.

This makes it possible to use several controllers and applications on the same physical network. For instance, it provides the opportunity to test new algorithms directly on the productive network without influencing it [66].

OpenVirteX

Actually FlowVisor was very successful, but unfortunately the basic design and its principals are only working for the protocol 1.0 standard. When multiple flow tables were introduced with the OF version 1.1 it was clear that another solution needs to be found. Again, an innovative idea was realized, the full network virtualization.

OpenVirtex is using an internal Internet Protocol (IP) mapping mechanism which maps every traffic entering the OpenVirteX controlled network at the ingress packet forwarding element to an internally managed IP class A private network address. This is used to separate traffic belonging to different users/controllers. It is completely transparent, since the user gets a full virtualized network [13]. He is even not able to draw any conclusions about the underlying physical topology, similar to a full virtualized host is not aware about the underlying hardware.

OpenVirtex is currently available in an alpha version. Unfortunately recent investigations [28] concerning the usage of this tool in research environments, like the in chapter 3 described OFELIA TUB testbed, are indicating that not yet the entire possible feature set is completely implemented and working. Still, this implementation is the first real open source SDN based network hypervisor [14].

Chapter 3

Testbed for Evaluation

"Equipped with his five senses, man explores the universe around him and calls this adventure Science."
- Edward Powell Hubble

Contents

This chapter introduces the evaluation platform called OFELIA TUB island, which was used to evaluate the different exemplary implemented network services which will be presented in the following chapters.

Moreover, it contains and describes a major part of this work and thesis in terms of building and operating an innovative OpenFlow testbed addressing the world wide SDN research community since 2010. It builds the foundation for all network services which will be introduced and explored in the following chapters and presents a self-contained scientific innovation and contribution.

The following sections in this chapter are slightly modified citation of the in section 1.2 listed publications [35]. They further contain selected contents from the also in section 1.2 listed publications [71].

3.1 Open Flow in Europe: Linking Infrastructure and Applications

Openflow in Europe: Linking Infrastructure and Applications (OFELIA) [8] is a European Union (EU) funded project for the creation of a distributed OF testbed spanning over several European countries, similar to the Global Environment for Network Innovations (GENI) [6] project in the United States (US). OFELIA is generally focused on offering free of charge networking and computing resources for developer, experimenter and researcher for future Internet related research and experiments. It provides a platform for OF controller development and the corresponding evaluation of newly developed networking concepts. The OFELIA project is an association of several OpenFlow testbeds across Europe. A single testbed in this structure is called an island, which reflects the character of SDN islands surrounded and embedded in today's the common network technologies. Basically, the islands are not built homogeneously, every island hosts different types of equipment and has an individual topology.

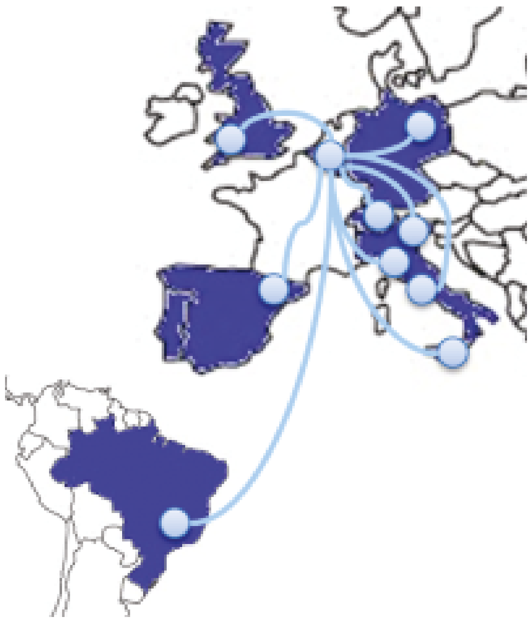

Figure 3.1: OFELIA islands

Altogether they are building the biggest OpenFlow testbed outside the US, which is still growing due to new testbeds which are joining e.g. the Federal University of Uberlândia (UFU) in Brazil. Fig. 3.1 depicts an overview of the current islands. With the growing amount of islands also the diversity of equipment increases. Actually, the federated testbed contains nearly any kind of networking substrate, from normal packet switches with copper and fibre links over optical switches up to wireless nodes. Furthermore, network equipment from several vendors like NEC, ADVA, HP, Pronto, Arista as well as Xilinx NetFPGA's is used, just to mention some of them. Active and operating OFELIA islands which are also participating in the federation are depict in figure 3.1 and are further listed with their concerning affiliation in table 3.1.

City	Country	Affiliation
Barcelona	Spain	i2CAT
Berlin	Germany	TUB
Bristol	United Kingdom	UNIVBRIS
Castelldefels	Spain	CTTC
Catania	Italy	CNIT
Ghent	Belgium	iMinds
Pisa	Italy	CNIT
Rome	Italy	CNIT
Trento	Italy	CREATE-NET
Uberlandia	Brazil	UFU
Zurich	Switzerland	ETH

Table 3.1: Federated OFELIA islands

3.2 General Distributed Testbed Architecture

OFELIA is designed for providing an independent standalone island operation and to become part of the distributed testbed with further federation options. Therefore, the following two main building blocks will be introduced, the soft- and hardware architecture for an OFELIA testbed.

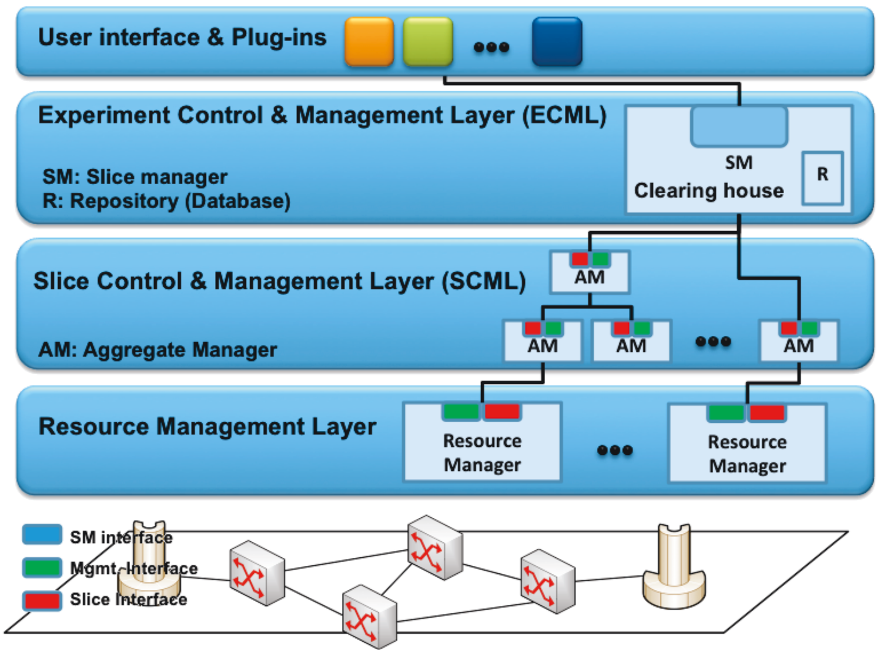

Figure 3.2: OCF software architecture

The software architecture in shape of the OFELIA Control Framework (OCF) [50], available under Berkeley Software Distribution (BSD) license. The OCF supports the whole experimentation life cycling which will be further explained as well as the building blocks of the OCF.

- Experiment Control and Management Layer (ECML)
- Slice Control and Management Layer (SCML)
- Resource Management Layer (RML)

First, the bottom of the software-stack, the RML is considered. This layer directly communicates with the resources, which are partially island specific, and exposes them through a simple resource management API. The RML is responsible for the resource reservation and configuration. The next higher layer, the SCML consists of Aggregate Manger (AM)'s, which again exposes the aggregated types of resources with an API over a northbound interface. This stands in contrast to the southbound interface which communicates with the RML. The SCML is also called the federation layer, because it can be used to expose resources to other federated testbeds, as done in OFELIA. The top layer is the ECML, which is used to manage the project as well as a used collection of resources, called slice. It sends resource requests to the SCML which is aware of the availability and waits for the regarding response. The OCF generally provides the entire software stack, as depict by fig. 3.2, and a template for the implementation of RML applications for testbed specific resources, like the Berlin Open Wireless Lab (BOWL) [5] nodes at Technische Universität Berlin (TUB).

The hardware or more precisely the network architecture consists of three networks, separated by their purpose and simplifying the federation process. The purpose and connection between these networks, depict by 3.3, can be reduced to three diffrent access levels.

- Management Network (MGMT)
- User Control Network (CTL)
- Experimental Network (EXP)

The MGMT is an out-band control network, exclusively for the local testbed provider and administrators, also called island manager. This network ensures the remote administration of physical and virtual testbed devices, and also enforces security through this separation. In contrast, the CTL is an island/OFELIA testbed comprehensive management network for users of the facility. It provides the users access to their project related resources and the control applications, e.g. the Expedient. The CTL is a routed private class A network which could be directly accessed by the user through an OpenVPN[55] tunnel. The last but most interesting network is the EXP. This is an island comprehensive plain L2 OF network and gives the user the opportunity for any kind of network experiment which possible on the regarding physical provided infrastructure.

Related to federation scenarios the CTL uses a star topology in order to avoid loops in the control part. The concerning hub is the iMinds-island in Gent, Belgium (BE) with the TUB island as backup hub. If the main hub is

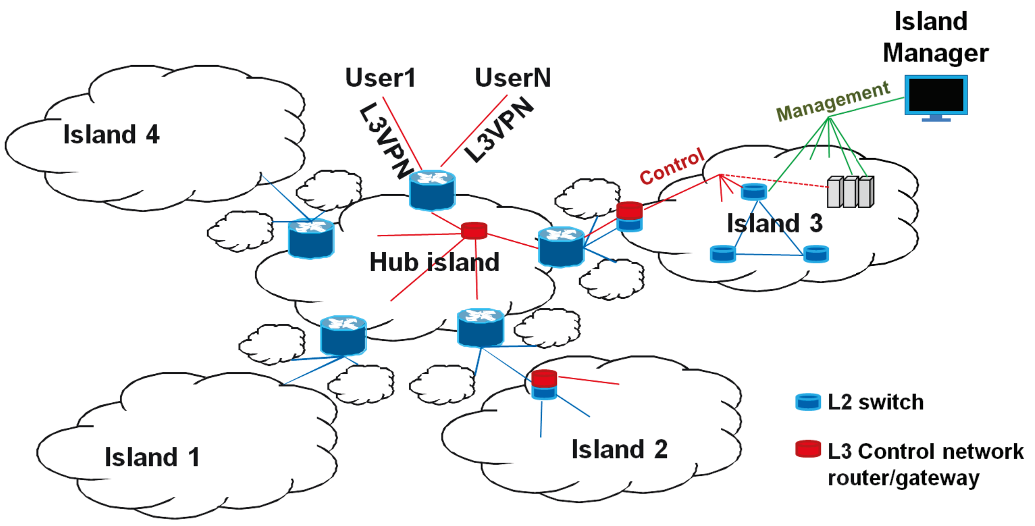

Figure 3.3: OFELIA network concept

not available an Open Shortest Path First (OSPF) instance on all islands is changing the routes so that allocated resources, especially VM's, can still be managed by experimenters.

3.3 The TUB Island

The currently available testbed installation at TUB is composed of a meshed network with three NEC switches working with a port bound OF instance and protocol standard 1.0. TUB-islands further hosts two user Virtual Machine (VM) servers (VMS1/2) each connected to all switches. These servers use a special OCF libvirt [7] agent for VM allocation via Expedient, which is the project and experiment management application for the entire TUB and OFELIA testbed. In addition, there is a third server (IBM) which hosts the OCF, Expedient and AM's for OpenFlow VM resources, as well as a Simple Network Management Protocol (SNMP) monitoring application and a Windows 2008 terminal server. The terminal server is used to operate with the Ixia test system, and hosts the required applications. This is a profession enterprise test system which can be requested at TUB island for several special tests from L2 till L5. An additional HP switch in legacy mode separates the different networks and provides Internet connectivity to the devices.

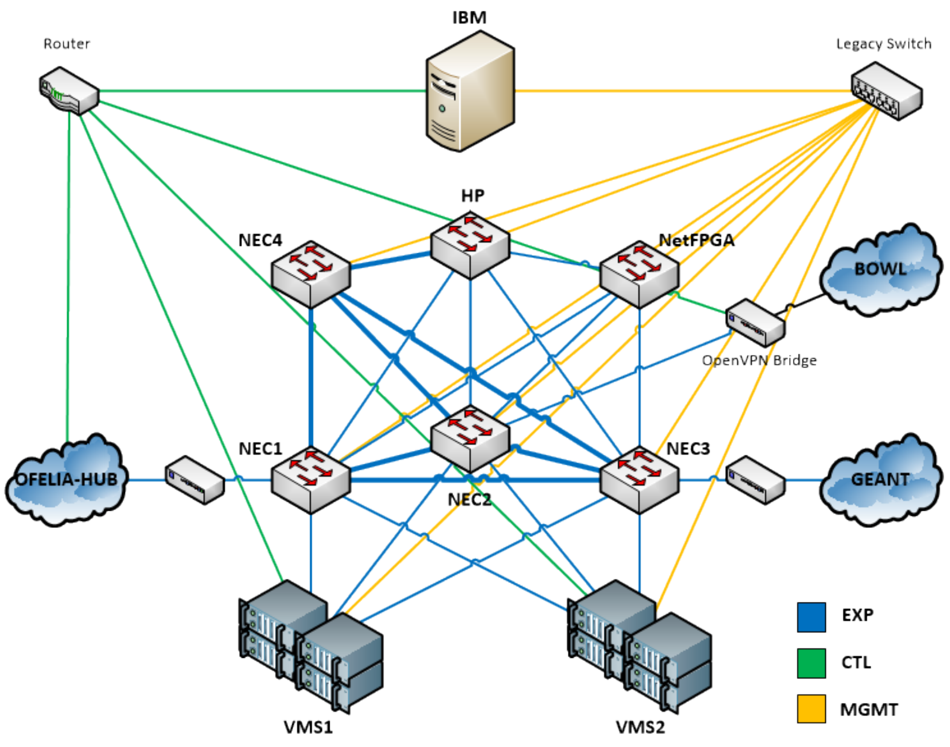

Figure 3.4: Simplified physical network topology at the TUB island

Name	Model	Comment
NEC1-3	NEC IP8800/S3640-48TW	Up and running
HP1	HP 5400	Up and running
NetFPGA	2x Xilinx Virtex-II Pro 50 FPGA	Maintenance
VMS1-2	E3-1240 4xCore with 16GB RAM	Up and running
Ixia	Ixia T1600 + 3x 4port GBit line cards	Up and running
IBM	2xXeon 4xcore 2.4 GHz 6GB RAM	Up and running

Table 3.2: Equipment hosted at TUB island

The OCF is one of the key components for the TUB island experiment management. It is a multi-layer software stack, as mentioned in sec. 3.2 and shown in fig. 3.2, which is able to build a testbed comprehensive management system on top of the physical OFELIA infrastructure. It provides a web-based User Interface (UI) called Expedient and supports all steps for creating experiments and allocating resources. This tool supports the whole experimental life-cycle. It communicates and visualizes resources with the help of plug-ins which are connected to the corresponding AM's, in this particular case the Virtualization Technology (VT) manager and the opt-in manager. The VT manager aggregates and administrates the VMs which are hosted on the Virtual Machine Server (VMS)s, as shown by fig. 3.5, supported by the OCF libvirt agents. The OpenFlow flowspace is allocated through the Opt-in manager, with the help of the OpenFlow slicing tool FlowVisor [65, 66]. Expedient, VT, Opt-in and FlowVisor are encapsulated in VMs on the IBM server. The VMS installation hosts a default Debian 6.0 image for the paravirtualized user VMs. This image contains several OpenFlow tools, e.g. the NOX [31, 74] OF controller, for a fast and easy experiment starting point. In general the researcher is free in his decision and can also download additional software packets like the FloodLight [49] if he prefers a special controller or wants to try a pre-developed piece of software.

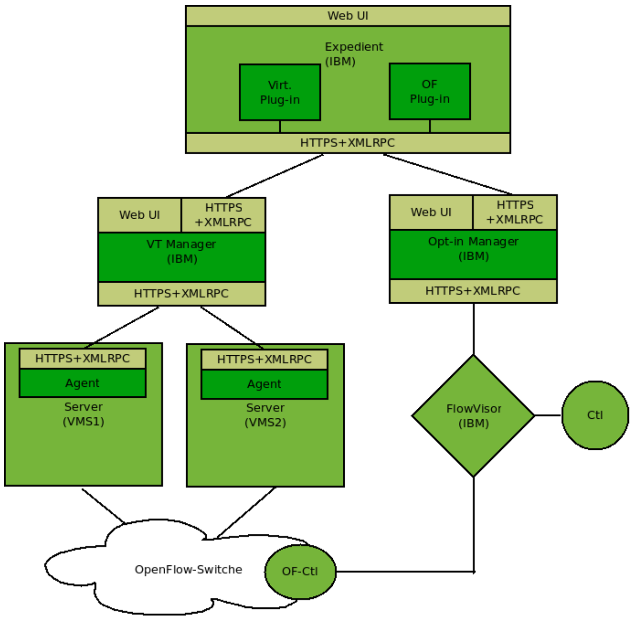

Figure 3.5: OCF installation at TUB island

Fig. 3.4 shows the physical topology of the equipment listed in tab. 3.2. The different colored lines are representing the different networks mentioned in sec. 3.2. The blue interconnections stand for the OF EXP network, which is completely under experimenters control concerning allocated resources and IP configuration. The green connections are CTL related, which is used by the experimenter to access the resources. It is a private network with fixed IP addresses for every island. The black lines are the TUB internal MGMT interconnects. The picture further shows the additional connection, realized with an OpenVPN bridge, to connect BOWL network. The wireless BOWL nodes are in the integration process. They are attached to a TUB island project under control of the German Telekom Laboratories those who are the BOWL operators. The T-Labs work on an AM and the corresponding Expedient plug-in to integrate them into the TUB island and make them available as resources.

3.4 Experiments

In October 2013 there were 14 active projects only on TUB-island, the Berlin OFELIA site. Furthermore, the TUB-island provides resources for three more federated and co-located projects on other OFELIA testbeds. Besides the periodically scheduled commissioning tests, TUB has also used the testbed for several research evaluations. For example the developed OpenFlowS load balancer for multiple services was evaluated on the local island. The concept and the measurement samples collected on the TUB island were published [38] to the networking research community. Moreover, an early version of the load balancer was presented as a demo [3] a few month before at the GENI Engineering Conference (GEC) 13 in Los Angeles.

Also other OFELIA project partners demonstrated their research results and the concerning publications collected on the federated testbed in the area of virtual topologies [64, 24], inter-testbed network virtualization [29] and Information Centric Networking (ICN) [80, 47]. Additional publications from people not involved in the project [67, 15] also demonstrates the excellent research opportunity for scientists.

Chapter 4

Packet Filtering

"Sometimes a cigar is just a cigar."
- Sigmund Freud

Contents

Firewalls are a powerful instrument to protect networks and enforce security. They are very common and every data-center or enterprise network needs and uses them. Moreover, they are building the first line of protection between the public Internet and the data-center. For the data-center service chaining model this means that an incoming data inquiry or service request has first to pass a firewall network service as presented in figure 4.1.

This chapter provides a design approach and evaluation for a firewall network service on the basis of a network application. The contents appearing in the following sections are selected and slightly modified citations of the in section 1.2 listed publication [41]. The mentioned application, its additional pros, as well as the potential impact will be explained in detail. Different aspects of network protection based on packet-filters are explained, which

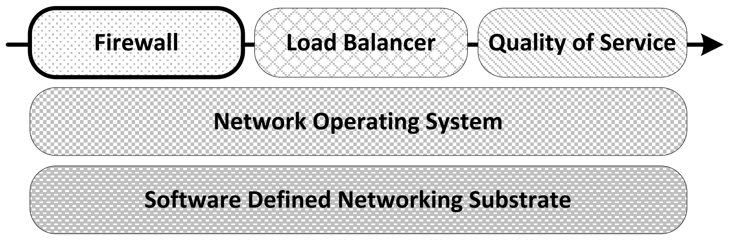

Figure 4.1: Firewall network service chain link

finally leads to the network service proposal as innovative distributed firewall concept. The introduced concept combines several established and new technologies to a powerful firewall application named “oftables”, which is able to cover security requirements of data-centers and enhances this operational bottleneck.

4.1 Packet Filters and Perimeter Networks

Nowadays, complexity of the Internet has reached a level where IPv4 addresses are running out. Quantities of data with several contents are distributed all over the world. With the rising amount of data more people also focus on the sensitivity of the data and the corresponding security. The privacy issue is one of the most important requirements and challenges today, especially for personalized data. Firewalls are supposed to provide a powerful protection mechanism for data and networks. They can be separated into two main areas: The network firewalls, which work on the lower International Organization for Standardization (ISO) OSI layers, typically layer three till four, and those which work on the higher layers called application firewalls. Network firewalls are also called traffic or packet filters. They filter network traffic by characteristics of the packet header. Usually they are deployed and operate on routers, where they decide which traffic is allowed to pass the router from one network to another. On the other hand, many application firewalls are working on hosts and observe processes or the process data. An exception are for instance proxy servers or deep packet inspection firewalls, which are also located in the network path, but have a closer look into the payload a packet is carrying. They also work as packet filters, but are very expensive due to their specialized hardware for the payload data analysis at line rate. However, firewalls which are working with packet filtering based on the header pattern are security tools which are used in every enterprise network.

The security level of a facility and the accessibility demands for its applications and services is crucial for the concerning network design. Security network designs can be distinguished into three security separation concepts: Networks in top-secret areas are typically totally isolated. They consist of private network with no link to the Internet, they are strictly physically separated and provide only internal data communication. These networks are not public accessible. Furthermore, there are public available networks which are typically only protected by a firewall. These firewalls usually allow all incoming traffic related to the provided services, e.g. web-sites or e-mail. Between these concepts are perimeter networks [83], which are often called Demilitarized Zone (DMZ). A single DMZ consists of the fire-walled Internet link and semi accessible network behind, as well as the not external accessible internal zone protected by a second firewall beyond the DMZ. These networks are usually designed to provide different services hosted by the servers in the perimeter network for external and internal usage. Internal means all local e.g. workstations behind the second firewall, which are not supposed to be externally accessible. This basically reflects the two different user groups – trusted and untrusted – and their corresponding access authority depending on their physical network location.

Sometimes, depending on the authority level of the departments in for instance a university, it is required to block traffic between different locations in the internal network. For example filtering or blocking traffic between student computer pools and university departments. The typical solution is to separate them into sub-networks and route and filter traffic on layer three between the sub-networks. In this scenario the filter rules can be directly deployed on the router to define and categorize traffic which is allowed to pass the concerning network border. To avoid separate wiring for each of these networks, they are typically separated with VLAN tags based on the IEEE 802.1Q standard [2]. The virtual separated traffic is forwarded by the access switch to the aggregation switch and usually up to the core network where the router is located. The router firewall inspects and approves the traffic and sends it via the VLAN gateway the same way back. This traffic overhead and the other security issues due to e.g. faked VLAN tags do not provide a secure network solution at all.

A good example for a packet filter firewall is the iptables [4] application. It is a traditional multilayer [48] firewall for Linux hosts and routers with all usual architectural modules of a packet filter. Distributed firewalls are an approach to improve a regular packet filter concept like iptables. These firewalls have many advantages, e.g. better reliability and scalability as described in [19]. The introduced approach uses IPSEC[16] and provides a decentralized host to host protection mechanism, which can extend the filter mechanism between networks in general. With regards to OpenFlow the Floodlight controller has also a module which enables a packet filter opportunity. It provides an Access Control List (ACL) for filtering traffic at the ingress switches of the OpenFlow network. Unknown flows are processed in a reactive manner and are compared to the rule list. After being approved by the ACL they can pass the entire network without additional examination. This example shows the relevance and importance of this topic.

4.2 Distributed Packet Filter Architecture

The fundamental purpose of firewalls, or as in this particular case port-filter based firewalls, are a couple of security demands. Firewalls are providing confidentiality and integrity in networks. They are blocking or permitting traffic to protect sensitive resources or data from unauthorized access. It is an enforcement mechanism of access control policies between networks, or more precisely trusted and untrusted networks, which could also be a semi trusted perimeter network. The oftables tool is extending this approach by also enforcing security policy e.g. between neighboring hosts. The oftables concept goes beyond every of the former mentioned concepts in section 4.1 and delivers a completely decentralized, scalable and reliable firewall application with an approved management concept. This approach introduces a clean and modular architecture and controller independent design with a central persistent storage and a decentralized administration interface. The administrative interface uses a command line administration and parametrization which is similar to iptables. Furthermore, the implementation could later be easily adopted to other controllers without reimplementing the application core components. This is also working for the central filter rule storage database, which can be replaced as well.

The major benefit of the presented approach in this section is that the firewall rules are distributed over several devices. This increase the processing performance or more precisely the processing speed, by also providing a much finer granularity. Basically, in this context granularity means that oftables enables traffic filtering between any ports of a switch. This offers the opportunity to avoid several attack scenarios, which will not be covered by regular firewalls. Basically network attacks can be differed in two scenarios, consisting of external- and internal attacks. Common firewalls are separating into trusted and untrusted networks, as mentioned in section 4.1 and secure the internal network against the external as well as attacks which are coming from this network.

In contrast, distributing firewall rules over the switches in a network create several perimeter networks. This allows to avoid internal or insider attacks by e.g. subverted hosts. It increases the processing speed by reducing the length of the filter rule chains. Rule chains are typically all processed on one device and sequentially checked against the packet header. Moreover oftables map the filter rules into the Ternary Content Addressable Memory (TCAM) of the switches. TCAM processes look-ups in parallel and also supports wild-card matching. This is an innovative mechanism to distribute filter rules and the concerned look-ups to every switching node in the network. This can also reduce the overall network load by early packet drops and the avoidance of a single inspection unit where the traffic has to pass-through first. The afore described VLAN routing overhead can also be avoided.

Basically two insertion modes for flow entries can be differed. These modes

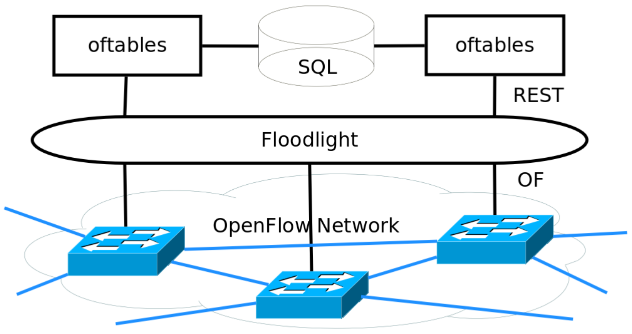

Figure 4.2: System concept

are proactive and reactive. Reactive means the insertion of a flow entry by the controller into the switch as result of a packet-in message, received from the switch. Packet-in messages are sent to the controller, if a packet without matching flow entry is detected. Proactive means insertion of predefined flow entries without the OpenFlow notification message by the switch. To reproduce the behavior of a normal learning switch, it is common to use a reactive controller mode to insert flow entries with the corresponding communication partner destination port and the Media Access Control (MAC) address. Reactive flow processing is a good approach in this particular case, but can also result in performance restrictions in the data-path depending on the processing speed of the controller and the number of packet-in messages to process. However, the oftables architecture is focused on a proactive processing concept to avoid any influence on the processing performance of the data-path and other processing units. This is achieved by a pro-active firewall rule installation with the concerned flow entries.

The iptables application provides several default chains, input, output, forward, pre- and post-routing. In contrast, the oftables application uses only the input chain. This is not a restriction, it is a conclusion of the multiple controller network approach with predefined firewall rules. The oftables application only enforces dropping of packets, which violate security restriction, defined by the regarding flow entries on the switches. Every other operation is a task which will be processed by another network service, after approval through the oftables. In other words, features like forwarding or routing are in this case processed by the regarding OpenFlow floodlight module and are not part of this firewall application. This provides a clear separation concerning the network REST services and avoids interference effects with other application modules on the controller.

The rule specification string/command can contain every matching pattern from ISO/OSI layer one until four, supported by OpenFlow. This means in detail:

- Device (switch or router data-path ID)
- Interface (number)
- MAC header (eth-type, vlan, src-mac, dst-mac)
- Network header (src-ip, dst-ip)
- Transport protocol (tcp, udp, port-number)

Also wild-cards are allowed and completely supported in every layer and are part of the regarding header and concerned matching pattern. They can deliver additional network performance by e.g. pre-filtering traffic. The input chain can be used with an accept or a drop policy. The usual procedure to formulate rules is to use accept rules if the general policy is drop and vice versa. The default mode for oftables is the accept policy. This mode is almost the ideal default solution for the proposed oftables application, because predefined dropping rules are supposed to match against several packet types. The benefit of this concept is that with oftables installed flow entries with no action are dropping the forbidden packets or flows before they where given to any other controller, e.g. switching. This is ensured by the regarding flow entry priority, wherewith installed in the flow tables. This is a straight forward solution if we consider the technical background of the underlying technology. Using a drop policy and accept rules can also be realized but makes things more complicated. Thus, this was not implemented. Nevertheless, the following enumeration will briefly point out some opportunities for an engineering solution based on the in section 4.3 presented implementation.

1. The first solution is to have one special flow entry which, reflects the drop policy. This can be realized by a flow entry with a layer one till four wild-card matching header pattern and a low execution priority for a final packet dropping order. In case that the switch is not finding an appropriated other flow-entry in its table, this special entry will fit and drop the packet. The accept rules have to use the forward to "CONTROLLER" action, which encapsulates the packet and sends it like a regular unknown incoming packet in a reactive manner from the switch to the controller. This triggers the reactive regular processing of another service, for example the MAC learning switch. The only requirement is that the other service has to use a higher priority for the flow entries as the drop entry from oftables.

2. Another opportunity to realize such a drop policy, is the logical negation of the filter rules. This means, that the policy engine has to reformulate the rules with a logical negated content by taking care on the rule statement itself. This is definitively a very TCAM exhausting method, because a simple accept rule will end up in a huge amount of blocking rules or more precisely flow entries. Since the TCAM amount is restricted and currently one of the issues in OpenFlow- and OpenFlow hybrid switches, this method is not recommended.

3. OpenFlow version 1.1.0 [23] and later versions and the hardware related implementation switches support multiple flow tables. This can be used to realize the mechanism mentioned in 1 to separate the rules into two tables. The first table is used by oftables to filter and the other tables can be directly used for a service specific application. With this procedure a matching packet will be directly forwarded with the corresponding action to the next table, before it reaches the drop entry of the first one. The only problem is that still not every OpenFlow hardware switch supports this version.

The oftables filter rules are stored in a database. This raises the question why a separate storage is required, even if the rules are permanently stored in the flow-tables of the switches. The answer is simple and bound to the architectural concept. Other network services or applications are supposed to write flow entries to the flow tables as well, which makes it inalienable, that oftables have to differ exactly which rules are under its own administration. Storing the rules in a database also simplifies the rule administration by multiple oftables instances and enables the multiple operator administration. Moreover, the continuous polling of flow tables is avoided, which again reduces the utilization in the control channel.

4.3 Implementation of oftables

The oftables system architecture consists of two software abstractions layer. On the bottom of this design is the hardware substrate, presented by a network composed out of OpenFlow switches. Above is the Floodlight controller, which builds the first software abstraction layer. Flodlight is using the OpenFlow 1.0 protocol to connect to the switches. The oftables implementation again is connected to the northbound interface of Floodlight using its REST API and the StaticFlowPusher module. An additional database is used to store the deployed rules.

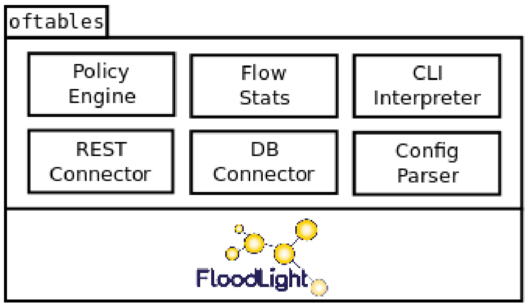

Figure 4.3: Application module diagram

To support a platform comprehensive and independent application runtime environment the implementation is based on the Java Software Development Kit (SDK). Oftables consists of six main modules, as depicted by figure 4.3. These modules are the core implementation and build the basis for this packet filter, as described in sec. 4.2.

- **Config Parser:** The function configuration parser module is to read the local configuration file, which specifies network address of the Floodlight controller and the address of the Structured Query Language (SQL) database. The configuration file also includes the database access credentials, table name and the regarding database type, which specifies concerning java connector.

- **REST Connector:** This module implements a connector for the communication with the REST interface of the controller. Basically, using methods with predefined strings, wrapping method parameter values and sending these strings via socket interface to the Floodlight REST interface.

- **DB Connector:** The database connector is a SQL wrapper class with the opportunity to change the underlying database type by using an other database driver. This module provides all methods for the interaction with the database to store and request filter rules.

- **CLI Interpreter:** The CLI interpreter is the first implementation for a user interface. The interpreter parses the CLI call arguments for oftables internal processing. It can also be used interactively or even in a shell script.

- **Flow Stats:** This module provides a list functionality and displays a statistic output with the number of dropped or accepted packets and the referenced rules.

- **Policy Engine:** The policy engine is the main module which coordinates all in- and output values and delegates their data to the regarding communication module. It combines all application logic and provides a deterministic program behavior.

The following example application calls are demonstrating how oftables can be used as a packet filter. As previously mentioned the parametrization for the program call is nearly the same as it is for iptables. The abbreviations used in this example are $-dp$ for the OpenFlow switch data-path id, which expects an address notation similar to the IPv6 address [32] style. The $-i$ specifies the input interface of the specified switch and the $-nwd$ parameter represents the network destination address. Every filter parameter, as mentioned in section 4.2, has an abbreviation for the filter rule definition which can be listed with the well known $-h$ option. The following fictive examples give an insight how the tool can be parametrized, shown in listing 4.1 and 4.2, and a possibility how an oftables use-case can look like.

- switch seven in the access network has the unused switch-port number 12 in a student room, where every incoming traffic will be dropped with this rule

  Listing 4.1: Example parametrization for oftables

  ```
  oftables -A -dp ::07 -i 12 DROP
  ```

- switch two in the aggregation network gets instructed with this rule to block traffic from department A connected at switch-port three to department B (netmask)

  Listing 4.2: Example parametrization for oftables

  ```
  oftables -A -dp ::02 -i 3 -nwd 10.0.1.1/24 DROP
  ```

4.4 Evaluation

The experimental evaluation is based on the OFELIA TUB island infrastructure. This particular testbed was created in scope of the OFELIA [8] project as introduced in chapter 3. The oftables prototype implementation contains only the CLI parser and the REST Connector module described in section 4.3. This provides an opportunity to evaluate the general distributed filter model and evaluate the general approach as well as the performance behavior with pre-filtered network traffic. Figure 4.4 shows the allocated resources for the evaluation on the afore mentioned testbed.

The samples are collected on dedicated Ethernet links, to provide the full physical throughput capacity at the the VM servers (VMS1 & VMS2)

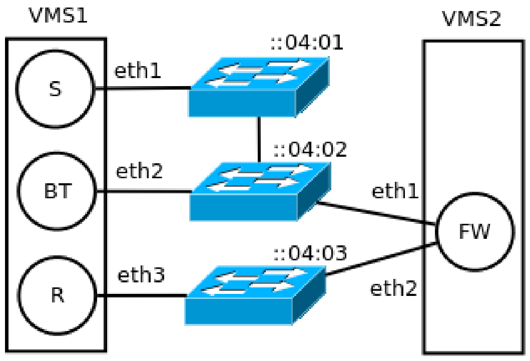

Figure 4.4: Experimental slice setup at the OFELIA TUB Island

interfaces. The experimental setup is further composed out of four VMs distributed over these two servers, using a port and vlan based network slice spanning over three switches. The allocated switches are data-path id ::04:01 until ::04:03. The first server is used to host the following three VMs: a sender (S), the background traffic emitter (BT) and the receiver (R). The firewall VM (FW) is located on the second server. The FW is directly placed as usual in network path and configured with bridged interfaces (eth1 & eth2) without any IP assignment. This configuration forwards any layer two traffic, as long it does not match a filter rule. The collected values are measured with iperf in the User Datagram Protocol (UDP) mode. This configuration was used for the background traffic (BT → R) and also the regular productive traffic (S → R). The data induction rate was 800 MBit per second. All VMs are para-virtualized with the Xen hypervisor.

For having a reference sample an iptables derivate called ebtables was used. This firewall application was installed on the FW VM and finally a filter rule with the source mac address of virtual machine which is injecting the background traffic was assigned, as shown in listing 4.3.

Listing 4.3: Experimental ebtables parametrization

```
ebtables -A FORWARD -s 02:04:00:00:00:57 --protocol IPv4 --ip-
    protocol UDP --ip-destination-port 5001 -j DROP
```

The same filter is now applied with oftables, but with the additional parameter of the switch which takes over the filtering process in this sample, as shown in listing 4.4.

Listing 4.4: Experimental oftables parametrization

```
oftables -dp ::04:02 --smac 02:04::57 --dport 5001 DROP
```

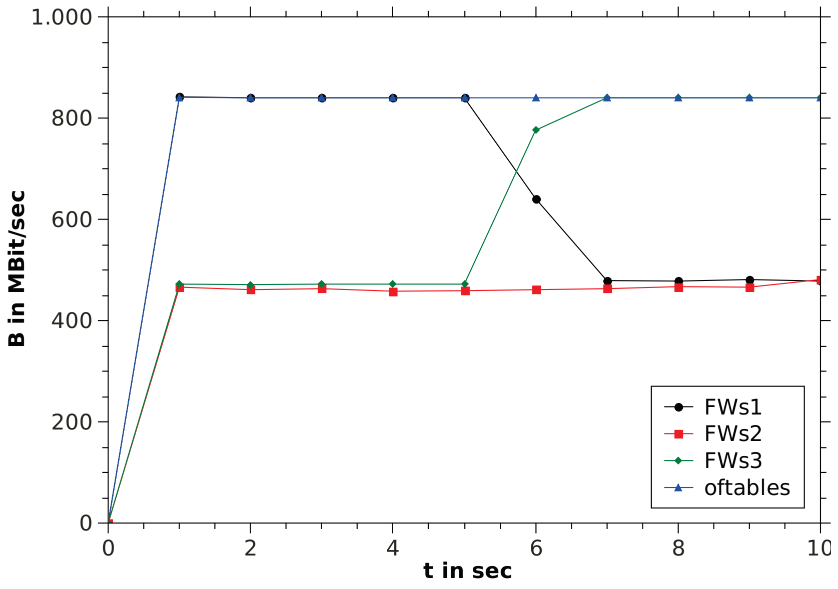

Figure 4.5: Measured transmission rate between sender and receiver VM

In the reference samples, the background traffic is directly filtered on the firewall VM behind the aggregation switch. The oftables samples on the other hand, uses the opportunity of pre-filtering with an OpenFlow switch, so this time the traffic is directly filtered at switch ::04:02 before it enters the firewall. Three measurements were collected for each scenario. The first with background traffic starting in the middle (five seconds) of the regular transmission (FWs1), constant background traffic (FWs2) and background traffic stopping after reaching the middle of regular transmission (FWs3). These measurements are repeated for both scenarios and are depict in fig. 4.5. The oftables sample indicates no further impact to any background traffic interference in contrast to the regular firewall approach. Moreover, in the pre-filtered oftables scenario all three samples had the same outcome – no effect on the regular traffic transmission.

The oftables filter rules were deployed during run-time using Floodlight with the learning switch application for layer two packet forwarding. All three ebtables samples are showing clearly, that the shared bandwidth between the regular traffic and the interfering background traffic ends up in a smaller transfer rate and higher packet drops caused by the output queue. With constant background traffic and ebtables iperf detects 44 % packet drops, compared to the oftables scenario which has only the usual 2.2% packet drops of a regular and not disturbed transmission. The impact on the transmission rate is mainly caused by the shared link to the firewall. The aggregated transmission exceeds the 1 GBit link capacity and causes the intended behavior. Nevertheless, this is an excellent example result which shows clearly the difference of a regular firewall compared to filtering with oftables.

Summarizing the oftables benefits, this application is an innovative firewall which can be used in addition to the established and usual packet filters. It builds a new approach and further point of view regarding packet filters in Ethernet networks. Which is archived by enabling traffic filtering on switches. Furthermore, the filtering process itself is enhanced by using TCAM look-ups and the distribution of the filter-rules across the network infrastructure. This provides a completely new firewall granularity, which evolves the general perspective on perimeter networks. The iptables related management interface provides a well known filter rule administration environment for network operators the application can be used as a central management point by several operators at the same time. Altogether, this application could be powerful new security enhancement tool for local area networks.

Chapter 5

Load Balancing

"Design is not just what it looks like and feels like. Design is how it works."
- Steve Jobs

Contents

Load balancers often have a centralized management and control function in data-centers and ensure the availability and scalability of their services. They basically distribute every incoming service requests to an array of redundant servers to split the workload and provide an appropriate response time to the service consumer. This procedure usually guarantees a fixed minimum amount of response time to serve the clients and provide a proper QoE.

Due to decisive and centralized role load balancers have a disadvantage: they mostly present a single point of failure in the core network. In the worst case this could effect all services and their availability. The problem is usually addressed by complex clustering solutions. These solutions require an often complex administrative competency of the concerning proprietary hardware equipment.

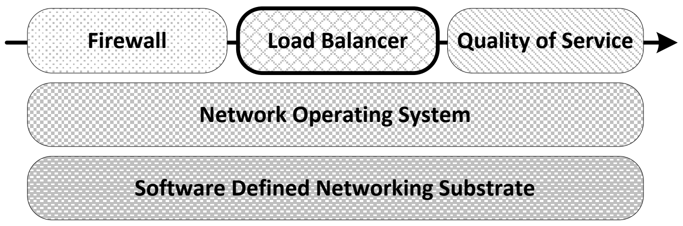

Figure 5.1: Load balancing network service chain link

While current load balancers are mostly implemented and deployed as specific hardware components, also called network appliances, this chapter evaluates several approaches regarding the previously introduced corresponding network service paradigm, as highlighted in figure 5.1. The following sections 5.1, 5.2 and 5.3 are selected and modified citations of the in section 1.2 listed and previously published peer reviewed publications [38, 39] and [40].

5.1 Multiple Services Load Balancing

The contribution of this section is an approach and prototype to process load balancing directly on top of the local network infrastructure as introduced by the network service concept. Therefore, the load balancing strategies are included in the implementation of the regarding network application. The evaluation of this concept is based on multiple controllers running a service depended network application – one controller with the concerned network application per service – that forwards the incoming traffic to the corresponding redundant server cluster.

The distributed and dedicated nature of this concept offers the opportunity to adapt the balancing policies to an a-priori known processing pattern, but also to the current load conditions of the network. Furthermore, failover mechanisms can be implemented easily, as the controllers can handle the management for two or more services, if necessary. The data-center network topology is usually redundant by concept, which also eliminates the physical single point of failure issue. Even more, the controller is able to take the connections and status of the switches and servers into account and can provide features like OSPF, port-channels or the multi-path concept described in section 5.3.2 to improve the workload and network performance. With a network service which covers the load balancing, it is feasible to distribute service requests from a single IP up to a subnet and forward them to the concerned application servers. Another benefit is that OpenFlow provides a native statistic environment based on the flow statistics. This is further to be considered as an additional input for the load balancer logic and balancing algorithm. Additionally, with server agents it would also be possible to adapt the balancing algorithm according to the server workload. All this opportunities deliver a comprehensive load balancer design which considers servers and network utilization for the balancing policies.

To use different network load balancing applications in parallel, a specific separation mechanism is required. This is realized for the following controller application with the OpenFlow network slicing tool FlowVisor[66, 65]. Figure 5.2 shows an example of a visualized sliced network with four slices. Thus, each of the server pairs can handle one service. In the depicted case the servers of a slice are neighbors, but generally the slicing technique allows any possible combination between them. Slices can be defined and configured by using any flow entry matching pattern as specified in the OpenFlow 1.0 specification. FlowVisor is not supporting any recent versions due to technical design restrictions. Still, this mechanism allows a flexible extension of the overall approach: if a particular load balancing service is completely utilized the slice can be extended at runtime. The concept offers the opportunity to use a flexible combination of servers, switches and slices for every load balanced service and the corresponding controller load balancer service. It is further an opportunity to distribute the forwarding utilization from one packet forwarding device to several devices, which improves the performance and also covers reliability aspects.

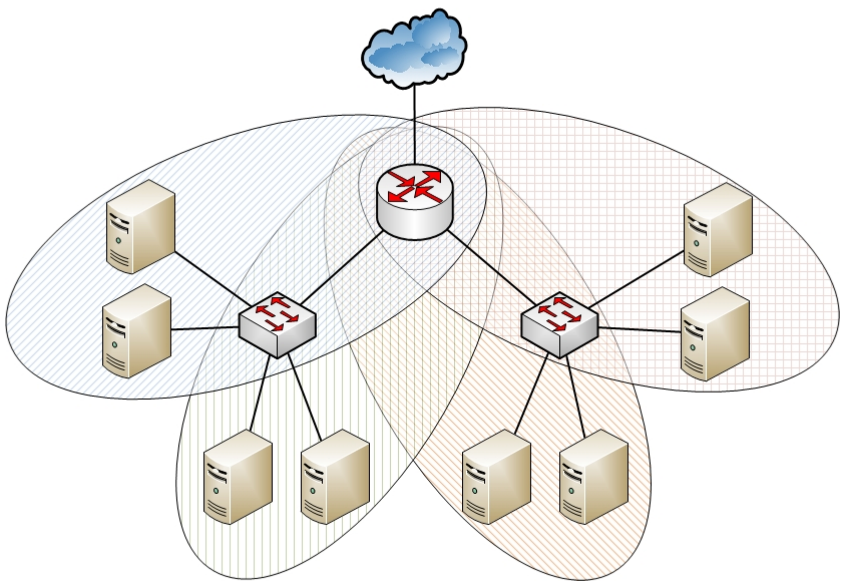
Figure 5.2: Example with four load balancing slices

The actual load balancer network service prototype is using the NOX [31] controller platform. It was developed as a NOX application module and implements a round robin load balancing algorithm. Modules which are supposed to be executed, must be specified in the program call and will be loaded during the NOX startup procedure. The controller again is encapsulated with a network slice and is attached to FlowVisor over the OpenFlow control channel.

If a new service request is detected on the network substrate layer and indicated by an unknown packet due to a client service request, it is passed to the FlowVisor application. Depending on the header information of the packet and its slice policies FlowVisor decides to which controller it must be forwarded. For instance, an incoming packet with the destination port 80 is forwarded to the controller application which handles the HTTP load balancing. In general, a new service request, respectively flow request, is always delegated from FlowVisor to the corresponding controller. The controller again sends the flow entires back to FlowVisor, where they are validated based on the slice policy and forwarded to the corresponding switches belonging to the concerning slice. These flow entries contain the packet header rewrite information for the Network Address Translation (NAT) based balancing procedure, and also the output port for the final packet forward.

Src. IP	Dst. IP	NewDst. IP	Ctl
1.2.3.4:12345	5.6.7.8:80	10.0.0.10:80	A
2.3.4.1:23451	5.6.7.8:80	10.0.0.11:80	A
3.4.1.2:34512	5.6.7.8:21	10.0.1.10:21	B
4.1.2.3:45123	5.6.7.8:80	10.0.0.12:80	A
1.2.4.3:51234	5.6.7.8:21	10.0.1.11:21	B
...	...	...	...

Table 5.1: Incoming packet header rewrite example with the corresponding controller mapping

Src. IP	Dst. IP	NewSrc. IP	Ctl
10.0.0.10:80	1.2.3.4:12345	5.6.7.8:80	A
10.0.0.11:80	2.3.4.1:23451	5.6.7.8:80	A
10.0.1.10:21	3.4.1.2:34512	5.6.7.8:21	B
...	...	...	...

Table 5.2: Outgoing packet header rewrite example

Table 5.1 and table 5.2 are examples for the in both directions processed packet header rewrite. This rewrite is necessary to realize the NAT based load balancing with was briefly introduced with the tables. The tables are also indicating the mapping between the requested service, represented by the regarding Transmission Control Protocol (TCP)/UDP port number, and the corresponding controller. The first rewrite is usually triggered and processed based by a detected new incoming packet, while the second rewrite is processed on the already known and outgoing packet. This process is independent from the physical network part (core/aggregation/access). It can be processed for instance on the ingress/egress switching nodes of the access network but also on the core network router. Clients form the Internet do in any case only notice that all inquiries are processed by one single entity. This is usually the physical load balancer appliance, which is replaced by the introduced load balancing network service in this case.

5.1.1 Load Balancing Network Service Implementation

The load balancer network-application implementation is based on the previously described NOX plug-in. A simple round robin algorithm was used to evaluate the Destination Network Address Translation (DNAT) load balancing performance on the SDN equipment. The plug-in is implemented in C++ and executed on the NOX runtime environment. The implemented plug-in is structured into the following three main parts.

- In the first implementation part all packets are dropped which should not be processed or forwarded by the load balancer, like Link Local Discovery Protocol (LLDP) packets.
- The second part covers the Address Resolution Protocol (ARP) requests. These broad- and unicasts are processed in a special manner, in order to simulate the virtual load balancing device. This is directly covered and implemented by using the *send_openflow_packet* method from the NOX API with an ARP response. This method provides an opportunity to advise the switch to send out a custom defined Ethernet-Frame bit pattern.
- In the third and last part, all traffic between the clients and servers is handled by deploying the flow-entries for a MAC and IP address rewriting of the source and destination address. Therefore a linked list is maintained where all processed client requests are temporarily cached with their source IP, source port, and server destination IP, until the flow entry for the server response is installed in the flow table.

In general, the entire process is established by setting up flow modification entries for a forwarding between a client and one of the servers with a partial replacement of the layer two and three packet header fields. The forwarding and replacement operation is processed with a total of five OpenFlow actions associated with one entry per direction. This entry again, consists of the previously mentioned layer two and three rewrite actions as well as the regarding output action. Eventually all actions are executed by the switch, so that the measured performance has a direct correlation to the switching node and not to the controller. The load balancing prototype uses an idle and permanent hard flow timeout of five seconds. This means that every flow entry is removed from the switch internal flow table if the client server connection is not used for five seconds. This is a best practice value for the performance evaluation following in section 5.1.2. This stands in contrast to a productive environment where the flow entry timeout needs to be adapted depending on the demands of the service to load balance.

5.1.2 Forwarding Performance Evaluation

The experimental evaluation of the implemented prototype is focused on the OpenFlow load balancer performance which is delivered by the packet forwarding elements and the feasibility of the proposed concept. The measured values are collected by using the load balancer NOX load balancing plug-in implementation as described in sec. 5.1.1 for maintenance of the flow entries. All measurements are collected within FlowVisor slices. The samples are processed on the in sec. 3.3 described testbed.

Parameter	Mode	Average Value
Latency	ICMP	2.066 ms
Bandwidth	untagged	7.89 MBit/s
Bandwidth	tagged	6.46 MBit/s

Table 5.3: Measurement of average values in a single slice

The first sample was collected as reference, on a physical port based slice with no interfering traffic. Figure 5.3 shows the measured latency between two virtual machines, which are located on two different physical hosts within the slice. Figure 5.4 again shows the measured bandwidth between these

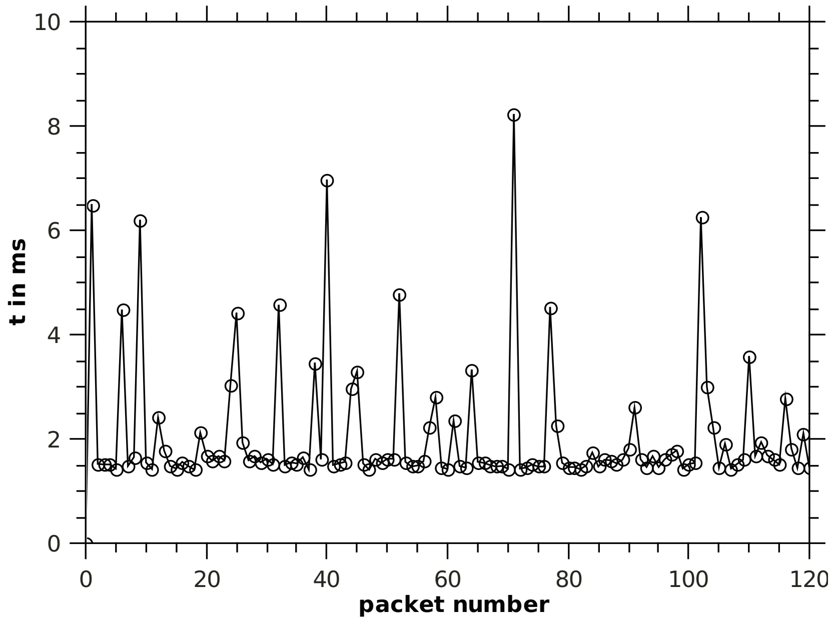

Figure 5.3: Latency

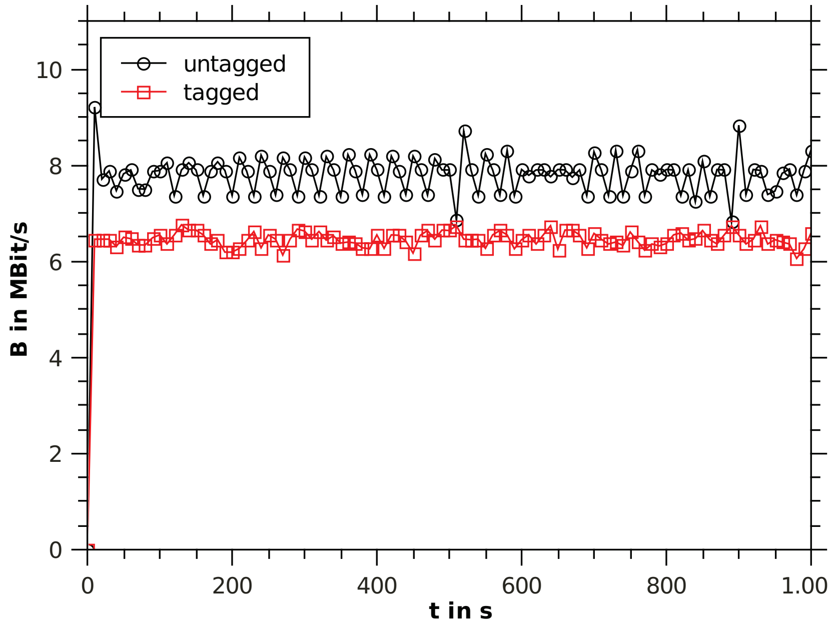

Figure 5.4: Bandwidth of the reference load balancer performance in one slice

endpoints. The latency values were collected with the *ping* application, the regarding Internet Control Message Protocol (ICMP) request, and reply packets. The bandwidth was measured with *iperf*, using a TCP connection setup. The sample values depict in fig. 5.5 are collected to verify the functionality of the proposed model using two load balancers in parallel on two vlan based slices sharing the same ports. The load balancers are basically two instances of the introduced implementation – one per FlowVisor slice.

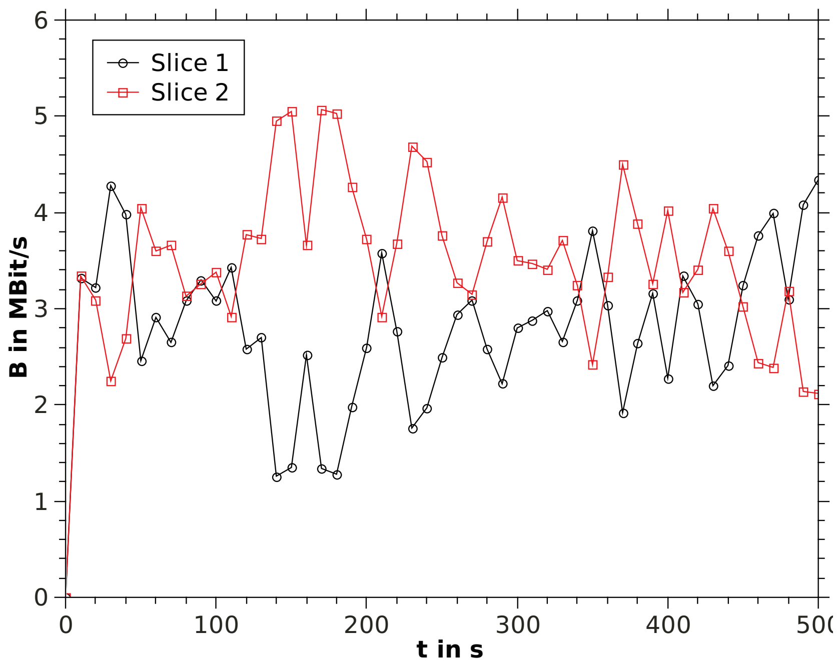

Figure 5.5: Bandwidth in two parallel slices with one load balancing service instance per slice

Slice	Bandwidth
slice1	2.83 MBit/s
slice2	3.53 MBit/s
agg.	6.36 MBit/s

Table 5.4: Measured average bandwidth values with simultaneous load balancing

The measured bandwidth is only around one percent of the overall performance which can be achieved on this particular hardware by using NOX with the regular switching (L2 based forwarding) plug-in. The reason for this is that the used hardware is not capable to process OpenFlow modification actions for OSI layer three in hardware.

As a conclusion this network service implementation is basically able to provides a complete NAT based load balancing, but the achieved performance on the switching hardware is not suitable for the use in a productive environment. Another option is the flat load balancing, which works only with destination NAT. In this procedure the load balancer needs to be the network gateway as well, which is in conflict with the isolated network application approach in this thesis. In general, bringing both network applications together is an opportunity to reduce the rewrite complexity by the four byte long client IP address. Still there is still the layer three action for the sever IP rewrite and the further application has to handle the entire routing as well.

5.2 Load Balancing Performance Optimization

During the evaluation in sec. 5.1.2 it was observed that, the NAT based load balancing method has some major performance issues. The performance restriction belongs to the processed header rewrite operation. Especially network address rewrite operations are currently redirected from the Application Specific Integrated Circuit (ASIC) to the embedded core of the switch, where they are processed in software. This is the identified reason for the performance issues. Moreover, this behavior is currently shown by all OpenFlow enabled switches which were evaluated within the testbed. Switches are typical layer two devices and most of the used ASICs or OpenFlow vendor implementations on the switch are not able to process layer three header manipulations with line-rate. They usually solve this in software on the embedded Central Processing Unit (CPU).

There is a procedure for layer two load balancing which is called Layer 2 Direct Server Return (L2DSR) [20]. The idea of this concept is that a load balancer in the local server network forwards and distributes service requests and packets directly to an array of servers with the same virtual IP address, equal to the load balancer's network address (MAC). The LB solely substitutes the destination MAC address and forwards the packet. This mechanism is called MAC address translation (MAT) and works similar to NAT. The server which is answers the request forwards the packets directly back to the router in the local broadcast domain. This procedure is adapted in the following section in order to improve the performance of the previously introduced load balancing service on switching devices.

5.2.1 Modified Load Balancing Procedure

This section introduces the fundamental idea how to use the existing OpenFlow enabled switch hardware to deploy a full functional load balancer, which is able to deliver line-rate forwarding performance. The focus of this particular approach is a modified load balancing concept, which considers the packet header rewrite opportunities of OpenFlow. As previously mentioned, a load balancing approach working on network layer has not enough performance for a proper load balancing solution as needed by comercial data centers. The further explained optimized model uses a known layer two concept to improve the forwarding performance on the OpenFlow devices. Therefore a L2DSR similar architecture is considered for a slightly modified implementation of the network service, which is setting the forwarding rules for the load balancing.

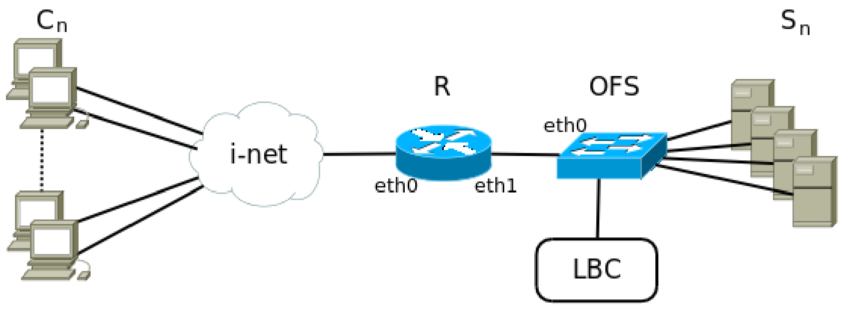

Figure 5.6: Adapted L2DSR load balancing concept

As depicted in figure 5.6 the application servers are directly connected to an OpenFlow switch in a routed data-center broadcast domain. The Open-

Flow controller application, which is labeled with LBC, is responsible for the load balancing and decouples the servers against the rest of the network in order to avoid address conflicts. As described in sec. 5.2, L2DSR works usually with virtual server IPs. This is not needed in this particular approach, because the servers are isolated through the OpenFlow switch or more precisely through the way the network service is advising the switch to handle the traffic. First of all, the modified network service is not directly forwarding any L2 traffic with the "switch". That means, it blocks ARP broadcasts, processes, and answers them directly. Thus, the controller forces the switch to act as a device with an own non-transparent interface. This is covered by handling every kind of address resolution traffic to the legacy network and by advertising the associated MAC and IP of this virtual device. The IP must also be assigned to every server in the server network where the load balanced service is hosted. This technique is the reason for avoiding the source and destination IP rewrites. Table 5.5 explains this procedure, which is also considered in the following paragraph. Internal address resolution requests by the servers are directly answered by the switch, the information of the cached MAT table maintained and stored in the network service application. All MAC broadcasts are not flooded: neither in the server network nor in the data-center network. Thus, layer-three address conflicts due to the same IP addresses of the application servers are prevented. If there is no cached MAT information available, the switch forwards the packet by replacing the MAC address with its own assigned address and answers the request, if it gets the reply from the requested machine. The switch forwards traffic with its associated IP and rewrites the destination MAC address with an address from the server MAC pool. The forwarding MAC address depends on the balancing algorithm and decides finally which server receives the client request. Furthermore the packets of the server reply on the way back to the client are also forwarded with a modified source MAC. The switch solely removes the server source MAC address and replaces it with its own, similar to a network routing operation. This ensures a deterministic L2 addressing in the legacy network and avoids network addressing conflicts. The particular difference compared to routing is that the LB not only separates the broadcast domains, it also forwards the traffic which is appointed to the server array by using a load balancing algorithm for the traffic distribution among these servers with the same network address. The controller has the information of the mapping between the source IP and the destination MAC and uses this knowledge for a state-full packet transfer between client and server due to a fixed flow entry.

#	Src. MAC	Dst. MAC	Src. IP	Dst. IP
1	C_n	R_{eth0}	C_n	OFS_{eth0}
2	R_{eth1}	OFS_{eth0}	C_n	OFS_{eth0}
3	R_{eth1}	S_n	C_n	OFS_{eth0}
4	S_n	R_{eth1}	OFS_{eth0}	C_n
5	OFS_{eth0}	R_{eth1}	OFS_{eth0}	C_n
6	R_{eth0}	C_n	OFS_{eth0}	C_n

Table 5.5: L2 header modifications

The following example will give a better understanding of this packet processing and forwarding procedure. Table 5.5 shows all manipulations concerning the packet header on the way through the example network, as depict in fig. 5.6. In particular, the described header modifications are reaching from the packet arrival at the router, over the implemented load balancer, up to the server where the requested service is hosted and vice versa:

1. An arriving packet with the IP of the load balancer – in this particular case the IP, which is associated with the OpenFlow switch by its LB network service – is forwarded into the local network.

2. Therefore, the router replaces the source and the destination address in the layer-two header of the packet. It is the usual routing operation which every router is using. The network destination address of the packet is the OpenFlow switch (OFS) with his emulated IP and MAC address.

3. The switch forwards the packet to one of the servers in the server network array and replaces the destination MAC address again. The chosen MAC address and of the corresponding server depends on the

MAC/server pool as well as on the selected load balancing algorithm. The destination IP is not replaced, because both IP addresses are identical. This means that S_n and OFS_{eth0} use identical IP addresses in order to avoid the layer three rewrite.

4. Also the source IP is not modified because the server directly replies to the requesting device.

5. The OFS forwards the packet and replaces the source MAC address. It is again changed to the OFS MAC.

6. Finally, the router is forwarding the packet by a standard routing operation to the client in the Internet

With this procedure it is required to handle the described load balancing as well as the redundant server array with a black box model. This means, there is no communication between the server nodes or within the legacy network. It is essential that no ARP traffic leaves the OpenFlow switch, otherwise it will cause a layer two address conflict. In contrast, the benefit is that every processing node in the LB server array is an identical replication of a certain image template and that is totally encapsulated. The only difference is the layer two address of the physical host.

5.2.2 Adaptation of the Service Implementation

The implementation is generally based on the previously introduced NOX load balancing plug-in by using some modified OpenFlow actions and a MAT address list instead of a NAT list. It has no dependencies to other plug-ins and is designed as stand-alone network service application. The plug-in is written in C/C++ and uses the packet modification architecture and model, which was introduced in section 5.2.1.

The implementation is separated into two different main parts. The first part is dealing with the ARP requests and replies and forces the switch to react on any kind of ARP message. This means the switch interface is forced to act as an independent device. Processing ARP lookups is a slow operation, because every interaction needs to be processed and approved by the load balancing network service. In contrast, the rewrite and forwarding actions with the manipulation of the L2 address are processed by OpenFlow actions executed on the switch. These are comparatively fast operations, because the controller approves the flow for the entire data-exchange only once. Every further following packet which matches the FlowMod-Entry is directly modified and forwarded by the network service without additional approval. The actual forwarding entries of this approach consist of a matching pattern and two OpenFlow actions. The first action is the manipulation assignment for the destination MAC address. The second action is the output action which causes the forwarding of the packet.

Basically, the crucial functions of this implementation are responsible for the ARP mapping and processing, pushing the L2 rewrite FlowMods and handling the load balancing. Some additional functions are implemented and used to management and maintain the LB entries and the concerned ARP mappings. The method which is responsible for the LB matches every incoming new packet except LLDP or ARP. If a new incoming flow is detected, the networks service directly applies the flow rules with the rewrite information and deploys them as a flow entries on the switch. It is works the same way as explained in section 5.1.1. The used algorithm maps a server MAC address depending on the requesting source IP of the client and stores this information in the internal LB mapping table. The entry is removed, if the corresponding flow expire event from the switch is received.

5.2.3 Performance Comparison

This section is about the comparison of the forwarding performance within the investigated approaches. The evaluation of the former presented DNAT measurement results are directly compared to the results which were collected with the modified MAT network service. Therefore, the data-path forwarding performance for both approaches is directly compared to each other regarding latency and bandwidth, as shown in figure 5.7 and 5.8.

Besides the flow entry installation time, at the beginning of every trans-

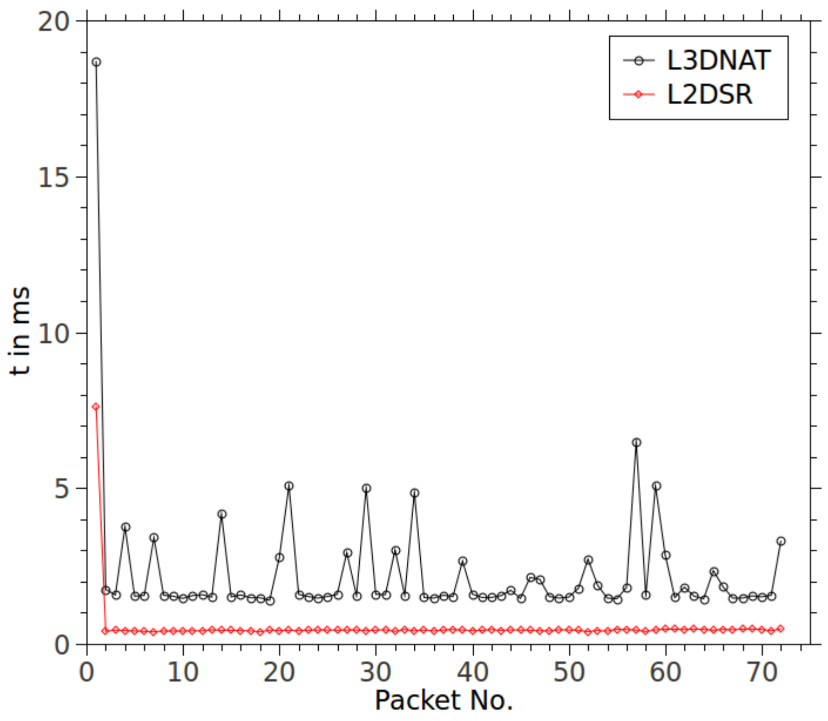

Figure 5.7: Latency

mission, the revised approach presents a considerable performance gain. The obtained performance gain is achieved due to the avoidance of any layer three rewrite operations as presented in section 5.1.1. The optimized approach with the a MAT based load balancing mechanism only uses layer two packet header rewrite modifications which are completely supported and processed on the ASIC. This is also indicated through the decreased CPU utilization of the embedded core. The embedded core load was closely to idle, in contrast to the previously measured NAT based implementation where it reached a utilization of approximatly 95 percent. This reflects the estimated behavior of a typical layer two forwarding hardware which is in this particular case a switch.

The values presented in figure 5.7 and 5.8 were again collected with the *iperf* and *ping* application in the OFELIA testbed described in section 3.1. They are measured between the client and server VM's of VMS1 and VMS2

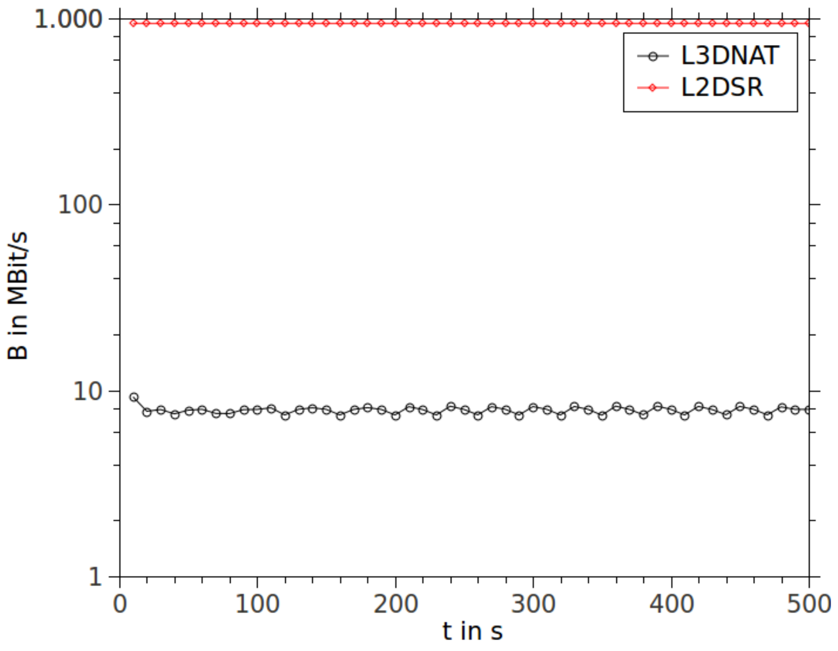

Figure 5.8: Throughtput

	L3DNAT	L2DSR
Latency:	2.066 ms	0.445 ms
Throughput:	6.46 MBit/s	939 MBit/s

Table 5.6: Measurement average values

to evaluate the forwarding performance of the two load balancing approaches based on the regarding OpenFlow enabled hardware of the testbed. In both diagrams the black series shows the outcome of the DNAT implementation while the red chart presents the revised layer-two implementation.

As shown in figure 5.7 and also summarized in table 5.6, the latency was reduced to a quarter of the previously taken NAT sample. Moreover, the Round-Trip Time (RTT) scatter was reduced to a minimum of 0.2 ms with a standard deviation of 27 ns, while TCP throughput was increased up to 939 MBit per second. This means an overall increased throughput of 145 times and a latency decrease to an average of about 0.5 ms. The only exception is the flow entry set up time which was measured for the first packet with a RTT value of 7.62 ms. Altogether, this revised approach is capable to realize load balancing on OpenFlow switches as used in the testbed.

5.3 Multipath for Broadcast Domains

Nowadays a certain data-centers network topology is the so called fat-tree topology, where the Top of Rack (ToR) switches use a redundant connection to the aggregation network and connect at least two aggregation switches in order to provide reliability, and thus, redundancy for the inter rack and core network traffic. This concept with an increasing interconnection capacity to the upper network layers is usually deployed to avoid bottlenecks and to have fail-over mechanisms if an aggregation switch fails. The physically redundant connections to different aggregation switches can be used to realize a concept called "rack to rack multi-path" which is introduced in this section. It goes slightly beyond the current state of the art in this area. It basically describes a procedure for transferring the realization concept for port channels to the redundant wired aggregation network. It is a very basic SDN network service approach for a typical data-center use-case without getting network operators fear loops or nondeterministic network behaviors.

5.3.1 State of the Art

Section 5.1 showed the opportunities which are offered by OpenFlow regarding load balancing. These opportunities were used to create a load balancer network service, which uses regular OpenFlow capable switches in order to distribute client requests directly to an array of redundant servers. The introduced approach is a typical data-center use-case for a virtual network appliance using OpenFlow. It used a procedure to forward destination NAT manipulated packets from the ToR switch to the physical nodes and vice versa. The approach has some major performance issues due to the not in hardware supported OpenFlow L3 rewrite actions. In 5.2 a revised concept was presented which uses a MAT based approach and which is able to deliver nearly line-rate. In this section the previously presented load balancing approaches are extended by a concept for the next higher intermediate network topology layer, with the aim to optimize throughput between different nodes.

Previous scientific explorations with similar topics uses for instance an OpenFlow software emulated network ring topology with Mininet [75] and a particular IP based path separation, as described in [30]. The in [45] proposed approach is similar to the one presented in section 5.3.2, but the presented evaluation just uses a software emulated topology. Others approaches again, use Multipath TCP (MPTCP) [79] in combination with OpenFlow to enhance the data transfer rate in a traffic engineered network. Another interesting area to enhance the available network throughput and reduce packet drops is the L2 multipath approach, which is a special form of load balancing. The multipath opportunities introduced by the Transparent Interconnection of Lots of Links (TRILL) standard [57] and described in [56] by using different VLAN tags to separate and share the load are considered. This approach is very similar, the redundant wiring between the ToR switch and the aggregation network is used to distribute different flows over redundant physical available network connections. In contrast, the following approach deals with real OpenFlow equipment as it appears in data-centers. It is focused on the evaluation of the feasibility and the achievable performance in such an environment by using an OpenFlow multipath scenario.

5.3.2 Rack-to-Rack Multipath for OpenFlow Networks

This section introduces the actual OpenFlow multipath approach. It works similar to TRILL approach but does not require an additional packet header encapsulation, because OpenFlow already provides all required capabilities due to its out-band control concept. Nevertheless, the SDN based solution which is proposed in this section, also requires a mechanism to separate and to distinguish the flows. This is a requirement, but OpenFlow provides the opportunity to use any header field which is defined for the flow entry packet matching process introduced by the concerning OpenFlow standard and section 2.1.1. If the network hardware supports OpenFlow version 1.0 every packet header field from the Ethernet header till the transport protocol port can be used for the separation process. Moreover, it completely differs regarding to the flexibility of flow separation patterns. While one flow is e.g. identified by its IP another flow can be identified and separated by the transport protocol. This can be changed on demand for every flow and the respective application purpose. This separation concept provides an innovative granularity for the traffic separation compared to the afore mentioned concepts like TRILL or Multiple Spanning Tree Protocol (MSTP) based VLAN mechanisms.

The flow concepts which are supported by OpenFlow are building the basis for the network application presented in this section. The separation opportunity which was previously introduced on a flow based concept is now used to differ and distribute the inter rack traffic between several ToR switches. Therefore, different routes are used between the redundant wiring of the involved aggregation switches in a pod. This means, that all flows which differ, can also be distributed between all available routes from any source to any destination inside or between a pod[1].

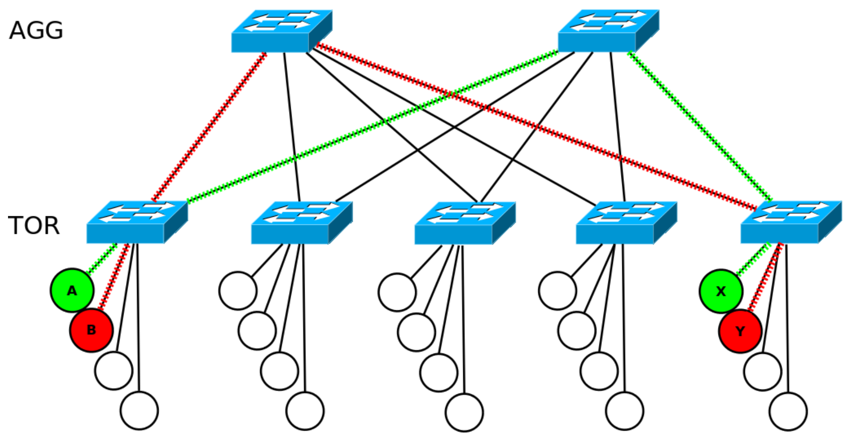

Figure 5.9: Basic load balancing example with two flows on two different paths

This network service application is able to enhance the data transferrate between the access- and aggregation- or aggregation- and core-layer.

[1]A pod is a block of racks grouped by ToR and aggregation switches which are connected to the core network as a unit

It is especially appropriate for data exchange or more precisely data I/O intensive applications, catchword big-data use-cases. For instance, with an infrastructure manager like OpenStack[54] and the multipath network service combination, a comprehensive VM and network scheduler can be used to find an optimized solution for data I/O intensive VM communication and their distribution. In general, this is also limited by the amount of inter rack flows and physical links between e.g. the ToR and aggregation switches. There is no way to completely avoid traffic congestions, but at least the proposed cloud middle-ware composition is able to reduce this effect and distribute the traffic with a smart and accurate VM placement and network multipath approach. To avoid flow based traffic congestions caused by communicating hosts, as shown in figure 5.11, the infrastructure manager should use opportunity to optimize network performance by placing VMs on a physical host node with an unused network path, as depict by fig. 5.10.

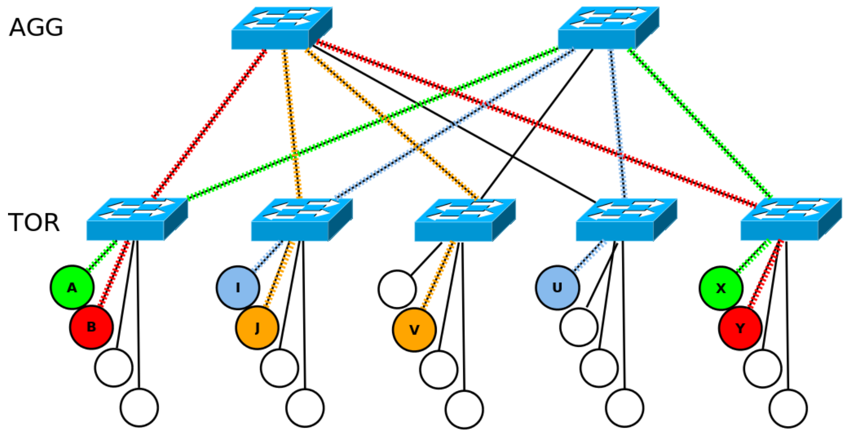

Figure 5.10: Example with four flows using four different paths

5.3.3 Measurements

The section is supposed to validate the proposed architecture and to investigate the feasibility regarding performance aspects in a multipath scenario for data-centers. Therefore, measurements were performed on a topology which is similar to the one depict by figure 5.9. The measured values were collected on the OFELIA TUB island. According to the use-case depict in figure 5.9, the

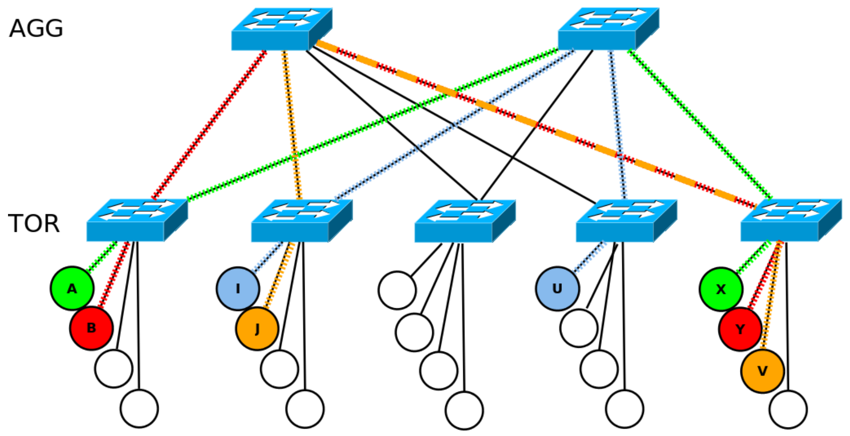

Figure 5.11: Flow overlay example with four flows using four different paths in one overlay

evaluation was processed on the testbed and used two access switches (NEC1 and NEC2) and two aggregation switches (NEC3 and NEC4). Furthermore, four VMs were booked (two on each server) to evaluate the simultaneous data transmission between each pair. This setup offers the opportunity for the simultaneous end-to-end traffic transfer over two different flow paths as previously described in section 5.3.2.

$$A \rightarrow NEC1 \rightarrow NEC3 \rightarrow NEC2 \rightarrow X \tag{5.1}$$
$$X \rightarrow NEC2 \rightarrow NEC3 \rightarrow NEC1 \rightarrow A \tag{5.2}$$
$$B \rightarrow NEC1 \rightarrow NEC4 \rightarrow NEC2 \rightarrow Y \tag{5.3}$$
$$Y \rightarrow NEC2 \rightarrow NEC4 \rightarrow NEC1 \rightarrow B \tag{5.4}$$

Therefore, two bidirectional flow paths between the four VMs were deployed. Each flow path with a dedicated physical Ethernet interface mapping ($A_{VMS1:eth1}$; $B_{VMS2:eth2}$; $X_{VMS2:eth2}$; $Y_{VMS2:eth1}$) for every VM. A fifth VM was used for the OpenFlow Floodlight [49] controller installation in version v.0.9, including the Static Flow Pusher module. This module was used to pre-install the flow entries by using a proactive flow installation installation. The flow entry installation was executed for all the involved switches. The deployed flow entries were installed by using the REST API from the Static Flow Pusher module and a self written *bash* shell script for batching *curl* based HTTP REST calls. The installed flows for the data transmission and the regarding reply are described in equation 5.1 till 5.4. All flows and the resulting flow-entries create a dedicated physical link based connection between both VM pairs. Therefore, a total amount of 12 flow entries is required - two paths over three devices in both directions.

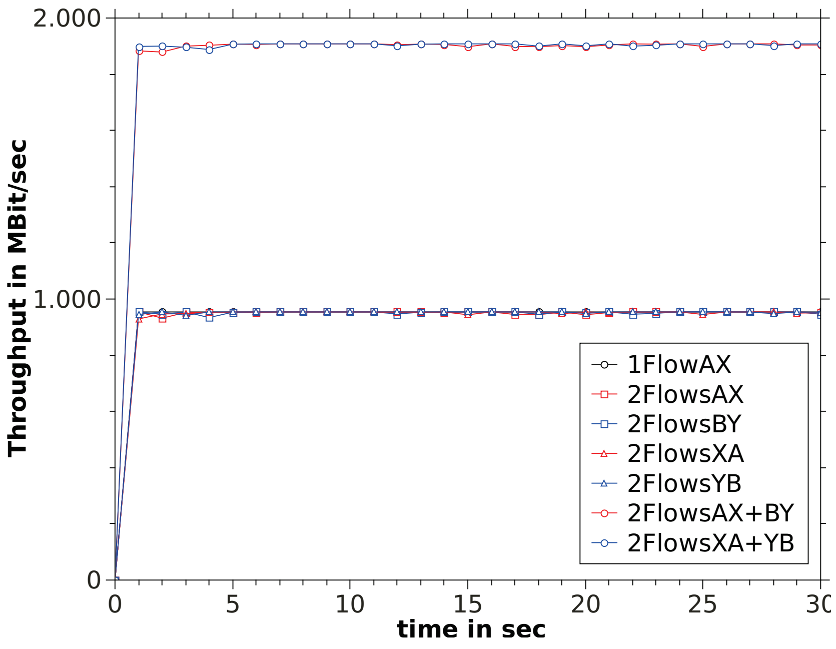

Figure 5.12: Measured throughput for a single and the aggregated simultaneous flows

The values were measured with *iperf* and the detailed parameters are listed in tab. 5.7. With this parametrization throughput values of around 950 MBit/sec. were achived per flow for the regarding simultaneous transmission between $A \rightarrow X$ and $B \rightarrow Y$, as depict in figure 5.12. This correlates with an aggregated throughput of approximate 1.9 GBit/sec. All five collected samples had a similar outcome, so a selected representative sample was used to visualize the outcome.

While a single transmission over one flow path passes the network without

Parameter	Value
Protocol	UDP
Data length	1450 Byte
Bandwidth	1GBit/sec
Duration	30 sec.
Report Interval	1 sec.

Table 5.7: Iperf parameters

any packet loss, using both paths at the same time ends up in a minor packet loss. The measured arithmetical mean value was for $A \rightarrow X = 0.24\%$ and for $B \rightarrow Y = 0.11\%$. The time series of the measured packet loss is depict in figure 5.13. The reason for this behavior is not clear due to the closed nature of these proprietary switching devices, but might be caused by a queue schedule issue which is a usual reason for packet drops.

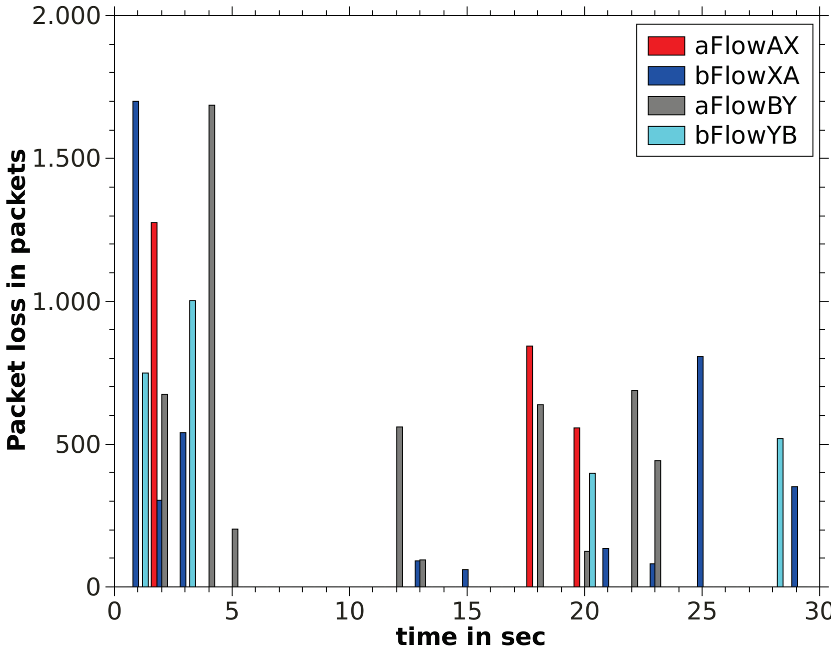

Figure 5.13: Measured packet loss

Full duplex mode is an other opportunity to increase the data transfer rate over the aggregation network. So the same samples were processed again, but this time in duplex mode. Therefore, the same end-to-end connection between $A \rightleftharpoons X$ and $B \rightleftharpoons Y$ was used. These are basically two flows between both VM pairs as described in equation 5.5 and 5.6. The measured values depict in figure 5.14 indicate that also full duplex works without any noticeable interference effects on the throughput rate.

$$(A \rightleftharpoons X) \Leftrightarrow \begin{cases} A \rightarrow X \\ X \rightarrow A \end{cases} \tag{5.5}$$

$$(B \rightleftharpoons Y) \Leftrightarrow \begin{cases} B \rightarrow Y \\ Y \rightarrow B \end{cases} \tag{5.6}$$

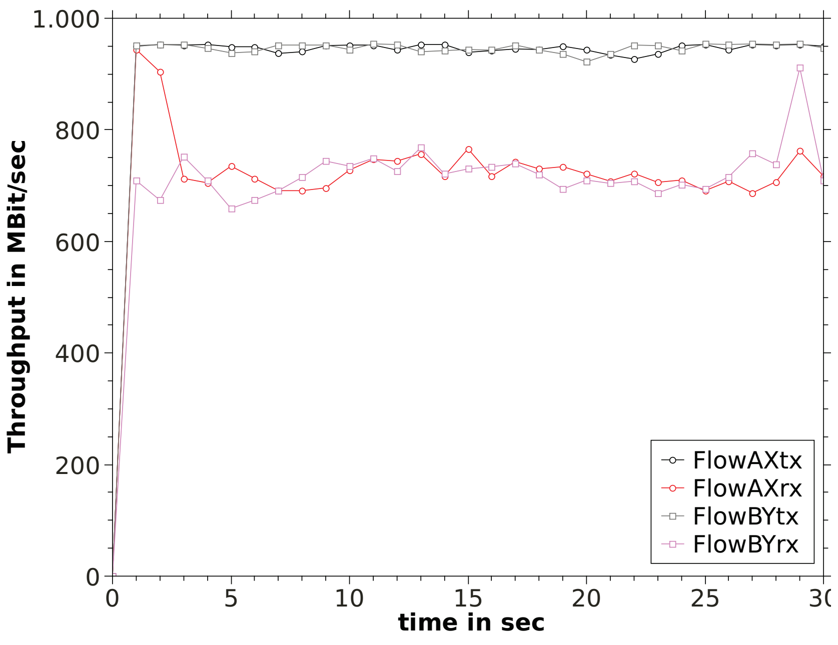

Figure 5.14: Measured throughput for both flows simultaneously in duplex transfer mode

Finally, latency measurements with the RTT were performed to determine any system delays. Therefore, two scenarios are investigated. In the first scenario, ICMP requests are sent from $A \rightarrow X$ and from $B \rightarrow Y$. The measured RTT values do not show any significant difference while a single

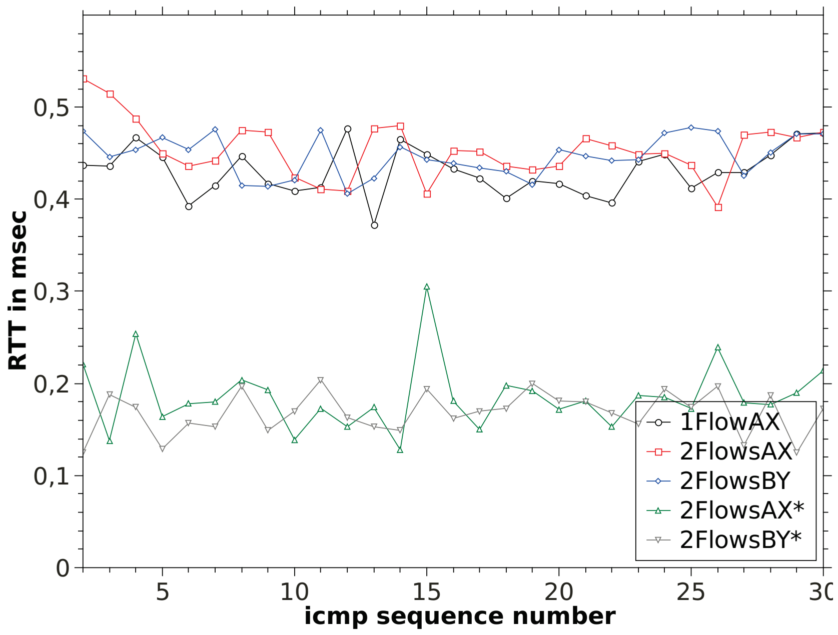

Figure 5.15: Measured RTT between VMs

flow is used or during a simultaneous transmission on both flow paths. In the second scenario the measurements were repeated in the same way, but with one difference: the other flow path was used to inject *iperf* generated traffic as described in tab. 5.7. The outcome of the second scenario is also depict in figure 5.15 and labeled with 2FlowsAX* and 2FlowsBY*. The additional traffic load in the second flow path reduces the measured RTT values. The reason might be an energy saving or sleep mode of the ASIC while it is under low load conditions. Nevertheless, more investigations in this direction are needed to finally clarify and determine this behavior, which is not part of this work.

This multipath evaluation section has shown that the presented approach is resulting in a clear performance gain for the rack-to-rack data exchange between different end points. The remaining question is how to distribute the flows and VMs to avoid congestion due to flow interference as shown by figure 5.11. In particular, the open issue is to find a suitable algorithm to distribute the flows in an overlay of x flows from y hosts through z physical network paths with c available capacity, so $f(x, y, z(c))$.

In spite of the promising results more investigations regarding the scalability and complexity in usual data-center L2 domains and the concerning amount of devices are required. Nevertheless, this OpenFlow network service provides an additional optimization opportunity as demonstrated. The redundant network infrastructure provides additional resources which were used to enhance the network performance in the regular operation mode. Furthermore, this approach comes without the need of additional port-channel or MSTP based VLAN encapsulations. The entire mechanism is a straight forward realization on the opportunities the OpenFlow based SDN technology offers. The evaluation is based on real switching hardware and shows clearly that this is a reasonable and feasible approach for common fat-tree networks.

Chapter 6

Quality of Service

"I never think of the future - it comes soon enough."
- Albert Einstein

Contents

More and more services and applications like Voice over IP (VoIP) rely on QoS aspects. QoS is actually a growing topic and enabler for several new applications. Especially OpenFlow and its opportunities for an application specific dynamic on-demand network QoS adjustment and support are very promising regarding the expected impact for this sector, e.g. bandwidth on demand.

This chapter introduces concepts and approaches to realize and cover certain QoS aspects in OpenFlow networks. In particular, it will introduce and describe approaches for the last link in the presented network service chain, also highlighted by figure 6.1.

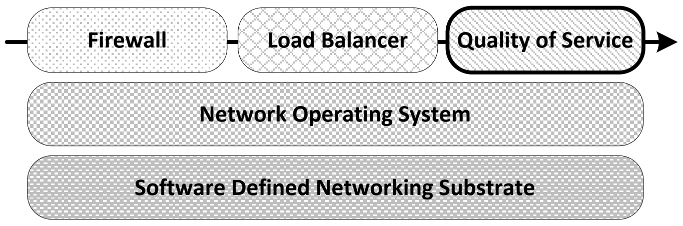

Figure 6.1: Quality of Service network service chain link

The following sections are about approaches for QoS in OpenFlow networks and the concerned experimental evaluation. Moreover, the chapter is closed by an approach for the OpenFlow control plan scalability, which is a basic requirement to realize QoS network services. The content presented in the regarding sections are slightly modified and selected citations of the in chapter 1 section 1.2 listed publications [43, 42, 70, 69] and [36].

6.1 OpenFlow Network Overlays

The broad range of Infrastructure as a Service (IaaS) products offers the opportunity to outsource or manage infrastructures as a cloud. In this process, customers or users are able to request several VMs and configure them according to their requirements. Nowadays, also network services are offered which allow customers to group VMs in Virtual Local Area Networks (VLAN) [82, 81]. This enables logically isolated private clouds where the infrastructure provider delivers the resources and the customer has complete control over the entire virtual networking environment. However, the network services are offered without QoS parameter and do not respect the current utilization of the service provider's underlying physical infrastructure. Also the infrastructure provider is not able to optimize or adapt the utilization of its infrastructure depending on the current available capacities. The flexibility of IaaS is currently very limited when it comes to network resources. For instance, the topology, routing policies and QoS characteristics between VMs in the cloud [43].

The OpenFlow technology enables this innovative mechanism to control exactly the networks used in this area [78, 25]. Moreover, it provides the opportunity to manage the network and the concerning capacity itself as a service. In particular, configuration and behavior of the SDN can be controlled by one entity and changed on demand. The properties of the underlying network and its connected servers can therefore be treated as a resource. However, the full range of SDN capabilities is not utilized in today's infrastructure managers. Furthermore, SDN is already uses to facilitate future data center networks, but the benefits are often not passed to the customer.

6.1.1 Quality of Service via Ethernet

Realizing QoS constrained data flows in Ethernet networks with guarantees regarding timing, latency or bandwidth was covered by several approaches, e.g. Differentiated Services Codepoint (DSCP) with Type of Service (ToS) field or VLAN based Class of Service (CoS) with the Priority Code Point (PCP) field [17, 51]. These are common representatives which exploit a QoS based priority mechanism, CoS for OSI layer 2 and DSCP for layer 3, to reduce possible delays in switching devices [18]. These approaches require decentralized set-up procedures, sometimes special hardware, and deliver only soft probabilistic guarantees. These guarantees are suitable for e.g. Voice over IP applications, but may fail in conjunction with requirements regarding soft real time or even deterministic behavior [68]. Enabling Ethernet networks to match these requirements and bridging the gap to usually used field-bus networks, was a challenge especially treated by the industrial automation domain, leading to several Real Time Ethernet (RTE) solutions. In order to gain real time behavior on top of Ethernet networks, the non-deterministic CSMA/CD MAC protocol used in Ethernet has to be bypassed. This can be achieved through usage of full-duplex switched Ethernet. However, RTE solutions still have to handle non-deterministic delay introduced in switching devices due to busy ports. Several approaches exist to overcome this issue

[26]. However, the achieved mitigation of delays do not cover the requirements of field-bus domains and are heavily influenced by non real time nodes participating in the same network. Moreover, having only eight CoS state values, limits the number of possible gradations and may fail in case of many QoS enforced data flows and complex end-to-end overlays.

6.1.2 Opportunities with OpenFlow

The major challenges for a QoS approach are to minimize non-deterministic delays in switching devices and allow a dynamic and centralized management of QoS parameters with a fine-grained distinction between the data flows. These QoS constrained data flows should be dynamically deployable within a network. OpenFlow is exactly the technology which enables opportunities to implement a network service which is covers these requirements. Due to the detailed information about all data flows in the OpenFlow network, the network service can make assumptions about the switching device workload and therefore may reduce the afore mentioned delays. One of the recently appeared OpenFlow specifications [53] (v1.3) provides basically two direct mechanisms to enforce QoS with OpenFlow:

1. **Meter tables** consist of meter entries, which are used to define simple QoS constraints per flow or respectively flow entry. These entries have one or more rate limits and a type field, which describes how to further process the flow, if the corresponding allocated rate is exceeded, e.g. drop packets.

2. **Port queues** provide an additional mechanism to implement more complex QoS constraints. They are part of the packet-scheduler and can be combined with meter entries. These queues, also depict in figure 6.2, are attached to physical ports and can be configured with QoS parameters, e.g. minimum guaranteed rate or queue length. Flow entries which are mapped to this queue will enforce to be processed accordingly to the queue configuration.

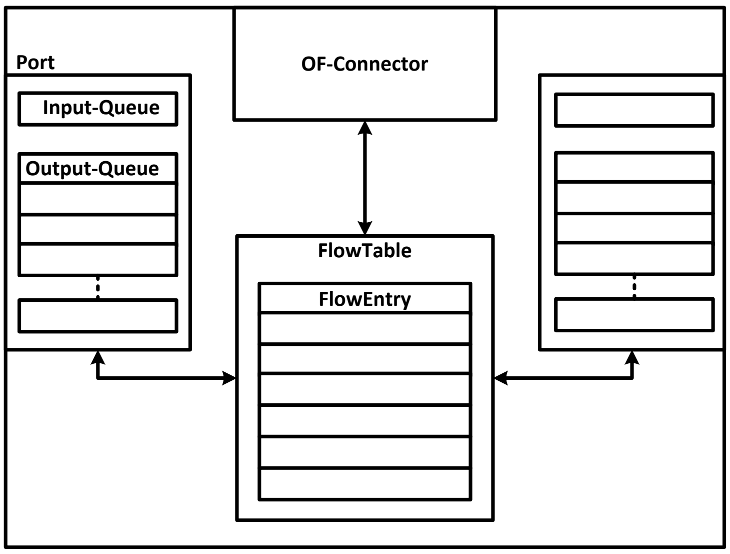

Figure 6.2: OpenFlow switch block diagram

These mechanisms give the opportunity to realize a network comprehensive QoS overlay, by distributing the total available bandwidth over several flow and queue bands. This means an end-to-end connection with its particular QoS constraints, sharing the same physical connection with others. A usual procedure for this strategy is for example an aggregation of subbands with a ten percent idle overhead to ensure the promised bandwidth and quality, as depict in figure 6.3.

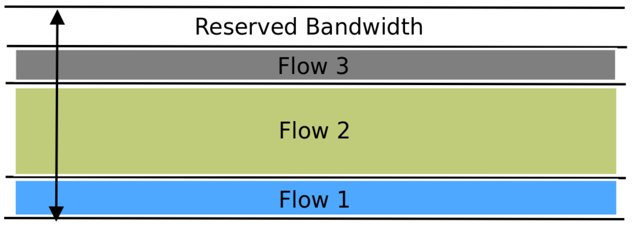

Figure 6.3: Flow overlay example

Requirements which cannot be achieved have to be rejected during the QoS negotiation or e.g. hand-it-over to another redundant wired connection, as introduced in section 5.3.2. In addition, OpenFlow can also influence the processing behavior of other legacy forwarding elements in the network path, by modifying the common layer two PCP or layer three ToS fields in the packet header, as described by Other opportunities to implement QoS would be to rewrite the ToS field when forwarding packets similar to DiffServ [9]. Adding information this to the flow entries attached rewrite action is further an opportunity to integrate legacy devices into the central QoS network service control concept. The pre-configured PCP can be used in combination with a dynamically rewritten packet header at the SDN network edge to enforce QoS on a legacy device or network part.

6.1.3 QoS aware Network Service Architecture

To cover QoS management requirements as they appear in infrastructure management software like OpenStack [54] the network service was designed with a web-service based API. This API is connected to a business logic for example and provides an high level booking system for the regarding QoS overlay connection in the network. It offers the opportunity to the business logic to monitor and book QoS constraint network resources on a flow basis. This business logic which is used for the later presented evaluation was an Service-Level-Agreement (SLA) negotiation frontend which will be treated as black-box system and is not further explained in the following sections. The presented *SLA4SDN* system consists of two separate software parts: the SLA frontend and the SDN backend. The backend again and the mechanism to enforce QoS will be further explained now.

Frontend-Backend Interconnection

The web service interface between the SLA frontend and the SDN backend provides three main functionalities: *discoverRoutes*, *checkRoute*, and *establishRoute*. These operations are described in detail below and are illustrated in the overall architecture, shown in figure 6.4.

discoverRoutes This operation retrieves all possible routes between two nodes. Therefore, this operation requires two IP addresses as parameters

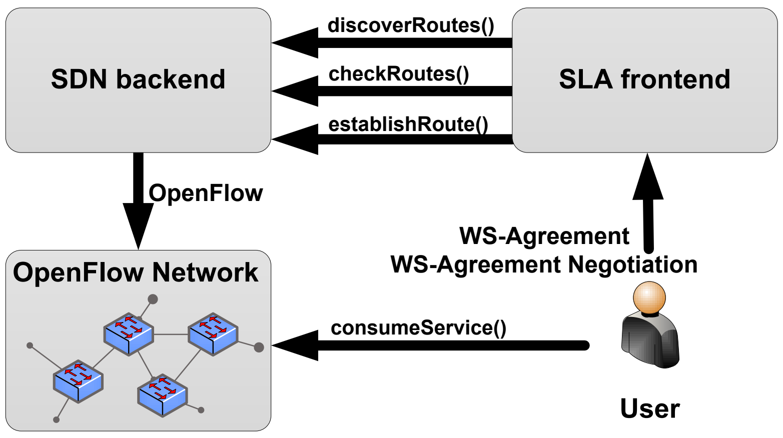

Figure 6.4: SLA4SDN architecture and interfaces

which are the two endpoints between all discovered routes are. All possible routes are returned with an unique identifier and a maximum bandwidth.

checkRoute This operation checks the current utilization or from another poit of view the available capacity of a route. This operation is required to ensure that the available capacity has not changed between the discovery process and the establishment of a route. Therefore, the unique identifier is passed as parameter by calling the *checkRoute* operation and as a result the available bandwidth is returned.

establishRoute This operation establishes the route between two nodes in the network with an advertised QoS. In particular, this operation requires the selected route identifier, the minimum bandwidth, the maximum bandwidth, and the expiration date as parameters. In order to map the bandwidth boundaries to a SLA, the maximum bandwidth is the user's request and the minimum bandwidth is the service provider's guarantee. If the expiration of a service is reached, the route will be deleted.

SDN backend implementation

When the SDN backend web service *discoverRoutes* is called by the SLA frontend, the backend probes the network to find all available flow paths from host A to host B. Afterwards, the available bandwidth for every identified path is retrieved. Therefore, the maximum available bandwidth on the participating links is subtracted by the already booked connections, so that a potential bottleneck can be identified. These results are sent to the SLA frontend in order to provide the required utilization of the network substrate to negotiate the QoS level for the virtual network overlay. After these properties are successfully negotiated and an agreement is completed, the SLA frontend sends these regarding properties to the backend to finally book the concerning connection. On each switch along this path the flow entries are required to build this connection are installed. Thus, the exclusive layer two route for the regarding user is assembled. In order to realize a QoS traffic shaping, OpenFlow 1.0 provides an enqueue action. Flows can process an incoming packet by enqueue it in a specific output queue which is attached to a physical port, as depict in figure 6.2. While information of these queues can be queried via OpenFlow, the configuration requires a separate management interaction over a CLI or NetConf [10]. These output queues provide throughput limiters with a maximum and minimum rate. Configuring this rate usually requires the configuration of the vendor dependent management interface, which is not accessible by design in the common SDN real hardware based OpenFlow testbeds like OFELIA [8] or GENI [6]. Therefore, it was decided to use the software switch implementation called Open vSwitch (OvS) [12] for the evaluation of this particular approach as presented in section 6.1.4. The described implementation uses the Floodlight controller and its internal Java API. The key components and structure of the backend are depict in figure 6.5 and described in detail in the following paragraphs.

FrontendConnector In order to receive requests from the SLA frontend, the *FrontendConnector* exposes three web services and thus provides an interface to the *CircuitControl* module. The benefit of this design is, that the

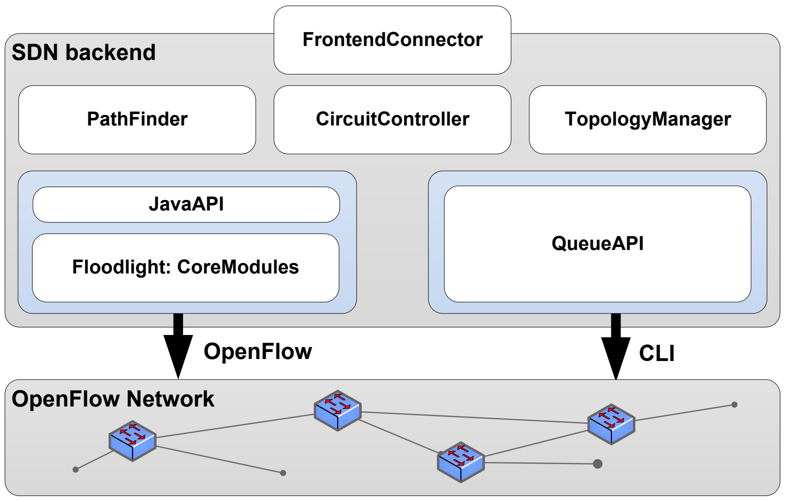

Figure 6.5: SDN backend components

SLA frontend and the SDN backend are completely decoupled and can be distributed and scaled over several machines.

CircuitControl This is the core module of the SDN backend. It exposes methods to obtain information about the network and methods to adjust it. This module is directly called and wrapped by the *FrontendConnector*. The *SLA4SDN* system is currently limited in its functionalities concerning the offering and establishment of connections with a particular QoS bandwidth capability and guarantee. Side tasks of this module are to find layer two routes between the given end-points and to establishing the regarding paths. It has a generic design and other extensions (e.g. query and adjust network properties), which can be easily extended in this module along with the additional modules which then perform these tasks.

PathFinder This module is to find available and appropriated paths. It directly interacts with the *CircuitControl* to calculate these paths depending on the network topology. Using depth-first search the PathFinder determines all paths between two arbitrary nodes in the network.

TopologyManager The TopologyManager keeps track of the general state of the network. Therefore, it uses the Floodlight API to register a listener function to get notified about any changes related to the switches, ports, links and other topology relevant information. In addition, this module also stores all information about already booked routes, associated queues and their properties, the installed flow-entries, and the already allocated bandwidths in the network. Based on this information, the *TopologyManager* is also in charge of calculating the remaining bandwidth for all paths discovered by the *PathFinder*.

QueueAPI This interface module invokes a CLI to the Open vSwitch *ovs-vsctl* control tool in order to create, configure and modify the concerning rate limiters which are assigned to a particular queue. These rate limiters are building the substrate dependent bandwidth enforcement mechanism in the presented system. Depending on the switches, the network is composed of, the *QueueAPI* module itself can be replaced by another version containing the vendor specific queue configuration commands.

6.1.4 End-to-End QoS Overlay Evaluation

The presented results are focused on the basic verification of the deployed overlay connection and the validation of the concerned bandwidth boundaries. The SLA4SDN prototype and the concerning dynamically deployed QoS constraints are evaluated on an OvS based OpenFlow network using the Mininet v. 2.0.0 [11] environment. The maximum throughput capabilities of the emulated Mininet network are directly related to the underlying computer hardware. The presented performance measurements were processed on a laptop equipped with an AMD E-350 1.6 GHz CPU, further one of the two physical available cores was exclusively assigned to the Ubuntu Linux Mininet VM by using VirtualBox as a hypervisor.

The emulated network itself is composed out of six hosts, labeled with $H1$ to $H6$ and two switches, $SW1$ and $SW2$. The hosts are combined into two groups with three hosts each, which are connected to an aggregation switch

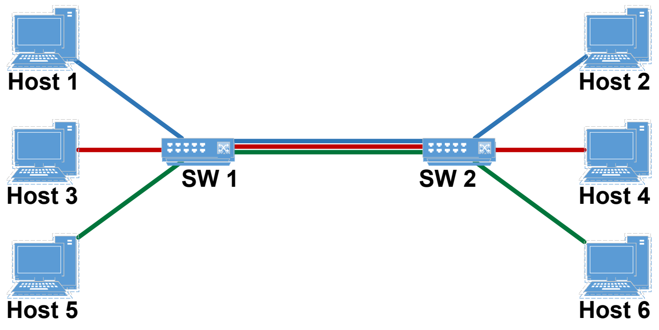

Figure 6.6: Network architecture

on each side. The aggregation switches again are directly connected to each other, as depict in figure 6.6. The focus of the evaluation is the behavior of the shared connection in between the switches and its ability to fulfill the specified throughput.

This is processed in two scenarios with two flows ($H1 \rightleftharpoons H2$ and $H3 \rightleftharpoons H4$) with narrow-band throughput (T) guarantees pretended by the frontend and enforced by the concerning queue configuration, as described in equation 6.1 and 6.2. The third flow ($H5 \rightleftharpoons H6$) is used to emulate the background traffic and utilize the network to maximum capacity. In the first scenario the background traffic is processed by flow-entries using the regular OpenFlow "OUTPUT" action. In contrast, in the second scenario an OpenFlow "ENQUEUE" based on an additional queue forwards also the background traffic. This third queue with a generously configured lower and upper rate limiter, as described in equation 6.3, is used instead of the default queue and takes over its function.

$$(H1 \rightleftharpoons H2) = \left\{ \begin{array}{l} T_{min} = 8MBit/s \\ T_{max} = 10MBit/s \end{array} \right\} \tag{6.1}$$

$$(H3 \rightleftharpoons H4) = \left\{ \begin{array}{l} T_{min} = 20MBit/s \\ T_{max} = 25MBit/s \end{array} \right\} \tag{6.2}$$

All three connections are measured simultaneously, to verify that the requested bandwidth is not violated. However, the transmission is started with an offset for each joining connection, to explore the effects and influences on the overlay in the shared aggregation part. Therefore, the bandwidth between every host-pair is measured with iperf in TCP mode. As previously mentioned, the first two connections are deployed with a particular bandwidth rate, while the third one is used to simulate the background traffic and to utilize the network to almost maximum capacity. This means, that the third transmission without an accurate throughput guarantee will be affected when the other transmissions are starting. All samples are collected over five minutes. The flows with bandwidth guarantees are joining with one and two minutes offset and a total transmission duration of two minutes.

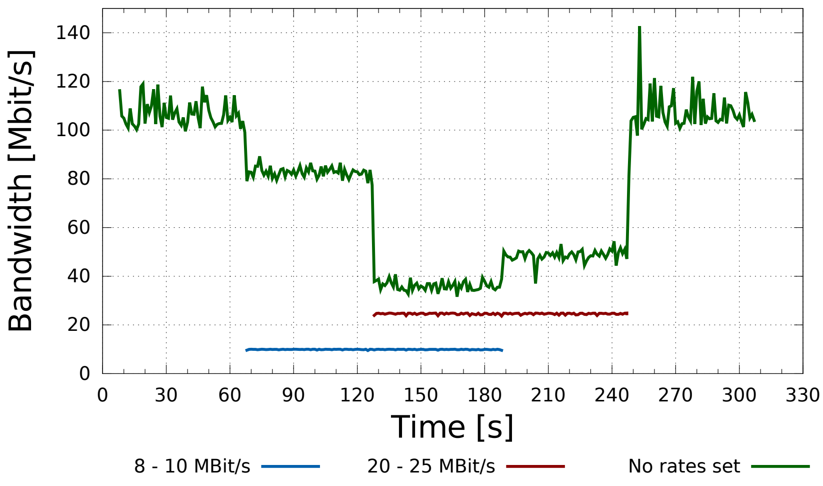

Figure 6.7: Throughput measurement - Queues are configured for two flows

The outcomes of both samples are depicted in figure 6.7 and 6.8. Figure 6.7 shows the measured throughput values of all three connections for the first scenario. In this scenario two queues with rate limiters are used to enforce QoS for the $H1 \rightleftharpoons H2$ (blue) and the $H3 \rightleftharpoons H4$ (red) connection. The last connection between $H5 \rightleftharpoons H6$ (green) again, is simulating the previously mentioned background traffic by using best-effort forwarding. This transmission ($H5 \rightleftharpoons H6$) utilizes the network with the maximum available capacity, which is roughly 120 Mbit/s. After 60 seconds, when the second transmission ($H1 \rightleftharpoons H2$) starts, the throughput of the best-effort connection drops immediately. The sample is showing a similar behavior when the third transmission ($H3 \rightleftharpoons H4$) starts. It is evident that the QoS bandwidth requirements of the second and third transmission are fulfilled during the entire sample time. In contrast, the best-effort traffic is automatically adjusted by the Open vSwitches in order to ensure the allocated bandwidth for the other transmissions. The samples of the second scenario are showing the same behavior, but a slightly lower variation compared to the best-effort background traffic measurements in the first scenario. Nevertheless, the samples are clearly indicating that the requested bandwidth for the transmissions are fulfilled in both scenarios.

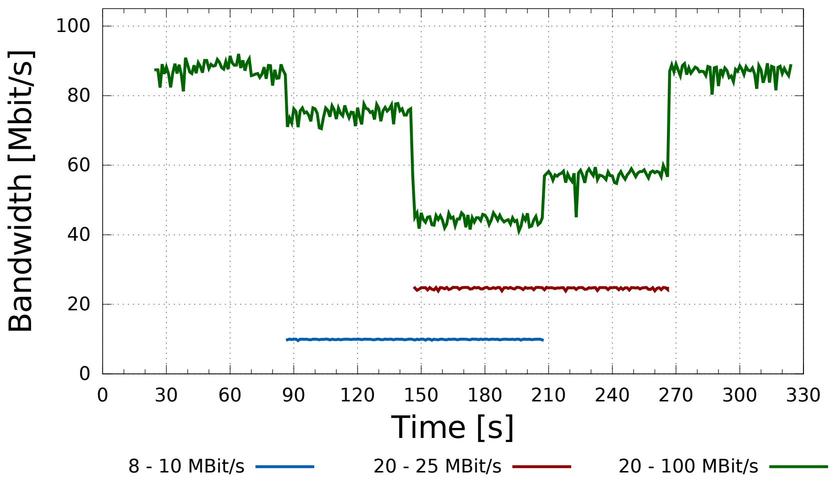

Figure 6.8: Throughput measurement - Queues are configured for all flows

$$(H5 \rightleftharpoons H6) = \left\{ \begin{array}{l} T_{min} = 20MBit/s \\ T_{max} = 100MBit/s \end{array} \right\} \tag{6.3}$$

Summarizing the outcome of the conducted measurements, the evaluation was successful in terms of demonstrating that the presented approach is able to ensure a certain bandwidth for an end-to-end connection on a shared network interconnection. Upon a thorough examination of the charts, it becomes clear that the background traffic which is handled by the default queue has some drawbacks. First of all, the simulated best-effort traffic in the first scenario has an identifiable higher throughput variation. Furthermore, it is also showing a disproportionate throughput drop compared to the additional utilized throughput of the joining flow. For instance, when the transmission $H1 \rightleftharpoons H2$ also depict in figure 6.7 starts, it is observed that the background traffic throughput drops from approximately 110 Mbit/s to 80 Mbit/s, although the additional throughput amount consumed by the joining transmission is only 10 Mbit/s.

In the second scenario, when an extra background traffic queue with a broad bandwidth range between the minimum and maximum rate limiter was used, a slightly more deterministic behavior concerning the throughput aggregation is achieved. As depict in figure 6.8, the maximum utilization from the background traffic has slightly decreased, but the variation is reduced to a minimum compared to the sample of the first scenario. Moreover, the drop of background traffic when the other transmissions, $H1 \rightleftharpoons H2$ and $H3 \rightleftharpoons H4$, are starting is nearly proportional to the consumed bandwidth of the additional transmissions. This makes it much easier to manage and calculate the overall network utilization and predict the available capacity in relation to all end-to-end overlays.

Concluding this particular QoS based evaluation, the second scenario is delivering a more reliable behavior, by explicitly configuring and using queues for all connections even if they have not met any particular QoS bandwidth requirement. On the other hand, the maximum overall network utilization in this sample decreased basically from 110 Mbit/s to around 90 Mbit/s.

6.2 Control-Plane Scalability

A completely different approach to scale OpenFlow networks and provide QoS is considered in this section. Software defined networks usually separate the control plane from the data plane of the network devices. Typically, the network control is a centralized piece of software running on a dedicated server. On large networks, consisting of thousands of active network devices, the performance of the central network control is expected to be an issue.

To overcome this problem and avoid congestions in the control palne the centralized approach must be extended to a distributed control plane design.

The concept proposed in this section addresses and solves this problem at least theoretically by taking the scalability issue in general into account. Therefore, the introduced solution avoids any restrictions regarding the design and the use of the software defined network. In principle, the application does not require any changes of existing protocols; it is supposed to be as transparent as possible and supports recursive instances. The principle behind the following introduced concept, is the separation of large networks into an appropriate number of smaller networks i.e. network aggregates. The basic idea is that each of those networks will have its own network controller. This network controller again will provide additional functionalities which allow the controlled network to be represented as a single virtualized network device to the next higher hierarchical layer. To summarize, a network with its ingress and egress links is represented as a virtual network device, e.g. an OpenFlow switch, with the concerning number of input and output ports.

The first hierarchical layer therefore consists of a number of those virtualized network devices, each representing its small underlying network aggregate. The interconnection of the ports of the network devices is represented by the ingress and egress links between the small networks on the underlying layer. Because every presentation of a network on layer n is a virtual switch on layer $n + 1$, this layer behaves just like a regular network.

6.2.1 Adapted Concepts

The proposed hierarchical concept is based on designs which were introduced by the developers of the hierarchical PNNI [76], defined and implemented as part of the Asynchronous Transfer Mode (ATM) network technology. Numerous performance evaluations, e.g. [58, 21] elaborated the advantages of this hierarchical concept. Regarding scalability, Cisco for example states in its PNNI Reference Guide [72] that for today's very large scale or future networks a hierarchy of up to three till four layers could be sufficient to cover the entire network management.

The hierarchical approach results in a significant reduction in topology information since network controllers outside a sub-network only see other networks represented as single switches. Each network controller on layer n is using detailed topology information concerning its own sub-network. Further details are not required, because the end-to-end connectivity setup is done based on the virtualized switches on the layers above. Based on this, it can be expected – but of course has to be proven – that our approach will be a suitable solution to avoid scalability driven performance issues regarding the network controller bottleneck in large scaled SDN topologies.

Based on the advantages of the SDN approach, the realization of a similar hierarchical layering in OpenFlow [46] networks is by far less complex compared with PNNI. The main benefit shown in the following parts is the separated control plane. The instance involved in the layering is just the controller, which has to be partly extended. In contrast to ATM, there is no need for a so-called peer group leader selection process.

A good example for a virtualization mechanism based on OF is FlowVisor (FV) [65, 66]. This application is a transparent policy aware OpenFlow proxy sever. It enables network slicing and provides the opportunity to use a controller per slice. ADVisor [64] and VeRTIGO [24] are implementations based on FV and extend the application with an additional virtualization layer, in terms of building virtual topologies on the physical substrate. Both tools provide a special form of slicing by exposing the switches or parts of the switches to a controller application. Moreover, the approach in [77] is about a multi-controller network, where the controllers are synchronizing each other to improve the scalability and availability of the regarding sub-network.

Many benchmarking approaches [34, 63] focus on the controller performance, which is measured with the established flows or outstanding packets compared to the network size by the number of switches. This suggests that the entire approach of a hierarchical management layer is also interesting in terms of controller performance gains through a reduced amount of devices to be managed by a sub-network similar separation.

6.2.2 Hierarchical Controller Extension Approach

OpenFlow generally assumes a central network management with the complete and deterministic knowledge of the network topology. This is a major point in the concept to provide an intelligent forwarding, related to the service requirements or the network topology. Intelligent in this context means a specific flow based forwarding with an appropriated path. This generally differs from legacy switched forwarding, where every forwarding element is only learning its local path assignments, which is a particular form of flow switching. Nevertheless, with increasing network size the flow management of a single controller will reach its limit due to the rising amount of new flows and the regarding processing requests from the switches to the controller. This fact indicates the need for a multiple controller approach, as presented in sec. 6.2.1, to cover the management of large scale or multi-tenant SDN networks.

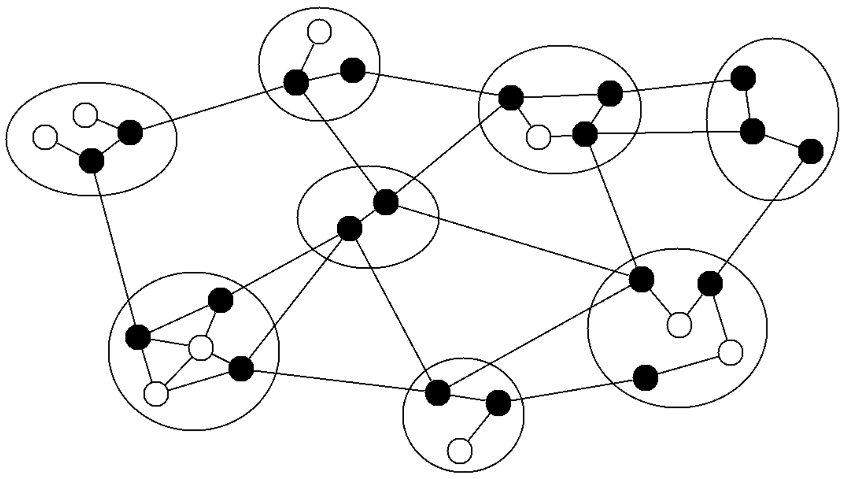

Figure 6.9: Example for two layers of HLA datapaths

The hierarchical control layer approach, which is presented here, will go beyond any of the former mentioned solutions and introduce a new point of view for the management of multi-tenant OF networks and their interaction. This concept is about a non-transparent presentation of sub-networks as switches. This mechanism presents a special type of network topology aggregation. This means in detail, that the controller of the next higher layer

$n+1$ will notice a specific sub-network as a single OF switch. The switch ports which are exposed to this controller are a summarized collection of the edge-switch-ports from the concerning sub-network with connection to an other network or provider, which is under control of the corresponding layer $n+1$ controller. Figure 6.9 depicts a typical PNNI related network architecture which is directly adopted to the Hierarchical Layer Application (HLA) approach. Every circle is a packet forwarding node and the black filled circles are representing the ingress and egress nodes, also described with δ_{edge} in the equations, with links to other networks. They are surrounded by the network borders which are encapsulated by an HLA controller and can be again considered as switching node in the next higher layer. This is a theoretical aspect of the model and can be done on every higher or lower layer.

A brief mathematical consideration of the concept will reflect that the proposed application is able to reduce the management complexity with a growing number of hierarchical and accordingly vertical layers. The following equations use the parameter δ for a number of switches in the regarding sub-network. Sub-network in this context does not necessarily mean an IP sub-net, it general describes a composition of OF switches in a closed management-unit or domain. The iteration parameter i is used for the hierarchical layer n and j for the regarding sub-net m in a layer, with $\{\delta, i, j, n, m\} \in \mathbb{N}^+$. To compare this particular concept, the following equations build the number of switches to be managed in total δ_{all}, by FV δ_{FV} in an inter-domain approach and with the HLA δ_{HLA} proposal.

$$\delta_{all} = \sum_{i=1}^{n} \sum_{j=1}^{m} \delta(i,j) \tag{6.4}$$

$$\delta_{FV} = \sum_{i=1}^{n} \sum_{j=1}^{m} \delta_{edge}(i,j) \tag{6.5}$$

$$\delta_{HLA} = \sum_{i=n}^{n} \delta(i) = \delta(n) \tag{6.6}$$

$$\Rightarrow \delta_{all} \geq \delta_{FV} \geq \delta_{HLA} \tag{6.7}$$

In conclusion of equation 6.7, the presented management approach clearly shows that HLA significantly decreases the number of devices to be managed for the next higher layer. Only in the worst case scenario they are equal, but the usual case will be an exponential decreasing behavior of devices to be managed per abstraction layer compared to the FV approach. In contrast, the HLA concept performs with a constant amount of devices and sub-networks per layer. Nevertheless, it has to be emphasized that FV traces and presents a completely different approach. This thesis is not focused on denigrating FV, this example was just chosen due to its versatility and the ability to reproduce a similar behavior.

To get a better understanding of the concept, the following brief real world scenario is explained under the assumption of a completely SDN enabled and comprehensive network including the required extensions for the optical backbone network part. Usually, universities have several departments. This is the first layer in this example. In every department the regarding administrator has control over his network and can also provide department specific requirements or topologies it only exposes the links to the core network by using the proposed application. The interconnects between the departments and the additional forwarding hardware in the core network is managed by the Network Operations Center (NOC) of the university, which is the second abstraction layer. If a host in the access network contacts another host in another university, the national research and education network is used. The regarding provider processes the forwarding in the same way as it was explained for the universities, but in a much larger scale. This is the third abstraction layer, similar to the architecture depicted by fig. 6.10. This process can be continued in the same way.

The introduced concept presents the usual procedure for today's network management. Every sub-network, or in a lager scale every provider, has fixed handover points where the corresponding network management unit takes over the control for the further traffic forwarding. Every local network is

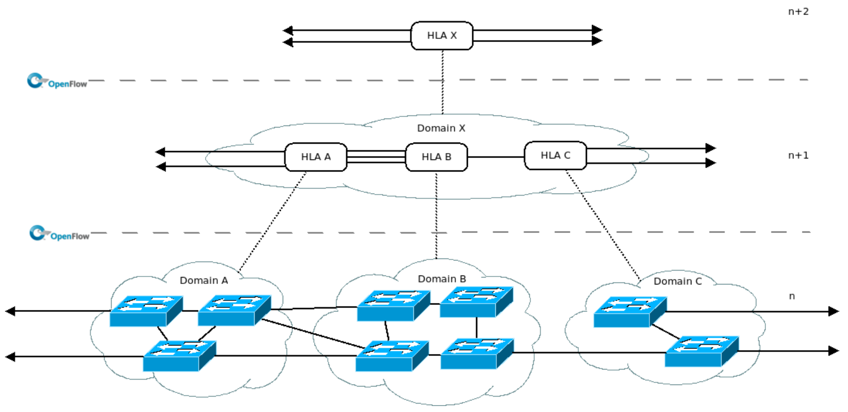

Figure 6.10: HLA OpenFlow network example with three layers

under control of the concerned network operators and is not exposed to the provider. This is exactly the key idea which provides the opportunity to build an hierarchical management application by keeping the SDN paradigm. The management handover point in this approach is the HLA. The data-path handover is given through the connected links of the network. The presented application is a network controller with an additional northbound OpenFlow API. This additional interface is used to emulate an OpenFlow switch, which hides the internal network topology and exposes only the external links in form of its interfaces of a single switch. This completely encapsulates the network and makes the forwarding path to an internally managed process. Again, the provider can deal with the same concept to expose the control for its links to the next higher institution and so on. This approach basically provides a concept for reducing management complexity and tenant controlled network size of SDN comprehensive substrates. Furthermore, the processing in the data-path of an aggregate is under control of the local network controller. In contrast the control between the aggregates is done by the next higher layer controller and can for example use a completely different routing strategy.

An other useful side effect of this control plane design is that the namespace, in this particular case the data-path IDs, used in a sub-network, can overlap. Moreover, the provider or management instance of the next higher layer can declare data-path ids to use, similar to the internet provider associated IP address for example, without influencing sub-net internal assignments. The proposed model has no negative effects concerning the SDN paradigm in general. All characteristics of SDN and the offered opportunities can be realized. The only exception is that the new elementary unit is now a single aggregate which isolates the control plane size in the aggregate. The aggregate itself is not restricted in its size, but in order to avoid problems due to the growing scale, the proposed concept is also a good candidate to be considered for building smaller structures. In conclusion, a sliced network architecture created with a tool like FV is now restricted to an sub-network but can still be used to create network slice in it.

Chapter 7

Conclusion

"Nothing endures but change."
- Heraclitus

Contents

This chapter concludes this thesis and provides a summary with the classification of the herewith achieved outcomes. It will further briefly describe the limitations of the presented approach. Finally, it closes with a short perspective with regards to the introduced data-center network services.

7.1 Summary

This thesis introduced new and innovative concepts to realize the usual ingress data-center network service chain. Therefore, the typical data-center network appliances like firewall, load balancer, and QoS packet forwarding were redesigned and transferred to OpenFlow network applications. The services were revised according to the SDN paradigm. Thus, services were further designed and realized by considering the NFV concept. All the conceptual developed services are evaluated on an innovative OpenFlow testbed which was especially created for such purposes. The presented overall approach provides a dynamic scalability of these services while unifying and decoupling the management interface using the SDN technology.

The state of the art of the presented network appliances was investigated, revised, and transferred to the SDN based network function approach. Thus, new concepts, methods, and the opportunities provided by the SDN technology were considered to address and overcome restrictions of the currently predominating deployments. Therefore, an innovative SDN research testbed was created and further used to evaluate the proposed and developed network services and applied methods. The entire OFELIA testbed was composed of a federation of different OpenFlow network testbeds from international partners. The testbed provided researches access to heterogeneous SDN and computational resources by using the slice based federation architecture approach. The experiences which were conducted in cooperation with the partners with regards to the technology, methods, architecture, and limitations built the basis for the design of the developed new network services. The introduced firewall network service approach was revised with a completely distributed filtering approach which is administrated from a decentralized management interface. A common DNAT load balancer network service was evaluated with the new technology and improved by using a MAT based balancing method distributing the requests to the redundantly deployed application servers. Moreover, a method for multipath flow switching in Ethernet based broadcast networks was presented to increase the overall throughput in redundant wired network parts. The proposed QoS forwarding approach is applying a global end-to-end overlay connection between hosts which were using a dynamically controlled enqueue mechanism in order to create a real time Ethernet similar network behavior. The further presented PNNI OpenFlow network OS extension provides a model to increase the control channel scalability. The developed network services, approaches, and results were published in a total of 14 scientific papers.

The presented outcomes are restricted, because the scalability was basically not considered due to different limitations of the experimental setup, as for instance the testbed size. This is a weak point, since all presented network services are basically under high utilization in productive data centers environments. Nevertheless, the presented work is evaluating the general opportunity by using the SDN technology in order to deploy these virtual network appliances as network services in data-center networks. The proposed concepts and evaluation results are demonstrating the feasibility based on today's OpenFlow enabled physical network equipment.

7.2 Future Work

The presented network service concepts and the concerned realizations are a first step into a dynamic data-center NSC paradigm. With the raising performance and specific ASICs designed which directly supports OpenFlow actions in hardware, it will be possible to overcome the currently investigated performance limitations. Moreover, also the OpenFlow specification progress and the increasing amount of functions will further provide more opportunities to address the identified QoS difficulties. Altogether, this thesis indicates that the introduced methods for data-center NSC have the potential to revolutionize the resource management and administration in this data center network area.

Acronyms

ACL	Access Control List
AM	Aggregate Manger
API	Application Interface
ARP	Address Resolution Protocol
ASIC	Application Specific Integrated Circuit
ATM	Asynchronous Transfer Mode
BOWL	Berlin Open Wireless Lab
BSD	Berkeley Software Distribution
CLI	Command Line Interface
CoS	Class of Service
CPU	Central Processing Unit
CSMA/CD	Carrier Sence Multiple Access with Collision Detection
CTL	User Control Network
DMZ	Demilitarized Zone
DNAT	Destination Network Address Translation
DSCP	Differentiated Services Codepoint
ECML	Experiment Control and Management Layer
EU	European Union
EXP	Experimental Network
FIB	Forwarding Information Base

FV	FlowVisor
FW	Firewall
GEC	GENI Engineering Conference
GENI	Global Environment for Network Innovations
HAL	Hardware Abstraction Layer
HLA	Hierarchical Layer Application
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
IaaS	Infrastructure as a Service
ICMP	Internet Control Message Protocol
ICN	Information Centric Networking
IP	Internet Protocol
ISO	International Organization for Standardization
L2DSR	Layer 2 Direct Server Return
LB	Load Balancer
LLDP	Link Local Discovery Protocol
MAC	Media Access Control
MAT	MAC address translation
MGMT	Management Network
MPTCP	Multipath TCP
MSTP	Multiple Spanning Tree Protocol
NAT	Network Address Translation
NFV	Network Function Virtualization
NOC	Network Operations Center
NSC	Network Service Chaining
OCF	OFELIA Control Framework

OF	OpenFlow
OFELIA	Openflow in Europe: Linking Infrastructure and Applications
OS	Operating System
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
OvS	Open vSwitch
PCP	Priority Code Point
PNNI	Private Network-to-Network Interface
QoE	Quality of Experience
QoS	Quality of Service
REST	Representational State Transfer
RML	Resource Management Layer
RTE	Real Time Ethernet
RTT	Round-Trip Time
SAN	Storage Area Network
SCML	Slice Control and Management Layer
SDK	Software Development Kit
SDN	Software Defined Networking
SLA	Service-Level-Agreement
SNMP	Simple Network Management Protocol
STP	Spanning Tree Protocol
SQL	Structured Query Language
TCAM	Ternary Content Addressable Memory
TCP	Transmission Control Protocol
ToR	Top of Rack
ToS	Type of Service

TRILL	Transparent Interconnection of Lots of Links
TUB	Technische Universität Berlin
UDP	User Datagram Protocol
UFU	Federal University of Uberlândia
UI	User Interface
US	United States
VLAN	Virtual Local Area Networks
VM	Virtual Machine
VMS	Virtual Machine Server
VoIP	Voice over IP
VT	Virtualization Technology

Bibliography

[1] Ieee standards for local area networks: Carrier sense multiple access with collision detection (csma/cd) access method and physical layer specifications. *ANSI/IEEE Std 802.3-1985*, page 144, 1985.

[2] 802.1q virtual bridged local area networks, 2003.

[3] GEC13: Parallel service dependent load-balancing, March 2012. http://groups.geni.net/geni/wiki/GEC13Agenda/EveningDemoSession.

[4] Netfilter: firewalling, nat and packet mangling for linux, 2012.

[5] Berlin open wireless lab, April 2013. http://www.bowl.tu-berlin.de/.

[6] Global environment for network innovations, April 2013. http://www.geni.net/.

[7] libvirt virtualization API, April 2013. http://www.libvirt.org/.

[8] Openflow in europe: Linking infrastructure and applications, April 2013. http://www.fp7-ofelia.eu/.

[9] *Configuration Guidelines for DiffServ Service Classes: RFC4594*, July 2014. http://tools.ietf.org/html/rfc4594.

[10] *Network Configuration*, July 2014. http://datatracker.ietf.org/wg/netconf.

[11] *Mininet*, Januar 2015. http://mininet.org/.

[12] *Open Virtual Switch*, Januar 2015. http://openvswitch.org/.

[13] Ali Al-Shabibi, Marc De Leenheer, Matteo Gerola, Ayaka Koshibe, Guru Parulkar, Elio Salvadori, and Bill Snow. Openvirtex: Make your virtual sdns programmable. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 25–30, New York, NY, USA, 2014. ACM.

[14] Ali Al-Shabibi, Marc De Leenheer, Matteo Gerola, Ayaka Koshibe, William Snow, and Guru Parulkar. Openvirtex: A network hypervisor. 2014.

[15] C. Argyropoulos, D. Kalogeras, G. Androulidakis, and V. Maglaris. Paflomon – a slice aware passive flow monitoring framework for openflow enabled experimental facilities. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 97–102, 2012.

[16] Randall Atkinson. Security architecture for the internet protocol. In *RFC 1825*, 1995.

[17] Fred Baker, David Black, Steven Blake, and Kathleen Nichols. Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers. Technical report, RFC 2474, Dec, 1998. http://tools.ietf.org/html/rfc2474.

[18] B. Batke. Implementing and deploying quality of service (qos) for ethernet/ip, February 2009.

[19] Steven M. Bellovin. Distributed firewalls. In *Journal of Login*, 1999.

[20] Tony Bourke. Server load balancing. *O'Reilly Media*, August 2001.

[21] Kyung Taek Chung, Byun Gon Kim, Kwan Woong Kim, Hyun Soon Shin, and Byoung Sil Chon. Performance evaluation of topology aggregation in pnni network. In *TENCON 2001. Proceedings of IEEE Region*

10 International Conference on Electrical and Electronic Technology, volume 1, pages 261 –264 vol.1, 2001.

[22] The OpenFlow Consortium. Openflow switch specification / version 1.0.0, December 2009. `http://www.openflow.org/wp/documents/`.

[23] The OpenFlow Consortium. Openflow switch specification / version 1.1.0, February 2011. `http://www.openflow.org/wp/documents/`.

[24] R.D. Corin, M. Gerola, R. Riggio, F. De Pellegrini, and E. Salvadori. Vertigo: Network virtualization and beyond. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 24 –29, oct. 2012.

[25] Andrew R Curtis, Jeffrey C Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. Devoflow: scaling flow management for high-performance networks. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 254–265. ACM, 2011.

[26] M. Felser. Real-time ethernet - industry prospective. *Proceedings of the IEEE*, 93(6):1118 –1129, june 2005.

[27] ONF Open Networking Foundation. Software-defined networking: The new norm for networks. *ONF White Paper*, April 2012. `https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf`.

[28] Constantin Gaul, Marc Koerner, and Odej Kao. Design and implementation of a cloud-federation agent for software defined networking. In *Proceedings of 2015 IEEE International Conference on Cloud Engineering (IC2E 2015)*, pages 323–328, Tempe, AZ, USA, March 2015. IEEE.

[29] M. Gerola, R. Doriguzzi Corin, R. Riggio, F. De Pellegrini, E. Salvadori, H. Woesner, T. Rothe, M. Sune, and L. Bergesio. Demonstrating inter-testbed network virtualization in ofelia sdn experimental facility. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pages 39–40, April 2013.

[30] Srinivas Govindraj, Arunkumar Jayaraman, Nitin Khanna, and Kaushik Ravi Prakash. Openflow: Load balancing in enterprise networks using floodlight controller. *University of Colorado*, May 2012.

[31] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martin Casado, Nick McKeown, and Scott Shenker. Nox: Towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, July 2008.

[32] R. Hinden and S. Deering. Ip version 6 addressing architecture. In *RFC 4291*, February 2006. http://tools.ietf.org/html/rfc4291.

[33] IEEE. Media access control (mac) bridges. *IEEE Standard for Local and metropolitan area networks*, June 2004. http://standards.ieee.org/getieee802/download/802.1D-2004.pdf.

[34] M. Jarschel, F. Lehrieder, Z. Magyari, and R. Pries. A flexible openflow-controller benchmark. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 48 –53, oct. 2012.

[35] Marc Koerner. The ofelia tub-island an europe-wide connected openflow testbed. In *Proceedings of the 38th IEEE Conference on Local Computer Networks (LCN)*, pages 452–455, Sydney, Australia, oct 2013. IEEE publishers.

[36] Marc Koerner and Herbert Almus. Hla - a hierarchical layer application for openflow management abstraction. In *Fourth International Conference on Network of the Future (NoF'13)*, IEEE, pages 1–4, Pohang, Korea, oct 2013.

[37] Marc Koerner, Herbert Almus, Hagen Woesner, and Tobias Jungel. Metrics and measurement tools in openflow and the ofelia testbed. In *Lecture Notes in Computer Science (LNCS) 7586*, pages 123–134, Heidelberg, 2013. Springer.

[38] Marc Koerner and Odej Kao. Multiple service load-balancing with openflow. In *Proceedings of the 13th International Conference on High Per-*

formance Switching and Routing (HPSR), IEEE, pages 210–214. IEEE publishers, 2012.

[39] Marc Koerner and Odej Kao. Optimizing openflow load-balancing with l2 direct server return. In *Fourth International Conference on Network of the Future (NoF'13)*, IEEE, pages 1–5, Pohang, Korea, oct 2013.

[40] Marc Koerner and Odej Kao. Evaluating sdn based rack-to-rack multi-path switching for data-center networks. volume 34C, pages 118–125. Elsevier, 2014.

[41] Marc Koerner and Odej Kao. Oftables: A distributed packet filter. In *The 6th International Conference on Communication System and Networks (COMSNETS)*, pages 1–4, Bangalore, India, jan 2014.

[42] Marc Koerner, Alexander Stanik, and Odej Kao. Applying qos in software defined networks by using ws-agreement. In *Cloud Computing Technology and Science (CloudCom), Proceedings of the 2014 IEEE 6th International Conference on*, volume 2, pages 893–898. IEEE Computer Society, December 2014.

[43] Marc Koerner, Alexander Stanik, and Andreas Kliem. An approach for QoS constraint networks in cloud environments. In *Fourth International Conference on Network of the Future (NoF'13) (NoF'13)*, IEEE, pages 1–3, Pohang, Korea, Oct 2013.

[44] D. Law, D. Dove, J. D'Ambrosia, M. Hajduczenia, M. Laubach, and S. Carlson. Evolution of ethernet standards in the ieee 802.3 working group. *Communications Magazine, IEEE*, 51(8):88–96, August 2013.

[45] Yu Li and Deng Pan. Openflow based load balancing for fat-tree networks with multipath support. *Florida International University*, 2013. `http://users.cis.fiu.edu/~pand/publications/13icc-yu.pdf`.

[46] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner.

Openflow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, April 2008.

[47] N.B. Melazzi, A. Detti, G. Mazza, G. Morabito, S. Salsano, and L. Veltri. An openflow-based testbed for information centric networking. In *Future Network Mobile Summit (FutureNetw), 2012*, pages 1–9, 2012.

[48] Dan Nessett and Polar Humenn. The multilayer firewall.

[49] Big Switch Networks. Floodlight, August 2014. `http://floodlight.openflowhub.org/`.

[50] The OFELIA Control Framework, April 2013. `http://fp7-ofelia.github.io/ocf/`.

[51] LAN/MAN Standards Committee of the IEEE Computer Society. Ieee standard for local and metropolitan area networks–media access control (mac) bridges and virtual bridged local area networks. *IEEE Std 802.1Q-2011 (Revision of IEEE Std 802.1Q-2005)*, pages 1 –1365, 31 August 2011.

[52] ONF. Open networking foundation, 2014. `https://www.opennetworking.org/`.

[53] Open Networking Foundation. *OpenFlow Switch Specification, Version 1.3.1 (Wire Protocol 0x04)*, September 2012.

[54] OpenStack. Open source software for creating private and public clouds, Jan 2015. `http://www.openstack.org/`.

[55] OpenVPN, April 2013. `http://openvpn.net/`.

[56] R. Perlman. Challenges and opportunities in the design of trill: A routed layer 2 technology. In *GLOBECOM Workshops, 2009 IEEE*, pages 1–6, 2009.

[57] R. Perlman. Transparent interconnection of lots of links (trill): Problem and applicability statement, rfc 5556, 2009.

[58] A.R.P. Ragozini, J.L. Rougier, A. Gravey, and D. Kofman. Analysis of the performance of a hierarchical pnni network. In *ATM, 1999. ICATM '99. 1999 2nd International Conference on*, pages 365 –374, 1999.

[59] Thomas Renner, Alexander Stanik, Marc Koerner, and Odej Kao. Portable sdn applications on the paas layer. In *Utility and Cloud Computing (UCC), Proceedings of the 2014 IEEE/ACM 7th International Conference on*, pages 497–498. IEEE Computer Society, December 2014.

[60] Leonard Richardson and Sam Ruby. In *RESTful Web Services.* O'Reilly, May 2007.

[61] Lawrence G. Roberts. Multiple computer networks and intercomputer communication. 1967.

[62] Lawrence G. Roberts. The arpanet & computer networks. 1995.

[63] Charalampos Rotsos, Nadi Sarrar, Steve Uhlig, Rob Sherwood, and Andrew W. Moore. OFLOPS: an open framework for openflow switch evaluation. In *Proceedings of the 13th international conference on Passive and Active Measurement*, PAM'12, pages 85–95, Berlin, Heidelberg, 2012. Springer-Verlag.

[64] E. Salvadori, R.D. Corin, A. Broglio, and M. Gerola. Generalizing virtual network topologies in openflow-based networks. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1–6, Dec.

[65] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru Parulkar. Flowvisor: A network virtualization layer. *Technical Report Openflow-tr-2009-1, Stanford University*, July 2009.

[66] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru Parulkar. Can the production network be the testbed? *In USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

[67] B. Sonkoly, A. Gulyas, F. Nemeth, J. Czentye, K. Kurucz, B. Novak, and G. Vaszkun. On qos support to ofelia and openflow. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 109–113, 2012.

[68] S. Soucek and T. Sauter. Quality of service concerns in ip-based control systems. *Industrial Electronics, IEEE Transactions on*, 51(6):1249 – 1258, dec. 2004.

[69] Alexander Stanik, Marc Koerner, and Odej Kao. Service-level agreement aggregation for quality of service-aware federated cloud networking. *IET Networks Journal*, -:1–6, 2015.

[70] Alexander Stanik, Marc Koerner, and Leonidas Lymberopoulos. Sla-driven federated cloud networking: Quality of service for cloud-based software defined networks. volume 34, pages 655–660. Elsevier, 2014.

[71] Marc Sune, Leonardo Bergesio, Hagen Woesner, Tom Rothe, Andreas Koepsel, Didier Colle, Bart Puype, Dimitra Simeonidou, Reza Nejabati, Mayur Channegowda, Mario Kind, Thomas Dietz, Achim Autenrieth, Vasileios Kotronis, Elio Salvadori, Stefano Salsano, Marc Koerner, and Sachin Sharma. Design and implementation of the ofelia fp7 facility: The european openflow testbed. *Computer Networks (COMNET)*, 61:132 – 150, 2014. Special issue on Future Internet Testbeds - Part I.

[72] Cisco Systems. Cisco PNNI Reference Guide, March 2001. Release 1.

[73] Andrew Tanenbaum and David Wetherall. *Computer Networks*. Prentice Hall Professional Technical Reference, 5th edition, 2010.

[74] A Tavakoli, M Casado, T Koponen, and S Shenker. Applying nox to the datacenter. *Eighth ACM Workshop on Hot Topics in Networks*, 2009.

[75] Mininet Team. Mininet, 2014. `http://mininet.org/`.

[76] Technical Committee The ATM Forum. Private network-network interface specification version 1.0 (pnni 1.0), March 1996.

`http://www.broadband-forum.org/ftp/pub/approved-specs/af-pnni-0055.000.pdf`.

[77] Amin Tootoonchian and Yashar Ganjali. Hyperflow: A distributed control plane for openflow.

[78] Amin Tootoonchian, Sergey Gorbunov, Yashar Ganjali, Martin Casado, and Rob Sherwood. On controller performance in software-defined networks. In *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, 2012.

[79] R. van der Pol, S. Boele, F. Dijkstra, A. Barczyk, G. van Malenstein, J.H. Chen, and J. Mambretti. Multipathing with mptcp and openflow. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 1617–1624, Nov 2012.

[80] L. Veltri, G. Morabito, S. Salsano, N. Blefari-Melazzi, and A. Detti. Supporting information-centric functionality in software defined networks. In *Communications (ICC), 2012 IEEE International Conference on*, pages 6645–6650, 2012.

[81] Timothy Wood, K. K. Ramakrishnan, Prashant Shenoy, and Jacobus van der Merwe. Cloudnet: Dynamic pooling of cloud resources by live wan migration of virtual machines. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '11, pages 121–132, New York, NY, USA, 2011. ACM.

[82] Timothy Wood, K.K. Ramakrishnan, Prashant Shenoy, and Jacobus Van der Merwe. Enterprise-ready virtual cloud pools: Vision, opportunities and challenges. *The Computer Journal*, 55(8):995–1004, 2012.

[83] Jianling Zhang. Internet firewall. Citeseer.