FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Monitoring the Correspondence of Physical and Virtual Network Resources in OpenFlow Based Software Defined Networks

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Software Engineering/Internet Computing

eingereicht von

Denitsa Djamiykova
Matrikelnummer 1028076

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.Prof. Dr. Schahram Dustdar
Mitwirkung: Dr. Philipp Leitner

Wien, 17.06.2013

(Unterschrift Verfasserin) (Unterschrift Betreuung)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Monitoring the Correspondence of Physical and Virtual Network Resources in OpenFlow Based Software Defined Networks

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Software Engineering/Internet Computing

by

Denitsa Djamiykova
Registration Number 1028076

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.Prof. Dr. Schahram Dustdar
Assistance: Dr. Philipp Leitner

Vienna, 17.06.2013

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Denitsa Djamiykova
Strozigasse 6, 1080 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasserin)

Abstract

As cloud computing gains more popularity, the need for easy provisioning, monitoring and prediction of the quality of the underlying network increases. In the cloud environment these requirements can be fulfilled by virtualizing the network resources. Even though network virtualization technologies already exist in the traditional networks, there is a need for new methods of network virtualization due to the new challenges that the cloud network imposes. Software Defined Networks (SDNs) are a new paradigm for designing and implementing virtual networks by abstracting the control from the forwarding plane. SDNs promise to bring better scalability, flexibility and efficiency to the cloud network compared to those provided by the conventional model. The only one production-ready implementation of the SDN idea is the OpenFlow protocol. It adopts a centralized management model of the virtual network by isolating the functions of physical devices from those of the virtual ones. Nowadays, one of the most challenging problems of cloud network virtualisation, built upon a SDN, is the poor management and monitoring of the correspondence between the physical and the logical devices.

This work introduces an extension to an existing OpenFlow controller that aims to monitor the physical network resources and to map gathered metrics to the corresponding virtual network. Using low-level monitoring in every cloud setup enables optimal utilization of the physical devices and better and faster provisioning of the logical network, as well as of the tenant's VMs. The proposed OpenFlow controller plug-in enables such low-level monitoring, centralized for the network. The plug-in provides information for changes and faults in the physical network providing feedback for any enforced management policies, and helping for an effective enforcement of network QoS and eventually, for a prediction of SLA violations. The functionality that is proposed by the implemented plug-in enables comprehensive testing and optimization of the virtual network infrastructure with regards to the physical resources.

Furthermore, an analysis of the interconnection between physical and virtual network resources is provided based on measurements gathered through continuous monitoring of the network behaviour. The methodology of the analysis includes simulations of network topologies that implement different relations between the physical and the virtual resources. The simulation results are verified by a test bed with the same topology implementations. The evaluation of the different networks shows that there is no universal solution that would fit different types of network requirements.

Kurzfassung

Mit der steigenden Popularität des Cloud Computings, nimmt auch die Notwendigkeit von leichten Bereitstellen, Migration und Voraussagen des Qualität des Netzwerks zu. In der Cloud Umgebung können diese Anforderungen durch Virtualisierung der Netzwerkressourcen erfüllt werden. Obwohl die Netzwerk-Virtualisierungstechnologien in den traditionellen Netzwerken bereits vorhanden sind, gibt es einen Bedarf an neuen Methoden der Netzwerk-Virtualisierung aufgrund der neuen Herausforderungen, die das Cloud-Netzwerk aufweist. Software Defined Networks (SDNs) sind ein neues Paradigma für die Gestaltung und Umsetzung von virtuellen Netzwerken durch Abstraktion der Regelung von der Weiterleitungsschicht. SDNs versprechen bessere Skalierbarkeit, Flexibilität und Effizienz des Cloud-NetzwerksDie einzige produktionsreife Umsetzung der SDN-idee ist das OpenFlow-Protokoll. Es implementiert ein zentrales Managementmodell des virtuellen Netzwerks durch Isolierung der Funktionen der physikalischen von den virtuellen Geräten. Heutzutage ist eines der schwierigsten Probleme der Cloud-Netzwerk-Virtualisierung, das auf eine SDN gebaut ist, die schlechte Verwaltung und Überwachung der Korrespondenz zwischen den physischen und den logischen Ressourcen.

Diese Arbeit präsentiert eine Erweiterung zu einem bestehenden OpenFlow Controller, der die physikalischen Netzwerk-Ressourcen überwachen und die entsprechenden virtuellen Netzwerk-Ressourcen zuweisen soll. Das Low-Level-Monitoring in jedem Cloud Setup ermöglicht eine optimale Ausnutzung der physischen Ressourcen und eine bessere und schnellere Bereitstellung des logischen Netzwerkes sowie der Tenant VMs. Das vorgeschlagene OpenFlow Controller Plug-in ermöglicht Low-Level-Monitoring, das zentralisiert für das Netzwerk ist. Das Plug-in bietet Informationen für Änderungen und Störungen in der physischen Netzwerk Bereitstellung, gibt Feedback für alle durchsetzten Management-Strategien und hilft für eine effektive Durchsetzung der Netzwerk-QoS und schließlich für eine Vorhersage von SLA Verletzung. Die Funktionalität, die vom Plug-in implementiert ist, ermöglicht eine umfassende Prüfung und Optimierung der virtuellen Netzwerk-Infrastruktur in Bezug auf die physikalischen Ressourcen.

Weiterhin ist eine Analyse der Verbindung zwischen den physischen und den virtuellen Netzwerk Ressourcen bereitgestellt, die sich auf eine kontinuierliche Überwachung des gesamten Netzwerk Verhaltens berührt. Das Verfahren der Analyse umfasst Simulationen, die die Netzwerktopologien von den verschiedenen Beziehungen zwischen der physischen und der virtuellen Ressourcen implementieren. Die Simulationsergebnisse werden durch einen Testumgebung mit den gleichen Topologieimplementierungen bestätigt. Die Auswertung der unterschiedlichen Netzwerke zeigt, dass es keine universelle Lösung für die verschiedenen Arten von Netzwerk-Anforderungen passen würde.

Contents

List of Figures

List of Tables

List of Listings

CHAPTER 1

Introduction

One of the hot topics in IT research and practice is cloud computing [25] [91]. Although sharing hardware and software resources over the network is not a new idea, cloud ideas are still not widely implemented in practice. Cloud computing is expected to supply elasticity, scalability, and fault-tolerance of the underlying resources to the customer within negotiated Service Level Agreements (SLA) [24]. Highly distributed, virtualized environments are built to satisfy the requirements for elasticity and scalability. Management of Virtual Machines (VMs) within the data center needs to be easier and automated in order to provide better service to the customer. Currently, the virtualization technologies give cloud providers the opportunity to meet the needs of customers for on-demand provisioning of machines. With the evolution of server virtualization and the growing needs of cloud, the network, as it is deployed in the moment, becomes a bottleneck [83]. In the cloud, properly planned data center network not only optimises VMs availability, management and performance but also protects application and data integrity. Due to the current state of the network technology most of the large Virtualized Data Centers (VDC)s are rather static. VMs are usually created and assigned to a server and remain there. The ability to move VMs around would minimize the power consumption and load balance the workload across the servers and the network. It should also allow the enterprises to significantly reduce both capital investment and operating costs.

It is possible to improve services provided to the cloud customers using large scale data center architecture's optimisation. Such an optimization can be achieved by better understanding the network behaviour and, thus, better network management. A first step in this direction is gathering network knowledge by monitoring the resource consumption in order to give visibility of tenants' and VMs' traffic.

1.1 Motivation

The cloud computing network [7] is the fabric that provides secure user access and an infrastructure for the deployment, interconnection and aggregation of shared data center components

as required, including applications, servers, appliances, and storage. These new network functions in the cloud impose challenges to the underlying physical networks, that existing network technology cannot meet [68]. The architecture of existing networks has a limited degree of user programmability for traffic engineering and management, and has an inconsistent traffic management between equipment of multiple vendors. This is a challenge for organizations deploying private and multi-tenant public cloud computing infrastructures. Requirements, such as end-to-end traffic engineering across heterogeneous networks to dynamically define flows, determining what path those flows take through a network, and controlling which types of network services are applied to specific flows are practically not possible without virtualized networks [52] [64].

Virtualization also extends the mobility, scalability and isolation requirements compared to most traditional physical deployments. As there is a common problem in Infrastructure-as-a-Service (IaaS) [22], and not only, with the speed and efficiency of the VM migration, mobility becomes more and more important. This, in addition to the consideration about the size of typical data center, extends the scaling limits several times. Unfortunately, standard networks are not able to face the challenge of fast, easy and efficient VM migration due to the manual configuration that every VM migration requires. Multi-tenancy is another serious problem for traditional networks. Having a number of different tenants sharing the same physical infrastructure implies strong isolation in the virtual network. This could possibly be done by *Virtual Local Area Networks (VLANs)*, but practise shows that this solution provides only partial isolation by network tagging [63]. Problems, such as Quality of Service (QoS) enforcement and SLA, are still present.

A possible solution to some of these problems provide the Software Defined Networks (SDNs) [35] paradigm. Throughout the last year, SDN gained more and more popularity as an approach to virtualize networks by separating the control plane from the data plane in different devices. One implementation of this paradigm is proposed by the OpenFlow [65] protocol, as it uses a "controller" to take over all responsibilities of the control plane in a centralized place in the network. Using an implementation of such a controller provides another level of abstraction to the cloud administrators, and the ability of programmatically manage the underlying network on-demand. Virtual networks on the other hand, provide advantages to the networking layer, such as software flexibility and well-defined end host events that are not present in physical networks. It provides inter- and intra-VM connectivity and is involving many of the same functions provided by the physical layer.

Basically, SDNs aim at simplifying the network management by abstracting its physical resources in a central place. Adopting such a centralized management model of virtual networks provides an isolation of the functions of physical devices from those of the virtual ones. Therefore, large implementations of SDNs need monitoring on both levels. Monitoring of the low level metrics of the physical devices firstly helps for better decision making by the VM allocation, and secondly sets the basis for network SLA [13] for different tenants. Further, QoS could be enforced according to these metrics. Another use of the proposed monitoring is in the implementation of self-healing mechanisms in the OpenFlow controller [54] [33].

Figure 1.1 shows an implementation of the SDN paradigm with the help of the Floodlight controller [21] [33]. The controller abstracts the physical resources and thus provides the tenants with homogeneous resources for building their own virtual networks. Then the controller

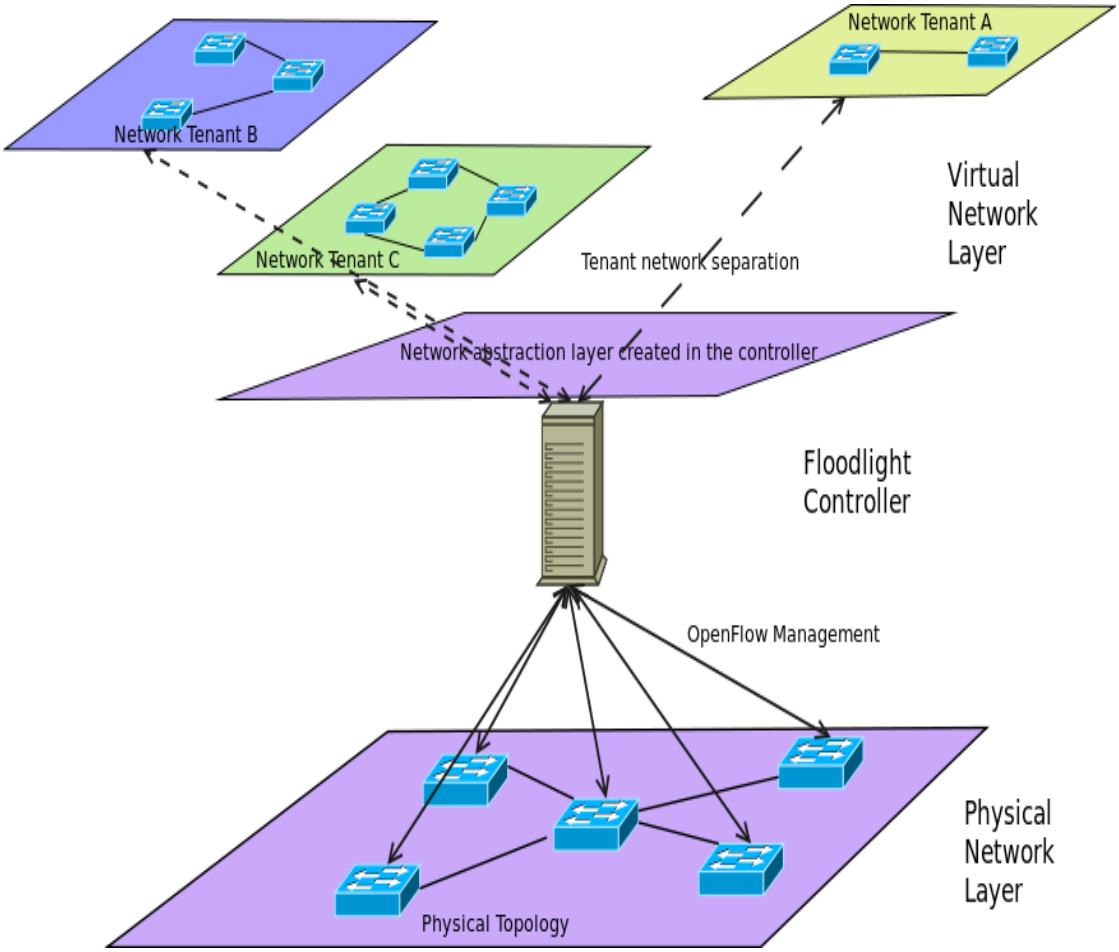

Figure 1.1: Network Virtualization via Floodlight Controller

handles the traffic between the tenant VMs. As Figure 1.1 shows, after the network has been abstracted, the correlation between the physical and virtual networks is lost. Management of the correspondence between the physical and the virtual resources is needed in order to provide better and efficient reaction in the virtual network.

In the few current implementations of the OpenFlow concepts the basics for primary flow monitoring exists, but they do not pay attention to monitoring the different levels of abstraction or to extensive monitoring of the physical resources. Measurements that are able to provide extensive information for the state of the physical network resources are normally expensive and inflexible. They are using special hardware and at times could deliver inaccurate information because of the use of an aggressive sampling rate on incoming packets. Therefore, monitoring of OpenFlow set-ups is usually done with external tools which measure end-to-end active metrics, such as round trip and end-to-end delay. They are considered the most important because the user's experience depends mainly on them. Nevertheless, the controller itself has no

information about them and thus no way of managing them. Thus, to automate the management process of physical in compliance with virtual resources, and to eliminate the need for manual troubleshooting or changes in the logical network, the monitoring needs to be done on the virtual (flow) level as well on the physical devices.

1.2 Contributions

The creation of a central measurement point for the virtual and physical network resources will provide accurate information for the device utilization, which will enable optimal utilisation of the physical devices. Further, the definition of physical-virtual mappings will eventually lead to faster and reliable logical network provisioning. The centralized monitoring in the OpenFlow controller will guaranty fast feedback on each radical network change or fault. Further, implementation of any kind of QoS will be as accurate as possible due to the data gathered throughout the controller's operation time.

Apart from providing feedback for any provisioning management policies, helping to effectively enforce network QoS in the controller is another benefit of this kind of operational measurements. As mentioned previously, monitoring of physical resources is crucial for any kind of cloud controller. The metrics are exposed to any cloud controller and set the basis for prediction of further network SLA violations. In addition to benefits that such monitoring brings to cloud environments, it will make testing and troubleshooting the network infrastructure easier and more comprehensive. This central measurement point is implemented within the OpenFlow controller, in this case Floodlight, as a plug-in, such that other modules and functionalities of the controller remain independent. The information gathered is exposed via two types of interfaces - Representational State Transfer (REST) [74] and file-based. The REST interface is intended for cloud controller usage, whereas the file-based is more appropriate for human interaction.

Currently, OpenFlow users are monitoring the logical network with the help of the library liboftrace [48] or in a cloud environment, NetFlow [44]. Primarily, examining these tools is needed to extract useful metrics. This information needs to be exposed to the other monitoring tools or cloud controllers in a suitable way. At the same time, the plug-in is expected to be consistent with the Flows configured in the Floodlight.

The main goal of this work is to extend the OpenFlow controller and thus to enable it to manage the correspondence between the physical and virtual network resources in a more efficient way. In order to achieve this aim, we develop a plug-in for monitoring the physical network resources and the metrics that are further gathered are mapped to particular tenant's virtual network. On the one hand, we have the southbound metrics per port and per switch, and on the other hand, the northbound OpenFlow metrics. Metrics of the physical resources that are recognised to be important, are categorised in two categories: host and network passive metrics.

In case of SDN, the network virtualization implementation, in most of the cases, happens on dedicated hosts. Each of them hosts multiple switches and/or controllers, and therefore it is very important for the cloud provider to be aware of the performance and the capacities of the underlying machine. The metrics in Table 1.1 provide vital information about the physical hosts and about the quality of the network. The network quality, in this case is measured by the network passive metrics for both port and switch.

Host	Per port	Per switch
CPU utilization	Packets per second	Packets per second
Memory utilization	Broadcasts per second	Broadcasts per second
Availability	Unicasts per second	Unicasts per second
	Errors per second	Errors per second
	Bits per second	Bits per second
	Maximum traffic rate	Maximum traffic rate
	Load	Load

Table 1.1: Physical Network Metrics

Other particularly interesting criteria are the OpenFlow metrics presented in Table 1.2. These metrics focus on the controller functionality and on the characteristics of the flows, which each switch maintain. The information that these metrics provide is relevant for the abstracted network layer. However, they give information also for the overall quality of the logical network and the function of the management unit. Further discussion and detailed description of the proposed metrics is provided in section 4.3.1.

Controller	Flow in switch
Flow set-up time	Active flows
Transactions per second	Traffic rate
Connections per second	Packets per second per flow
Flow arrival rate	Flow entry usage
	Maximum concurrent connections

Table 1.2: OpenFlow Metrics

In order to prove the simulation results we will build a test bed with two OpenFlow-enabled physical and different number of virtual switches. The interconnection between physical and virtual network resources will be analyzed based on measurements gathered through continuous monitoring of the network behaviour under specific bandwidth load. The results of this analyze will build a standpoint for optimal physical-virtual setup depending on the number of tenants.

The second important task of this work is to provide an evaluation of different physical-to virtual network resource mappings and to propose optimal mapping for different environmental factors such as the number of tenants, number of VMs and traffic load. The conclusions of this experiment are considerate useful while analyzing and designing data center network infrastructure. Further, in the operation and maintenance process this proposals will set the base for designing custom provisioning management policies.

1.3 Thesis Structure

The remainder of this thesis is structured as follows:

- Section 2 describes the current state of the art in the areas of cloud computing, cloud networking, virtual networks and cloud network monitoring. It presents the Floodlight controller, an implementation of the OpenFlow protocol. Furthermore, we briefly discuss cloud computing network standards, such as the overlay networks and Virtual eXtensible LAN (VXLAN) [18]. Finally, the SDN paradigm is discussed.

- Section 3 provides an overview of the related works in the area of network virtualization. Two different approaches of network virtualization are discussed. The first one is observing the underlying physical resources as a single device (libvirtnet [88]). The other virtualisation approach is a hybrid approach proposed by Cisco [18]. Similar approaches to metrics monitoring in server virtualization are briefly examined [85] [26]. Moreover, we present some tools for traffic monitoring in the existing OpenFlow network architectures.

- Section 4 contains precise definition of the metrics that are to be gathered and their classification. Subject of discussion are also all Floodlight's architectural characteristics that are relevant to this thesis. This section covers the practical part of the thesis, which examines the design of the proposed plug-in. Additionally, a brief discussion of a Simple Network Management Protocol (SNMP) module that is part of the plug-in is provided. Afterwards, the design of the four different types of physical to virtual mappings is examined.

- Section 5 focuses on the implementation details of the proposed monitoring plugin. Afterwards, details of the physical to virtual mappings implementation are examined. We further provide a detailed description of the simulated infrastructure. Further, focus of the section is also the implementation of the testbed infrastructure.

- Section 6 covers an evaluation of the implemented mappings. Each of these mappings is evaluated based on simulations and testbed. The testbed is implemented on OpenFlow-enabled switches, connected to the Floodlight controller. This infrastructure will connect the Quantum (Openstack's Network-as-a-Service provider). Furthermore, different traffic loads will be used to evaluate the explored mapping.

- Section 7 concludes the thesis with a short summary and plans for improving the plug-in and proposals for other useful functionalities for cloud networks.

CHAPTER 2

State of the Art Review

In this section we discuss the concepts of *cloud computing*, the core building blocks of *cloud virtual networks* and the current state of the art in *network virtualization*. The rest of the section is dedicated to the management of virtual network resources.

2.1 Cloud

As businesses constantly aim to optimize their Return of Investment (ROI), vendors as well as end customers can benefit from the idea of computing as an utility, known as the cloud. Cloud computing has been a buzzword for long, but still there is ambiguity in its definition. The definition of the National Institute of Standards and Technology (NIST U.S) [57] is the one that is widely used in the Cloud Computing community:

> Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction

It enables developers to innovate without worrying about losing costly resources and caring about capital investment in hardware. The idea of providing one server for 5 hours on a price comparable to the price of 5 servers for one hour is what makes the ROI even faster for more scalable programs.

However, the concept of using computing resources on utility basis does not appear first in relation to cloud computing. Therefore, we should first compare the concepts of cloud- and grid computing [30]. According to [30], grid and cloud computing differ mostly in their related business model. Being scientifically oriented, grids are not widely used for commercial purposes, whereas clouds are mainly customer oriented. This results in great differences in the operating scales and in the architectures of both computing concepts. While both have the fundamental *Fabric* and *Application* layer, in the cloud it is not possible to separate the unified resources in

connectivity, resources and collectiveness, as it is done in grid. Even though both base their compute model on a batch-schedule model, the cloud faces challenges, such as QoS and SLAs, which according to [30], are not that important for grids. Moreover, the authors point out that applications that characterise clouds are loosely coupled, transaction-oriented and likely to be interactive, while in grids they are strongly batch-scheduled. With regards to the data model, the paper states that they are very similar, except for the fact that in cloud security issues are more important in comparison to the grid. Another difference, implied by the business model in clouds, is the role of virtualization in the two computing paradigms. In the cloud, virtualization is used for abstraction and encapsulation, as well as server and application consolidation, configurability, increased application availability, improved resource provisioning and automation of monitoring and maintenance. [30] also clarifies the differences in both programming models: grids and clouds usually utilise the same programming models, but it is difficult to achieve high performance computing (HPC) in clouds as it is done in grids, as it requires low latency network for scaling to many processors. Security is another important aspect [30]; in cloud, it is brought to a higher level. Some security problems are relevant to cloud computing, but are not present in grids due to the organised resource sharing within its *Virtual Organisations* (set of institutions or logically related projects/groups [30]). Such problems are caused by the large number of different customers and the Internet exposure, this poses security threats not only for the customers, but also for the data center infrastructure.

In contrast, others [77] [6] are focusing on defining cloud computing without referencing to grids. In [6], the authors are defining the cloud: "*Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services*". The main presented characteristics are

- Massive scale
- Virtualization
- Free software
- Autonomic computing
- Multi-tenancy
- Geographically distributed systems
- Advanced security technologies
- On-demand self-service
- Ubiquitous network access
- Resource pooling
- Rapid elasticity
- Pay per use

Two of the cloud deployment types of models are discussed: the public and the private models. The private cloud is enterprise owned or leased, whereas the public is megascale infrastructure. According to [12], the third type is a hybrid, composition of at least two clouds and shared resources in a specified community. The authors classify the delivery models in three categories as follows:

- Software as a Service (SaaS) - Uses providers' applications over a network
- Platform as a Service (PaaS) - Deploys customer-created applications to a cloud
- Infrastructure as a Service (IaaS) - Rents processing, storage, network capacity, and other fundamental computing resources, and deploys customer-created applications

Another point discussed in [12] that characterizes cloud is the cloud computing stack. The cloud architecture is generally consistent of hardware and operating systems that are on various physical machines in the system. Another building block is the network, including the Domain Name System (DNS), Dynamic Host Configuration Protocol (DHCP) and the subnet organisation of the physical machines. The process of giving each VM a unique virtual Media-Access-Control (MAC) address includes virtual bridging of the network. Another building block in cloud architecture is the hypervisor. Popular Virtual Machine Monitors (VMMs) include Xen [94] and KVM [47], VirtualBox [89] and VMware [90]. In order for a VM to run it needs a virtual hard drive - the VM disk image. Hence, for practical purposes, there is an image repository with different operating systems already installed on the images and ready to use. [6] explains that the cloud controllers are mainly used to manage the image retrieving and signalling the VMM and the underlying DHCP and IP bridging when particular VM needs handling. [12] states that the cloud controller is also responsible for satisfying the end-user requests. Eucalyptus, OpenNebula and Nimbus are three of the main open-source cloud computing software platforms [77]. They play the role of a cloud controller and manage the VM provisioning for IaaS in the aforementioned cloud architecture. The three frameworks are thoroughly compared in [77]. Authors clarify that they have different disk image storages but basically operate on the same hypervisors. They have customisable images, but the OpenNebula's one is considered most customisable. They operate differently over DHCP: Eucalyptus has a DHCP on a cluster controller level, whereas Nimbus has one on each individual compute node. Nevertheless, according to [77], there are considerable networking issues in production, such as bridging management complexity and inability to scale.

[25] provides an extensive review on other adoption challenges rather than on the networking problems themselves. According to [25], security is the hardest challenge to overcome in order to adopt cloud computing widely in production. The lack of standardisation in this area of study, as well as the wide variety of different costly and charging models also slackens the pace of cloud implementation. SLAs are another interesting topic that remains open for expansion [13]. Easier and more correct SLA compliance will benefit not only the customer, but also the cloud provider.

As the main idea behind the cloud is pay per use, the customers are entitled to require the service at according quality level. SLAs are the contract setting this level of quality. Architectures on the cloud provider that take in to account the existence of SLAs are customer-driven and

QoS aware [25]. They are also risk management aware, and so they allocate resources dynamically to satisfy requirements for new and existing customers. Virtualisation technologies are crucial for flexible allocation of resources to applications and for effective SLA-based resource provisioning in clouds.

2.1.1 Cloud Network Infrastructure

Cloud computing with its specifics poses new challenges for network infrastructure and management [68]. The areas of network management for conventional networks are: fault-prevention, configuration, accountability, performance, and security. This requires managing the network devices and the physical links that connect the various network elements. As [68] points out, in cloud, the network is in most cases virtualized, so here management is concentrated on the virtualized resources. That is why the key aspects of the network management need to be expanded for cloud environments. The following proposals in this direction are made by [68]:

- Fault management - the ability of the cloud-based virtual network to respond to network changes that cause failures automatically.
- Configuration management - composed of three aspects: first, in terms of the VM network configuration, each VM should be able to reach other tenants' VMs; second, the end user needs to have the experience of direct Ethernet connectivity to tenant VMs; and finally, clients should have the ability to configure their cloud-based network.
- Accounting management - measuring the network utilisation for the client, but also in the cloud.
- Performance management - as the cloud network is built on top of other virtualized and physical networks, here performance management includes not only the top cloud network, but also each network item independently.
- Security management - gives the ability to use the cloud resources from the multi-level cloud providers in a secure way. Current approaches to security management for conventional networks have the same approach.

In the network infrastructure aspect, the cloud computing model poses requirements such as elasticity and scalability, as well as on-demand provisioning, which are inapplicable to the traditional network infrastructure. The existing cloud network, like the conventional, includes routers, firewalls, Ethernet switches, fiber channel switches, server load balancing and volume based billing/control devices [16]. A firewall is used to protect the servers and storages, whereas load balancers share the loads between each server. Traffic volumes are rated and billed by volume based billing devices.

Figure 2.1 represents the existing cloud computing network architecture. According to [16], cloud computing data centers are usually constructed of many network and service devices. One of them is a core router, which connects the data center to the Internet. On top of the data center core is the aggregation layer, where access layer devices are placed.

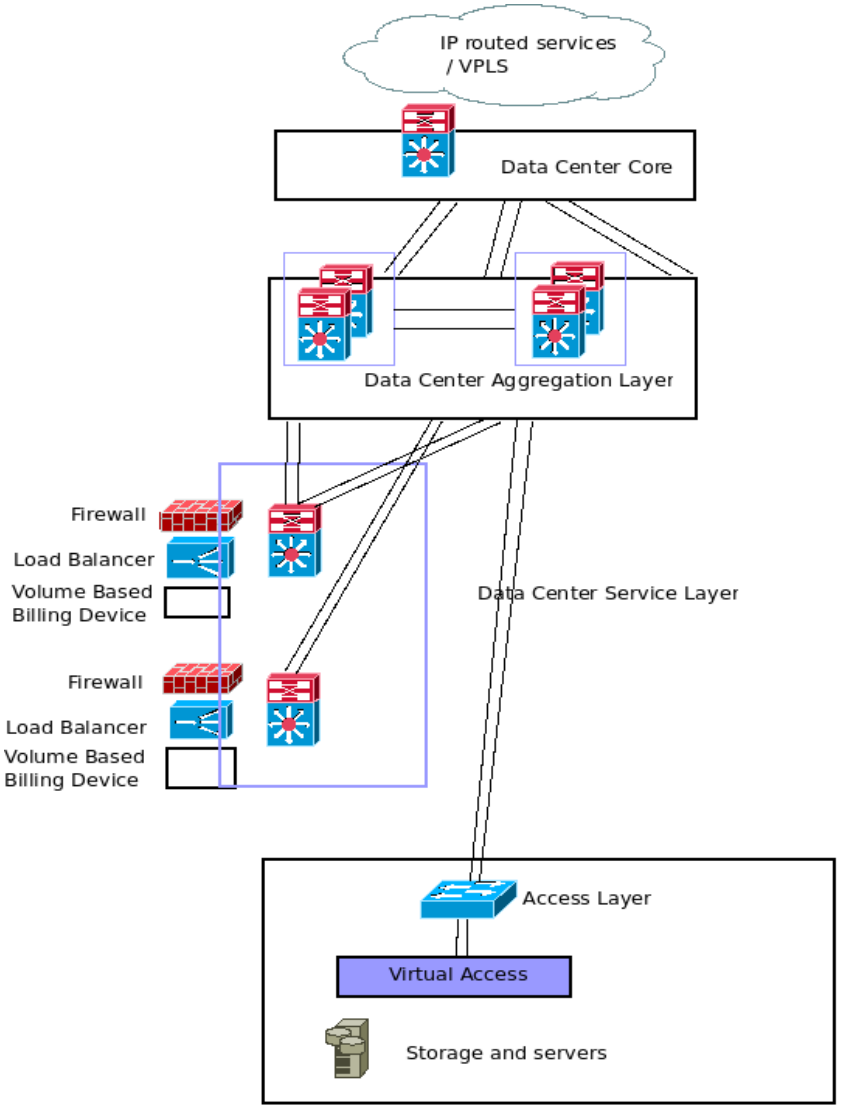

Figure 2.1: Cloud Network Architecture

In the services layer, there are the application devices (firewall, load balancer and volume based billing). As discussed in [16], they are connected to the aggregation layer and are routed by the aggregation device. Different services have different requirements concerning network architecture. Some may be using firewall others may not. Similarly, they can have different billing.

The next layer is the access layer, where the physical connectivity between the servers and the network is realised. It also includes VMs, servers and storage devices. Networks built on the above discussed model are considered static and in most cases slow and difficult to manage and provision.

The server virtualization itself have further effects on the network resources and management. Virtualizing server resources poses scalability challenge to the network layer. [40] examines the effects of virtuallization on the underlying network. A single CPU / RAM / hard disk could be emulated as multiple parallel CPUs / RAM / hard disks, and various Operating systems are enabled to run on one platform independently, which is accomplished through virtualization. [40] states that virtualization in current cloud infrastructures lacks the support for network administration among virtual machines. It also fails in monitoring the communication flows between them, as it takes place in one and the same host. For these purposes, the virtual switch has been developed.

The IaaS services implemented on current networks often have problems meeting the requirements of real-time execution. Therefore, the time constraints need to be enforced in virtual infrastructures within Service Level Objectives (SLOs) that meet virtual machine interconnection constraints. The SLOs are usually declared in the SLAs, so network virtualization needs to be SLA aware [63].

Other requirements for the network recognised by [40] to be set by the cloud computing model are:

- The network infrastructure needs to be flexible, instead of static.
- Network services need to be location independent: delivered wherever data, application, and users are and whenever services are needed.
- Network resources need to be abstracted so that provisioning can be automated and actions can be orchestrated through common interfaces.

2.2 Network Virtualization

Currently, in both research and practice, there is one prevailing approach to resolving the aforementioned problems, and that is through *network virtualization*. Network virtualization recently became a widely used term that generally means software abstraction of underlying physical network resources [4]. Network management, as practiced in the traditional networks, needs to be approached differently when dealing with network virtualization. The ways of doing this are discussed in the end of this section and are a consequence of the newly-developed virtualization technologies.

As discussed earlier the current cloud data center consists of tens of hierarchically connected switches. To improve cost effectiveness for each client, there should be a better resource utilisation. A progress in this direction is server virtualization. As a consequence, the underlying network infrastructure began to be the bottleneck in computing environments having the characteristics of the cloud. In the cloud, the virtualized infrastructure must serve multiple customers. These customers have individual divisions or departments, therefore, the provided infrastructure must be flexible enough so that virtual workloads can be moved into, out of and between clouds, on-demand. Likewise, the large numbers of diversely distributed applications impose new requirements for functionality and performance of the network. Requirements such as security, various communication paradigms (content-addressable networks [45]) and QoS (performance

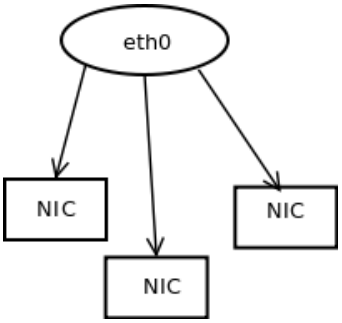

Figure 2.2: Bridging

guarantees for streaming media, etc.) are getting more and more significant. The suggested idea that aims solving these problems is sharing network resources [4] [92] via network virtualization, i.e., virtualization that abstracts the network resources (nodes, links and topologies). It provides network slicing and, thus, separation of concerns in the network infrastructure itself.

2.2.1 Technologies

There are a number of different approaches for network infrastructure virtualization, which are now listed and discussed, ordered chronologically as they appeared.

2.2.1.1 Traditional Network Virtualization

I/O Network Virtualization With respect to the I/O network virtualization, the most popular approach is *bridging*. A network bridge, described in [52], connects Ethernet segments together in a protocol independent way, it forwards packets based on the Ethernet address. Figure 2.2 graphically shows this concept.

The way a VM communicates with the other elements of the network is by having many virtual *Network interfaces cards (NICs)* that are mapped one-to-one to the same number of virtual NICs of the host machine. The host's virtual NICs are bridged to the physical NIC (cf. Figure 2.3). This approach is considered effective when managing a single instance of a physical machine without many VMs on it, as it requires manual configuration. It also introduces great overhead when the number of VMs grows [52].

As the bridging technology poses some performance problems, it is usually coupled with the NICs bonding. The paper explains the NIC bonding as a software aggregation of multiple physical network links.

Virtual Router Another approach to network virtualization is proposed in [55]. The data plane in the router is a very important part of the architecture, because it makes the forwarding decisions. Furthermore, it is responsible for processing of millions of packets per second. Therefore, it needs to be scalable, flexible and provide isolation. Those requirements are a target of the router presented in [55]. The proposed router consists of a number of virtual machines that are allocated on the underlying physical resource. Each VM is assigned a flow to manage.

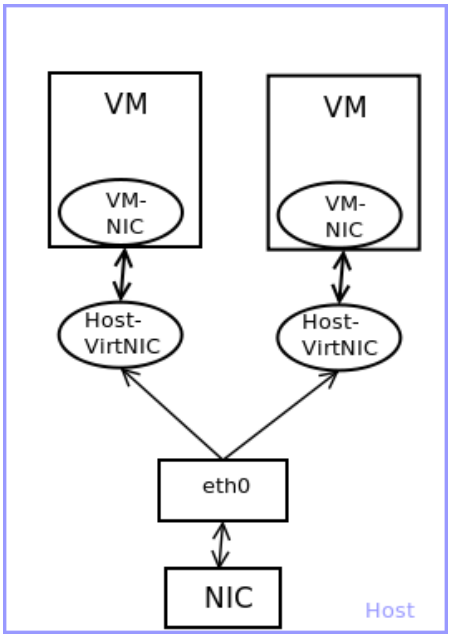

Figure 2.3: VM Bridging

There is also another VM, the last one in the model that controls the router itself. The VMM ensures that the VMs and the respective flows are isolated. Hence, evaluation with near cloud production traffic shows that because of the virtualization overhead, this kind of routing is not suitable for this traffic scale. The paper argues that in particular cases, such as not network intensive applications, a similar approach can be applicable.

Data-Center Network Virtualization To address these issues, diverse approaches of visualisation on the other levels of abstraction are developed. The next layer of network virtualization is accomplished using *Virtual Routing and Forwarding (VRF)* and *Virtual LANs (VLAN)* [17]. VLANs isolate different type of traffics, and switches through the same LAN interface using tunnelling. The segmentation of the network is host-centric and is maintained by different subnets. The VLANs virtualization, as shown in [17], happens on Layer 2, while in the case of VRF it is on Layer 3. VRF is based on physical routes, which support the virtual routes that maintain their own routing tables and routing protocol instances. In VRFs no tagging on packet level is needed, but on the interface level, which means that the traffic is separated solely on IP basis.

2.2.1.2 Traditional Network Virtualization Problems

Although the network I/O virtualization enables sharing of host network resources of a great number of VMs, it increases the complexity and cost of management of the network on the host level [51]. As hundreds of virtual NICs can be placed on a single host, even IP management becomes a problem. As the authors point out, the large number of VMs implies large volume of

traffic, and as a consequence, multiplexing of the packets within the host is difficult, slow and error-prone. A third problem is the load on the machine. Managing many VMs, their NICs and multiplexing the traffic causes CPU overhead.

In a cloud environment the use of the VLAN technology can cause problems, when there is a large number of tenants, each of which needs a different number of VLANs. According to [17] that leads to exponential grow of VLANs number, making the network very expensive and complex.

2.2.1.3 Overlay Network Technologies

Most of the protocols for network virtualization are based on creating virtual network overlays. Although the concept of overlays is not new, there is still no definition that is commonly acceptable. For the purposes of this work, we operate on the proposed definition by [20]:

> An overlay is a set of servers deployed across the internet that: a) Provide infrastructure to one or more applications, b) Take responsibility for the forwarding and handling of application data in ways that are different from or in competition with what is part of the basic internet, c) Can be operated in an organized and coherent way by third parties (which may include collections of end-users)

There are different types, depending on the purpose, of overlays, for instance peer-to-peer, content-delivery, routing and security overlays. In the past, the overlays are mainly implemented using VLANs and VPNs. These implementations are proven over time to be not optimal for largely distributed environments, such as a cloud [20]. In addition, the paper states that none of the current solutions enable VM communication and migration across Layer 3 boundaries without impacting connectivity. Therefore, recently other technologies are used for creating virtual overlay networks. Popular protocols include VXLAN, NVGRE, STT, and SPB (IEEE standard). At its core, the overlay technology aims at creating a network on top of the existing network. The nodes are linked via virtual paths that may correspond a number of other virtual or physical links. Figure 2.4 shows layerred logical view of simple overlay. There, the connectivity on the network layer is based on virtual links on top of the optical and data link layer paths. Popular examles of overlay implementations are the frame relay and Asynchronous Transfer Mode packet switching infrastructures.

VXLAN *Virtual eXtensible LAN (VXLAN)* [18] uses the MAC-in-UDP encapsulation mechanism to create Layer 2 overlay on a Layer 3 network. This enables Layer 2 connections running over a number of different Layer 3 physical networks. The *VXLAN Tunnel End Point (VTEP)* provided by the switch is responsible for the encapsulation. As explained in [18], the VTEP adds an identifier specific for the network. It encapsulates Ethernet Layer 2 Frames into an IP packet marked by the new 24-bit identifier. That means that more than 16 million virtual networks can operate within the same administrative domain. Due to this new identifier, the VXLAN is not visible from the VM, so that they still communicate on MAC-basis. The VXLAN segment is a Layer 3 construct that replaces the VLAN segmentation mechanism.

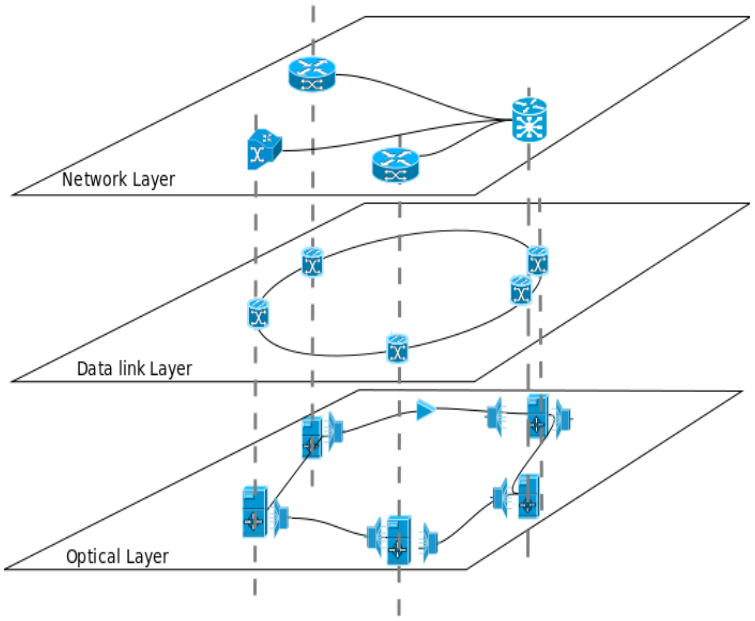

Figure 2.4: Overlay Network Example

NVGRE *Network Virtualization using Generic Router Encapsulation (NVGRE)* [27], as the name suggests, uses the GRE tunneling protocol. NVGRE has the same design as the VXLAN with two main differences. Most of the network devices do not parse GRE headers in hardware, so software parsing is needed, which may lead to performance and compatibility issues. The other difference is that the NVGRE does not have any control plane functionalities. Although the first problem, is believed to be solved by using intelligent network interface cards [27], the second one is still unresolved. The proposed solution in [27] with intelligent NIC is based on equipping them with APIs for integration with overlay controllers and hypervisor management systems.

STT *Stateless Transport Tunneling (STT)* [8] is another protocol creating Layer 2 virtual network over Layer 2/Layer 3 physical networks. Here also Virtual Network ID (VNID), which is 24 bits wide, is used to mark the network. Another similarity to the VXLAN is that the transport source header is manipulated to take advantage of multi-pathing. The two things that distinguish STT from the other two technologies described earlier are the statelessness and allocation of more header space to the per-packet metadata [8]. The first one allows tunnel endpoints within the end system to use the TCP segmentation offload of the existing TCP offload engines. The second one provides more flexibility for the virtual control plane, although none is specified at the moment inside the STT.

SPB *Shortest Path Bridging MAC-in-MAC (SPBM)* [23] is the only technology for network virtualization that is an IEEE standard. It uses MAC-in-MAC encapsulation and the IS-IS

routing protocol to provide Layer 2 network virtualization. This virtualization is achieved on top of the VLAN extension, enabled by the 24 bit Virtual Service Network (VSN) Instance Service IDs (I-SID). The I-SIDs are parts of the outer MAC encapsulation. A major difference to the overlay protocols described so far, is that no changes are required in the NICs and switching hardware, as long as they support MAC-in-MAC encapsulation [23]. As SPB utilizes IS-IS, the functions of the control plane are overtaken by it. Layer 3 forwarding and Layer 3 virtualization are possible using SPB, as it can provide IP encapsulation within the outer SPB Mac. Similarly to the VLAN extension, a VRF extension is possible. They both can run together on the same SPB network to provide semi-isolation on Layer 2 and Layer 3.

2.2.1.4 SDN Virtualization Technologies

The *SDN* is a new concept in networking. It concerns not only the implementation technology but also the network management, which is later discussed in details. The technologies that build the core of the SDNs are an extension over the network I/O virtualization. However, few of the existing technologies of the network virtualization can make a great contribution to the paradigm, because they provide different network abstraction.

Switch Virtualization Another approach to dealing with problems of the traditional network virtualization is adding a software layer in the host that acts either like a rudimentary Ethernet bridge or like an Ethernet switch. A virtual Ethernet switch is widely used in the SDN virtualised networks. Accroding to [51] the design of this switch needs to be the same as the design of the physical switch. This means that it includes fast- and slow-path components. The former are for VLAN packet encapsulation, traffic statistics collection, quality-of-service enforcement, and packet forwarding. While the latter are for configuration and control. The virtual switch management component is implemented either in the kernel or in the user space [51]. Fast-path components are implemented in the host's kernel, this way they achieve better performance. Packet-processing and maintaining the table and flow statistics are executed in the kernel to reduce processing time. The main drawback of virtual switches is that they use CPU, RAM and NIC of the underlying host, decreasing the resources available to the VMs. Another problem caused by the virtual switching is that the performance is not scalable for traffic greater then 10Gbps on NIC [52]. Switching virtualization itself brings not much benefits rather than considerably less time for configuration of the virtual machine instance. Coupled with the OpenFlow concepts, the idea of virtual switching becomes the main building block of the future data center network architectures [59].

Open vSwitch The next step towards network device virtualization is "full" virtualization. It focuses not only on the interfaces, but on the core functionality of the network device.

Open vSwitch (OVS) [72] [70] is at the moment the most popular implementation of switching virtualization that is OpenFlow enabled. It is licensed under Apache 2.0 and is production ready. It provides standard features such as VLANs, flow control and QoS [40]. Open vSwitch is usually in the hypervisor or in the management domain and can provide connectivity between the guest VMs and the physical host interfaces. [70] shows that it can apply policing per VM

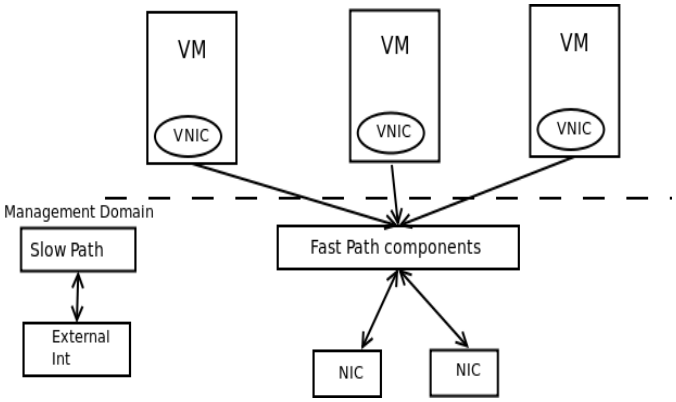

Figure 2.5: Open vSwitch architecture

instance while it also supports NIC bonding. OVS allows the network state associated with a VM to be transferred along with the VM on its migration. This minimises the manual configuration of the network. Another useful feature in the virtual networking is the abstracting of the forwarding layer, which implies that it is completely hardware independent [72]. Figure 2.5 presents the architecture of the OVS. It mainly follows the aforementioned switching virtualization. The figure depicts that the switch exports interfaces for manipulating the forwarding state. Remote processes are enabled to read and write key/value configuration pairs. Another feature of the switch is the setting of triggers that receive notifications on configuration state changes. With respect to connectivity management, OVS manages the virtual NICs as well as physical NICs connectivity. The flow-forwarding model is the same as in OpenFlow (discussed later in this section).

When used as a conventional switch it needs the same manual configuration as the physical switches. Nevertheless, when it is used as OpenFlow-enabled switch it needs notably less manual configuration - MAC addressing and controller specific configuration [70].

The usual design of data center networks with virtual switches described in [22] is the following: On each host, there is an installed number of virtual switches, which are connected directly to some physical switch in the data center network. The number of VMs on the host are directly connected to the virtual switches [52]. Using OpenFlow-enabled virtual switches provides fast feedback to the changing logical network [70]. This way, no configuration needs to be done manually on events, such as VM migration, creation or removing. Exploiting this functionalities of the OVS, building a hierarchical data center network becomes an easier task. The network architecture is an abstraction of the different virtual switches, residing on different physical hosts into one virtual switch. This enables easier management the underlying structure through single point-the high level switch [70].

OpenFlow Protocol As mentioned earlier, the implementation of other network virtualization types with Open vSwitch is only possible, when used in the OpenFlow concept [65]. Together, they virtualize the traffic more precisely by isolating it based on segregating flows [32].

OpenFlow is a protocol separating packet forwarding from routing function. It is the only

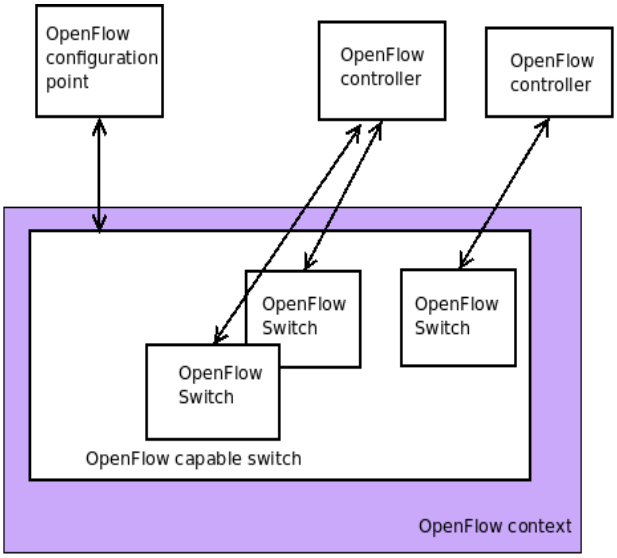

Figure 2.6: Open Flow High-Level Perspective

standardised method for implementing SDN. It describes the communication between the OpenFlow capable switch and the OpenFlow controller. The OpenFlow switch is responsible for the simple forwarding based on the entries in its *flow table*. Therefore, the following is achieved without relying on VLANs by isolating sets of MAC addresses in the filtering layer of the OpenFlow controller. The OpenFlow controller as a management unit is discussed briefly in 2.3.2.1.

In Figure 2.6 the high-level concept of OpenFlow is presented. The OpenFlow capable switch is configurable from the OpenFlow configuration point. As presented in [32], the switch is also composed of a number of OpenFlow logical switches. Each of these is connected to a particular OpenFlow controller. Thus, a single OpenFlow-enabled switch can be connected to many controllers managing a large number of networks. The parts of the OpenFlow protocol that concern this thesis are the OpenFlow resources, capabilities and OpenFlow messages. More precisely the resources, defined in the OpenFlow specification [65] are: OpenFlow port, OpenFlow queue, flow table and certificates. The flow table concerns the aggregated information about the virtual flow traffic. It is managed in the OpenFlow switch by the controller. This builds a rule-based forwarding implemented in the switch, where the rules are set by the controller. As the capabilities and messages are tightly connected to the implementation, they are discussed in Section 5

2.3 Virtualized Network Management

2.3.1 Decentralized Management

The traditional way to manage the network is decentralized. Each device can make decisions over the topology, with respect to the neighbour devices. Another aspect of the model is the QoS

policing. If any QoS needs to be enforced, each device needs to be manually configured. No central point exists where all knowledge about the network is focused.

A similar approach can be applied to the virtual networks.

2.3.1.1 Overlay Networks

Overlays networks widely exploit technologies for encapsulation and tunnelling. Managing the network infrastructure with the help of overlays is meaningful because of a number of benefits that they bring over the management of networks built upon the traditional technologies:

- multi-path forwarding in virtual networks
- easier VM migration - on VM location change, only the mapping tables in the edge switches are updated
- overlapping IPs between the tenants
- decoupling of the virtual from the physical network
- unlimited number of virtual networks

In a high-level perspective, all the protocols described in the previous section operate in the same way. A *VNID* is assigned to each endpoint address, regardless of the physical network location. Every end-point is also a *Virtual End Point(VEN)*, which encapsulates the VNID. Therefore, the end-point from other virtual networks will not see or drop the packets marked for other VNIDs. This creates an illusion of multi-tenancy.

2.3.1.2 Problems

Using encapsulation and additional tagging, each of the discussed overlay technologies brings additional overhead. Firstly, the frame-size gets unnecessarily larger by adding headers, secondly, the processing time in the server is increased, because of a lack of knowledge about the NICs. When using encapsulation, the header information used in hardware load-balancer is hidden and therefore, the task of load-balancing gets complex and expensive. Encapsulation also causes losing the ability to control flows on low-level. No control over the flows usually leads to network congestion, which becomes visible when the traffic scales up, as in the case of cloud environments. Another problem is usage of firewall devices, which also rely on the encapsulated information [69].

2.3.2 Centralized Management

In cloud environments, the decentralized network management is proven to be ineffective according [69]. It has problems with scalability, on-demand feedback to the applications that are using the infrastructure.

Another approach to managing the network is by putting together knowledge and decision-making. The idea of management centralisation is old and is not considered optimal. Anyways

when it is applied to virtual, no physical resources it is believed to have a great potential. Currently, the SDN is the only applicable concept in terms of centralisation.

2.3.2.1 SDN

Software Defined Network (SDN), similarly to all new paradigms, does not have a generally accepted definition. In the last year SDN is a synonym of decoupling of the network control plane from the network forwarding plane [36]. However, the research in the last months ([16], [7], [75]) showed that defining the SDN only as decoupling limits the actual concept too much. At present, the SDN is more focused on providing programmatic interfaces to the network. SDN is a dynamic and flexible network architecture that is believed [36] to bring the following benefits:

- improved network utilization
- automatic provisioning and management
- improved security
- implementation of network-wide policies
- visibility of applications that use network
- reduction of costs
- better testability
- dynamic response to application requirements

In addition, SDN makes the abstraction of network services, such as routing, multicasting, security, access control, bandwidth management, traffic engineering, quality of service, the processor and storage optimization, energy usage, and all forms of policy management possible, and thus it enables custom implementations of those to meet business objectives [79]. The SDNs, depending of the controller implementations, are divided in different types:

- decentralized: the controller is implemented entirely in the physical switch
- centralized: the controller makes all forwarding decisions and is central to the network
- hybrid: the controller is distributed between the switches and a central place in the network

Single Switch Network Abstraction *libnetvirt* [88] is a library consisting of set of drivers, that enable communication with nearly all kinds of underlying technologies. It uses a set of calls to create different virtual networks. This provides programmatic and on-demand virtual network resources creating in a reduced time. The only network implementation at the moment is based on OpenFlow-enabled network. Without much details, it replaces the controller with the above mentioned call-set. This way *libnetvirt* controls the forwarding of the network remotely. The

concept of the library is the same as the concept of *libvirt* [49] in the context of server virtualization. Thus, *libnetvirt* abstracts the description of the network from the physical devices and provides this description in XML format to the other applications that may use it. In such a way, third party management tools are able to control, modify and remove virtual network devices, through *libnetvirt*. [88] gives an example of a model of network abstraction, where an undefined number of physical resources are abstracted into the view of a single switch. This switch is configurable through a XML configuration file. This brings the following benefits [88] to network management:

- operating different technologies via one API
- on demand network creation, which is done entirely programatically
- no network management
- a single-node network, with all functionalities inside

OpenFlow Controller Like OpenFlow itself, the controller has a relatively simple yet flexible data plane residing in the OpenFlow switch, and it is mainly concerned with the control plane [50]. When a switch is not able to match a packet in its flow table, it refers to the controller to decide where this packet needs to be sent. According to the protocol, the flow is presented in the controller as a 5-tuple of

$\{source\ IP,\ destination\ IP,\ source\ port,\ destination\ port,\ protocol\}$.

No specification about the OpenFlow controller is now available, except the one concerning the protocol itself. It is entirely a software component, independent of the underlying hardware. As such it can be implemented in any programming language. A number of example implementation are now available:

- NOX (Network Operating System) [62]: it is written in Python and C++, includes a large number of controller applications. NOX provides control and visibility of the network built on OpenFlow switches.
- Beacon [43]: Beacon is an extensible Java-based OpenFlow controller. It was built on an Open Services Gateway initiative (OSGI) framework, allowing OpenFlow applications built on the platform to be started/ stopped/ refreshed/ installed at run-time, without disconnecting switches.
- Helios [78]: a C-based OpenFlow controller released by NEC.
- SNAC [80]: a controller targeting production enterprise networks. It is based on NOX, and features a flexible policy definition language and a user-friendly interface to configure devices and monitor events.
- Maestro [53]: an extensible Java-based OpenFlow controller. It is developed in Rice University. Maestro supports multi-threading and targets researchers.

- Floodlight: Floodlight is Java-based controller including a learning switch and different algorithms for path finding. It also provides interfaces to OpenStack's Network-as-a-Service (Quantum).

- BigSwitch [10]: released a closed-source controller based on Floodlight that targets production enterprise networks. It features a user-friendly command-line interface (CLI) for central management of the network.

The controllers that we listed have different purposes and, consequentially, are implemented differently. They have different algorithms for path defining, learning and topology implemented and so have different advantages and disadvantages. The wide diversity of the controllers enables the users to choose in accordance with their needs by taking a controller out-of-the-shelf.

The OpenFlow controller is a single point of management of the network. Depending of the implementation, it can be equipped with software load balancer or firewall. Knowing the topology of the entire data center network enables easier traffic engineering configuration and QoS configuration. Another benefit of this kind of network management is the great number of management interfaces to the network together with the interfaces to the northbound APIs. Intelligence in the forwarding decisions, centralized in a separate controller has the potential for simplifying and shortening the time for network provisioning.

2.3.2.2 Problems

Unfortunately, there are some problems associated with the OpenFlow controller such as a single point failure. A network, where the decision making is concentrated in a single point, is vulnerable to Denial of Service (DOS) attacks. Another problem with the rich featured implementations is their scalability. Although it provides the elasticity in every network aspect, in cases where large number of new flows are created such controllers are slow [76].

In all the current implementations of OpenFlow networks, there is a single controller instance, however, a hybrid management model is possible by extending the controllers or the switch implementation.

2.3.3 Hybrid Approaches

As the OpenFlow-based approaches are showing performance problems in some circumstances, such as pre-populating of the flow tables, another approach is devised as an extension on the OpenFlow - hybrid network management. At the moment, the approaches that are defined as hybrid are in experimental phase [76]. They couple both of the methods just described. As a result of the migration efforts a working group of Open Networking Foundation (ONF) [34] is created. There is a general consensus over the long-term transition to the server-based virtualisation model with the hybrid implementation of SDNs and overlay virtualisation. Many of the naive hybrid implementation coincide with the proactive flow management according to [76]. Another possible problem with this methodology is multi-tenancy. In this case no real isolation is provided, therefore, the only way to implement multi-tenancy is by setting a different controller for each tenant, which adds significant overhead in management.

VXLAN Plus OpenFlow VXLAN provides also some control plane functionalities. The VXLAN control solution uses flooding based on Any Source Multicast (ASM) to acquire end host location information. That in addition to OpenFlow-enabled switches can provide a more extensive control over the network, thanks to the variety of the counters defined in the protocol. An experiment with this idea is presented in [59]. An example implementation of this concept is the Cisco's Nexus 1000V. It has implemented the Virtual Ethernet module (VEM), which runs in a hypervisor and is responsible for forwarding. Another VM has the Virtual Supervisor module (VSM), which implements the control functionality. [18] proposes a design of this OpenFlow-enabled switch and gives an example of network infrastructure that is hybrid in terms of management: The physical network is partitioned into many semi-isolated virtual overlays (implemented with the VXLAN technology). Each of this logical networks can be pragmatically managed with the OpenFlow protocol. In the described case, the VEM supplies each VM with a dedicated port, where for each logical overlay network a dedicated VEM exists. The set of these VEMs builds the distributed network switch, in this case implemented as a VSM. The VSMs support OpenFlow interfaces for orchestration on top of the VXLAN integarted physical and virtual networks.

STT and External Control Plane In the current release of STT, there is no control plane. Nicira [60] has not officially specified the control plane. However, a draft [8] on the next Niciras virtualization solution includes OpenFlow-like hypervisor vSwitches and a control plane based on a centralized network virtualization controller that facilitates management of STTs.

LegacyFlow Another hybrid implementation is proposed by [28]. The architecture consists of edge nodes, which are type of OpenFlow switches and are used for identification of the characteristics packet, and core nodes - legacy equipment used to provide connection among core nodes. There are two layers: the datapath layer and the switch controller layer. The datapath layer is the layer that enables communication between the OpenFlow datapath and switch controller. It provides access to different features of the legacy equipment by OpenFlow. The Switch controller layer is where for each legacy equipment specific configuration happens. Most commonly, the communication interfaces with the legacy resources are enabled by interfaces like Web Service, Telnet/CLI and SNMP.

CHAPTER 3

Related Work

This section presents solutions of the monitoring problem in other domains, such as server virtualization and traditional networks. The current monitoring approaches in overlay networks are also discussed. Another important point discussed is the correspondence of physical and virtual resources in server virtualization. Furthermore, we briefly discuss the problems that are related to the proposed solutions when similar approaches are applied in the domain of SDN. The problem of controller placement is also presented, since most of the research work on the current challenges in SDNs is focused on the controller.

3.1 Monitoring

A vast variety of different monitoring approaches exists in the different domains which are related to cloud computing. These methodologies differ mainly by their metrics and the placement of measurement sensors.

3.1.1 Traditional Network

Monitoring the network in an end-to-end manner is a very popular approach for gaining better network management and application performance. The process on which it is based is the monitoring the state of each single path in the network in real time.

3.1.1.1 Network Metrics Correlation

Changes in the network are mostly unpredictable, therefore, the best approach is to measure all the paths' metrics as often as possible. Unfortunately, due to the diversity of the gathered statistics and their specifics during monitoring frequent measurement may be time and resource consuming [95]. In contrast, a measurement that is not frequent enough may cause service and application issues. However, the targeted metrics in paper [95] have correlations between one another. For instance, route changes affect bandwidth capacity and available bandwidth. The

bandwidth is in direct relation to the application performance, so a conclusion can be drawn that the route changes usually affect application performance. Therefore, the authors are able to detect a route change and know that a change in the application performance is to be expected, so reasonable actions to prevent it can be taken in time.

They measure route change in the end-to-end monitoring by a trace-route tool. This only gives information about the change in the path. However, the trace-route often returns "*" values, and thus makes the measurement unreliable. Although inexpensive, the trace-route is not a robust way to measure the route changes. As a result, another type of measurement is needed. [95] identifies a correlation between the hop count change, latency and route change. Whereas the change in the hops always implies a change in the route the same is not valid for latency change. For the latency, a percentage of change in hop count is calculated and based on series of measurements a route change is identified with certain probability. Thus, measuring inexpensive and more accurate metrics as hops and latency can provide knowledge needed in the application level.

Another example is the relation between the route change and path capacity. The route capacity change is unambiguously defined, since it can be measured in two ways: first, by link-mapping and second, directly in percentage. Both methods are expensive and are processed by the *PathRate* tool. The relation between these two metrics will enable measuring of the inexpensive one - the route change and if and when a deviation is found, measuring the accurate value of the path capacity change. Unfortunately, their experimental results are not unambiguous and it is difficult to draw a fixed relation. As there is a trade-off between cost and accuracy of the measurement methodology, more often the route change is used.

3.1.1.2 Network Monitoring Methodologies

Centralized Apart from designing the optimal set of metrics for gaining the highest level of knowledge for the network, this thesis has to tackle other issues related to monitoring. An example of which is the monitoring station placement. In the traditional network, monitoring is done usually in different points, to prevent a single point of failure of monitoring. Some other specificities of the traditional network design require distributed monitoring. In the current situation, distributed physical monitoring is not wanted. The monitoring has to be performed in the OpenFlow controller, which means centrally for the network. A similar approach is presented in [3], where the metrics are aggregated and partially gathered in a single location.

There are two monitoring methodologies according to [3]: event reporting and polling. In the former, the monitoring station pushes information to the management module when a particular event occurs. The latter is a technique in which the management station sends requests for specific information to the sensors, usually at a given time period. The monitoring stations, on their behalf, send responses with the information for the particular time interval. In order to gain efficiency the monitored characteristics of the network are usually specified as a set of local properties on each network entity. Thus, a pull-policy can be defined in the management node or equivalent push-policy in the monitored points. Neither of the mechanisms takes the event continuity and frequency into consideration and, therefore, this may lead to inaccuracy of the monitoring results. The approach proposed in the paper [3] is a hybrid, in which distributed "dump" stations are combined with a centralized monitoring manager. In order to achieve this,

the events are divided in independent and dependent events. The independent events are unpredictable and there is no recognisable relation to other events or gathered metrics. The dependent events are a set of events that usually happen consequently. Based on the type of the events a cost function is calculated for querying the corresponding metrics either of the models. Their proposed scheduling and monitoring cost optimization is then based on these functions. The monitoring cost in this case is defined as message exchange for particular event.

Decentralized A similar approach toward monitoring is adopted in [9]. The authors propose a multi-domain architecture that is adaptable and configurable. From a high-level point of view this concept is similar to the traditional distributed monitoring approach. The paper addresses problems such as a *confidential domain requirement* and an *adaptive export process requirement*. The confidential domain requirement is related to keeping the monitoring process of the specified domain secret, it is a black-box for the other domains. This requirement is not exactly equivalent to any of the requirements in front of the SDNs monitoring. With the assumption that a single controller defines a domain, this requirement becomes valid in meaning that the other controllers may be implemented in a different manner and are not concerned with the particular monitoring process of the others controllers' networks.

The proposed solution in paper [9] is based on segmenting the network in entities. The administration entity is the only one for all the domains. Additionally, there are many element-management entities that are dedicated to the inter-domain measurements. The measurement point selection is done by the central entity, where it is ideally different each time for the different elements. Thus maximum accuracy is provided for the composed multi-domain metric. The selection process can be either reactive or proactive. Via the collaboration between all the measurement entities coordinated measurement configuration is achieved. This configuration can evolve in terms of parameters' presentation.

3.1.2 Monitoring in Server Virtualization

Approaches Another domain, which has special characteristics with respect to monitoring is server virtualization. There are many problems concerning the monitoring of the virtualized and physical resources in the context of the server virtualization. Server virtualization is not new concept and there are also many technologies that are time proven for virtualizing the server resources. Mainly it solves the problems of high diversity of loads on each physical server in different time slots. This approach promises better resource utilisation, but adds complexity in the management domain.

Server virtualization can be achieved basically via two different technologies: hypervisor-based and operating system (OS) level. The two technologies are compared and analysed in paper [71] based on implementations of Xen [94] and OpenVZ [66]. One and more physical nodes are used for the monitoring experiments. Each physical node hosts one or more virtual machines - supported by Xen and OpenVZ respectively. A monitoring sensor is installed on each physical node. Collection of monitored data is done via Oprofile [67]. Data is stored on a different physical node for later analysis.

The monitored performance metrics that the authors focus on include CPU consumption, and memory consumption.

The measurement of the CPU specific metrics is achieved by a custom script based on provided Linux instrumentation:

```
top -b
```

respectively on Xen:

```
xentop -b
```

The other tool used is Oprofile, which is based on hardware counters gathering. Oprofile measures the number of cache misses that happen in a particular application. Another metric that is considered essential is the number of cycles outside halt state. And the last one is the number of instructions that are retired.

Physical	Virtual	Evaluated instance
one	two	one
two	two	one
two	two	two
two	N	$N/2$

Table 3.1: Physical to Virtual Relation and the Monitored High Level Instances Number

In the designed experiment in [71] for evaluating the performance of application, two virtual instances are created respectively for the Web and the DB tier. Basically these are two virtual entities but their metrics are aggregated and transferred to the overall application performance. Therefore complex test cases are designed, summarised in Table 3.1

Metrics Another approach for managing the virtual machines based on specified metrics is proposed in paper [46]. The focus here is on the performance during a VM migration rather than on the application. According to the discussed in the paper [46] testbed implementation and measured scenarios the relation between the physical and the virtual resources is always 1:N for the virtual instances.

The next important point in the paper is the metrics definition. According to the number of gathered metrics each VM is a d-dimensional vector, where d is the number of metrics. In the presented setup only three metrics are monitored: CPU, RAM and I/O. The three of them are not concretely defined. Notable is also that the monitoring is done by standard monitoring tools, similar to the approach applied in the paper discussed above.

Together with the server virtualization, the storages are also virtualized to become a building block of the current cloud architecture. In [81], there is an algorithm of optimising the resources with respect to the CPU and throughput between the storage entities.

Monitoring is recognised as an essential part of server virtualization implementation in the enterprises. It brings a number of benefits, such as troubleshooting, used for automated resource provisioning and SLA prediction and enforcement. Last but not least, it is used for cost and

energy savings. However, still in most of production-ready virtualization monitoring tools the number of metrics proposed for the corresponding virtual resources is limited.

Special attention needs to be drawn to the monitored metrics. Metrics that are monitored by most of the existing tools are summarised in Table 3.2. As they become fine-grained and

Physical	Virtual
CPU Utilization	CPU consumed by virtual machines, as well as the CPU utilization of the cores
Memory Utilization	consumed memory, active memory, overhead memory, shared memory, granted memory, reserved memory
Disk Usage	storage devices with high activity and the amount of free space available in each of the disk partitions
Network Utilization	health and status of the network interface
Hardware Metrics	depends on the tool

Table 3.2: Composed Metrics Fine-Grained Metrics for Server Virtualization

more frequently measured, the end-user experience also improves. This consideration can also be directly applied to the network virtualization monitoring.

3.1.3 Network Monitoring Metrics in Grids

We have been referencing to grids in Section 2 as predecessor of cloud, which is why we examine also the problems related to network monitoring in grids.

Grid computing is a similar concept to cloud computing. Therefore, analysis and experimentation in the area of network monitoring in grids is very interesting to our work. Monitoring in grids is supposed to bring benefits, such as identifying problems and fault detections. Monitoring provides information to the grid systems that is needed in order to make decisions about the resource utilization and performance. [5] discusses production network monitoring. It presents the different types of metrics that are to be applied in the grid environments and how they cooperate in an implementation of an overlay network infrastructure.

The monitoring approaches in grid systems are reasonable to be distinguished according to two different characteristics - according to network granularity and according to network behaviour. When characterising the network monitoring approaches based on granularity, two basic methodologies can be applied: by *link* and by *path* monitoring. The authors note that the link monitoring strategy does not provide the needed information for the utilisation of the end devices, and thus gives only partial knowledge about the network. In contrast, path monitoring is not scalable and its complexity grows quadratically with the increase of the network resources. As both approaches have different problems that make them not optimal in grids, [5] proposes a mixed approach that the authors called *domain-oriented*, in which in the domain, the network is monitored via path and between the domains via link monitoring.

Another characteristic of the network monitoring is the type of information concerning network behaviour. Monitoring network behaviour can be *active* - application specific network information, and *passive* - connectivity information.

Passive monitoring is implemented using passive sensors located in specified places in the network such that they gather information for node, the single link and different domains. In the grid system the passive network monitoring is more preferable, because it does not add overhead to the underlying infrastructure. In comparison, the active network measurement is accomplished by "probing" - injecting test traffic and then measuring the deviations. This burdens additionally the network infrastructure of the grid and could cause failures. The classification of the proposed metrics is summarised in Table 3.3.

	Passive	**Active**
Single Point of measurement	Network-level Round-Trip Time	Round-Trip Time
	Throughput	
	Retransmitted Packets	
	Packet Reordering	
Multiple Point of measurement	One-Way Delay and Jitter	Timing of packet stream
	Packet Loss Ratio	Length of packet stream
	Service Availability	

Table 3.3: Metrics Classification Based on Sensors Number and Tool Behaviour

3.1.4 Performance Measurement

The high-level goal of monitoring in all kinds of services is to provide the end-user with the best possible performance at the lowest possible price. In cloud computing, achieving this goal is a complex task. The cloud computing stack itself is complex and consists of many different parts that need to be monitored, both separately and all together. Separate monitoring provides enough information for the changes and faults in the infrastructure, whereas monitoring the service all together gives information about the entire service, which is relevant to the end-user experience.

3.1.4.1 Goals

For many enterprises' IT departments, cloud computing mainly includes outsourcing server provisioning, system administration or software development in order to save costs. However, choosing the provider that would overtake these responsibilities needs to be based on the quality of the provided service. This implies that performance measurements need to be done by both sides: the provider and the client. The implementation of monitoring on client side is not of particular interest to our work; instead we mainly concentrate on implementations for the cloud provider.

3.1.4.2 Implementation in Clouds - SLAs

Performance measurement in the cloud includes different number of metrics, depending on the services and on the level of security agreed upon between the customer and the provider. [93] summarises the main types. Firstly, special attention is drawn on *patching*. It is important for the client in SaaS and PaaS when a patch is applied to the application or the operating

system. Although it is an inevitable event, it is a main part of building user satisfaction and also depending on the agreement, client defined schedule may be extra charged. The same applies to unplanned maintenances. Vulnerability identifications are also highly graded by the users. Every state-of-the-art cloud provider needs to be equipped with vulnerability management program. Thus, any kind of security issues can be predicted and then the clients can be protected. According to [93] the same applies to security incidents and vulnerability remediation.

Another aspect of performance monitoring in the cloud is application level performance. Especially interesting for our research are network intensive applications. Some research on the topic of performance monitoring on the network intensive applications is presented in [56]. The proposed metrics are summarized in Table 3.4.

Metric	Description
Server throughput (req/sec)	Maximum number of successful requests served per second when retrieving web documents
Normalized throughput	One measured throughput as baseline reference throughput and normalize the throughputs of different configuration settings
Aggregated throughput (req/sec)	Impact of using varying number of VMs on the aggregated throughput performance of a physical host.
CPU time per execution (s/exe)	Average CPU time in microseconds during each run of the given domain
Execution per second (exe/sec)	Number of guest domains being scheduled to run on a physical CPU .
CPU utilization (%)	Average CPU utilization of each VM, including Domain0 CPU usage and guest domain CPU usage
Network I/O per second (kByte/sec)	Amount of network I/O traffic in kB per second, transferred to/from a remote web server for the corresponding workload.
Memory pages exchange per second (pages/sec)	Number of memory pages per second in the I/O channel; efficiency the I/O processing.
Pages per execution (pages/exe)	Performance indicator for average memory pages exchanged during each run of the given domain.

Table 3.4: Metrics in Network Intensive Applications

The network connectivity is crucial for the performance of the IaaS cloud providers and therefore, needs to be carefully monitored. The main problems that standard network monitoring tools face are scalability, dynamic configuration and accounting of the underlying infrastructure. [19] features a detailed discussion of the current implementation of the network monitoring in cloud and in grids. The authors focus on cloud overlay networks implemented using VLANs and grid legacy networks. The simplicity of the provided infrastructure is an assumption. As monitoring the traditional network takes advantage of the results of traffic sensor measurements,

in the discussed overlay networks there are persisting questions: how to adhere those results to the VLAN abstraction and how to provide those abstracted metrics to the network aware applications.

The tool that is claimed [19] to be widely used for monitoring the network performance across many network interface devices is *libpcap*. Unfortunately, in the overlay networks, these instances are bundled with VMs. In order to get any results out of *libpcap*, it needs to be configured by each user and thus the provider loses the results' validity. On the other hand, if the user wants to monitor the network, then it would not be possible, because monitoring of the intra-provider devices is not acceptable. The providers themselves usually monitor these devices (physical devices) using SNMP. However, the traffic between the physical servers is truncated and the links are aggregated so monitoring of the user traffic needs to be placed near or within the user-end points. This limits the measurement approaches to only using passive host monitoring.

Another point is that network monitoring needs to be as scalable as the network itself. This implies that on each VM instantiation, a respective network monitoring entity also needs to be instantiated. For that reason, the network monitoring should be distributed. However, it also needs to provide both user and administration interfaces, which ideally provide central knowledge. It is also challenging to maintain consistent description of the physical and virtual network simultaneously. The proposed software module in [19] is concentrated on providing the VMs with a virtual interface to such a software monitoring tool. This interface consists of two parts - the first one is participating in the network infrastructure implementation and accessing the real network resources, whereas the other is providing an abstraction of the virtual network through an API.

3.2 Physical - Virtual Correspondence

3.2.1 Goal

From a high-level point of view, the cloud infrastructure in a single provider it is not interesting to the client, how the cloud infrastructure is divided into physical and virtual resources, as long as it manages to deliver the agreed upon services. However, the structure of the resources becomes important when we consider administration and management. The ability to predict the results of a change in one of the layers (physical or virtual) in the interoperability of both is the basis for the progress in the automation of the resource management.

3.2.2 Server Virtualization

The relation between the physical and virtual resources takes a central place in any kind of virtualization. When we talk about SLAs the correspondence between the metrics that build the particular SLA and the low-level physical metrics is also significant.

[26] proposes such a mapping. The framework presented is based on the MAPE (Monitoring, Analysis, Planning, and Execution) cycle and is built out of two components - a monitor and an enactor component. The paper presents a distributed approach for host monitoring, that utilizes broadcasts. Thus a management system is built, where each host has a complete set of measurements, which ensures fault tolerance in the system. Another point is mapping the

SLA parameters out of the physical metrics. The mappings are divided into two types: *simple* and *complex*. Simple mapping is applied to parameters that are one-to-one to the physical metrics (storage - disc space), where the complex needs more sophisticated calculations based on predefined formulae.

3.2.3 Virtual Networks

Mapping To Physical Resources As the virtual networks promise to bring a great number of benefits to the industry, the interest in them grows. There are many obstacles, however, that prevent virtual networks from wide implementation in the production networks. One of them is the efficient embedding of virtual infrastructure in the existing physical network [61].

Not only do the virtual nodes need to be optimally placed over the physical, but link calculation and optimizations are also needed. That requires a great computational power and, therefore, becomes a stopper in the way of virtual networks to production environments. It is a new problem in science and in industry, and that is why there are few attempts to solve it. [61] proposes an algorithm that calculates the best virtual network placement in relation to the heterogeneous physical resources (nodes and links). Both types of resources impose different limitations for the virtual network: links - delay, bandwidth; nodes - CPU and RAM. Therefore, the algorithm provides different stress formulas for each type of resource. For the nodes, the stress formula consists of the number of the VMs running on it, where the link is based on the number of virtual hop count and link bandwidth. The algorithm is based on an heuristic using Constrained Shortest Path First (CSPF). It first calculates the stress of each network entity. The entities with greater stress are considered less likely to receive more virtual resources. The output of this step of the algorithm sets the basis for the virtual resources. For a specified virtual resource a set of candidate resources is extracted. The CSPF Dijkstra algorithm is applied on the set of output values, and a single physical resources is mapped to the specified virtual one.

From the algorithm evaluation, the following conclusion is drown in [61]: as the number of virtual network nodes increases, increases the link stress, but the node stress decreases. In many cases this may lead to deceleration of the algorithm processing. The feasibility tests are made with an average number of nodes (15), which is appropriate for private cloud implementations but far under the limits of datacenters of the scale of Google and Amazon. The results show that the link capacity becomes a problem when increasing the number of virtual links. This imposes a problem when mapping virtual resources on the network that already have a great number of virtual networks mapped. Eventually this may lead to virtual resources saturation on the physical ones. Although the algorithm is fast and feasible in large-scale networks, it is useful in the defined ranges of nodes and links number.

Additionally, the algorithm always considers the adding of virtual resources from scratch, which means that in specific moments, changes in the already implemented mappings are possible. In the algorithm no “separation“ of the virtual resources is taken into consideration. This limitation of the algorithm can prevent finding a solution to the link capacity issue by simply aggregating many physical links into a single virtual. The algorithm itself does not take into account any real-time network measurements. As the stress calculation is different for the different types of entities, and they are observed entirely isolated from one another, the calculation will not ensure the network convergence. The algorithm calculates the mappings based on the static

parameters of the underlying resources.

Monitoring In order to be as productive as possible, deeper levels of visibility are needed in the virtual networks, which is typically achieved via different monitoring technologies. A commercial solution about the correspondence of the physical to virtual network switches is presented very briefly in [2]. Generally, it is based on an integration of a network monitoring switch that is responsible for measuring physical switch performance of a particular number of switches. A part of the solution is developing mechanisms for load-balancing and gathering data aggregation on the switch. The network traffic recorded along with the network analyser is a helpful feature that promises to provide the enterprise with visibility of the network as a layered structure with strict relations. This structure is outside the actual implementation of any kind of network virtualization paradigm. It relies on gathering knowledge and on mimicking the virtualization mapping. Although promising a lot of features out of the box, still a great deal of analysis and manual mapping of resources needs to be performed.

It is also worth mentioning that monitoring is de-facto performed outside the application boundaries, which means that the applications on the higher level of the cloud stack are not able to benefit from it. The relations, that are presented would neither help in the optimisation of the resource usage for either of the applications nor will they benefit the VM migration or instantiation.

3.3 SDN - Controller

SDNs are new a topic in science and industry, therefore, there are only a few solutions related to them. Even less are these are production-ready. Among the few aspects of the SDN that have been studied so far is the controller placement problem. The two components of this problem are determining the concrete location and, in some cases, the number of controllers that are necessary. The relative location of the controller to the other network devices can influence the network behaviour greatly. The number of controllers that manage the network is also an important factor in SDN implementation. The implemented solution to this problem influences every aspect of the virtual network - from fault tolerance to performance metrics.

The impact of placement on real topologies is discussed in [41]. For measurement with different placements, the authors use the following metrics:

- Average-case latency
- Worst-case latency
- Nodes within a latency bound

The authors present two types of experiments: firstly with the same topology, but with different controller locations, and secondly, with the same topology but a different number of controllers. The experiments related to the controller location start with placing the controller where the nodes density is the highest. Afterwards, an optimisation algorithm places the controller optimal according to the average-case latency or to the worst-case latency. The other experiment

evaluates the results of the same topology managed by up to five controllers. The authors also provide a brief discussion over a hypothetical behaviour of the the experiment setups over different topologies.

The conclusions drawn from the experiments show that the placement of the one or more controllers depends on the topology itself, as well on the number of the controllers. However, the experiments show that for middle-sized network (nodes <30) one controller performs better than more, where for greater number of nodes it is preferable to use more controllers. For one controller the average-latency is the lowest as well is the worst-case. Whereas for the five controllers, the authors state to have measured two times the worst results for both metrics.

For topologies that are mesh it is best to place the controller in the center (where the node density is the highest), and in the case when more controllers are needed, they also need to be placed optimally around the central place. What is more, the factor of difference between most random and optimal placement increases faster with increasing the controller number, which also shows that placing great number of controllers in the network is expensive and error-prone.

3.4 Relevance To This Thesis

3.4.1 Network Metrics Correlation

[95] shows the correlation between the hop count change, latency and route change. It presents the relation between the change in the path-oriented parameters and the change in the application service quality. The considerations, discussed earlier, are very important for this thesis because in the working area, the end-to-end paths are virtual. Hence, it is impossible to monitor the metrics concerning each path at the same time. Therefore, a way to derive them from the available network metrics is needed. Measuring the current and the negotiated link throughput is a part of the targeted metrics, and calculating the available capacity is to be achieved based on the considerations presented earlier. The capacity as a metric is not considered in the targeted measurements. Nevertheless, it is tightly connected to the current load on the interfaces, so the considerations can also be applied to the load solution. Here is also important that the two scenarios are conceptually different. In the paper the capacity is the end-to-end bandwidth capacity, whereas in the thesis the capacity is a single link capacity. Therefore, in this case the capacity is related to the interface loads. Measuring the switch and port availability and further predictions about approximate path latency are also possible based on these considerations.

3.4.2 Centralized Network Monitoring

The problem solved in [3] is similar to the problem of optimal monitoring of the network from a centralized point. The proposed event classification (based on dependencies between events) is possible to be applied to SDN environments, but is not considered effective. The authors note that events in the physical network do not always trigger corresponding events in the virtual network. Therefore, having the “raw“ physical metrics is more appropriate. What concerns the monitoring cost, in the case of the OpenFlow controller, is that the additional monitoring messages add overhead to the overall management capabilities of the network. As a result, there

is a need for establishing a reasonable trade-off between the additional monitoring messaging and provided knowledge.

3.4.3 Decentralized Network Monitoring

Another approach that is relevant to this thesis is the multi-domain monitoring described in [9]. The problems that build the focus of the article can be related to the SDN only under the assumption that the controller builds one domain. A notable design decision discussed in [9] is the creation of a single administration point dedicated only to monitoring. This scenario ensures central knowledge about the network, but not central management. In special cases this may lead to inconsistencies and delay in getting the monitored parameters and, consequently, in responding to network changes.

The second problem is the adaptive export process and is not relevant to our research. Due to the specificities of the chosen OpenFlow controller, the export can be achieved in two ways, and both are implemented. The concept of adaptable measurement parameters is, however, considered useful for any cloud controller. Its implementation involves changing runtime the metrics measured for each device, which is not possible.

3.4.4 Server Virtualization Monitoring

Some of the approaches used in [71] in the server virtualization context are also applicable in solving the similar problem in the network virtualization domain. Experiments built on consistent correlation between physical and virtual resources are a useful approach for evaluating virtualization technologies in case of both network and server virtualization. Gathering metrics using build-in tools such as `top` and similar is also considered an efficient approach. However, with respect to network virtualization performing the monitoring, analysis and responding in near real time is essential, hence, gathering metrics on different machine is not optimal for SDNs.

As discussed earlier, according to [81], storage virtualization also has problems with optimising the resources with respect to the CPU and throughput between the storage entities. The most intriguing part of this paper, from thesis' point of view, is the evaluation and the testbed setup. The concept of validating the results by one and then by many physical servers is also applied. Similarly to the others articles, the approach is based on standard monitoring tools and non-event based metrics. The gathered metrics are used to calculate a load function, which then is used for analysis and subsequently for further automation of load-balancing.

3.4.5 Passive Monitoring in Grid

As cloud is related to the grid computing, the aforementioned research in area of network monitoring in grids is interesting for this thesis in terms of metrics and monitoring methods. The discussed measurement methodologies in [5], are exposed to a number of security threads, mainly DOS attacks. It is also important that there is a trade-off between the metrics' accuracy and the measurement overhead introduced in the environment, which eventually will also be the case in monitoring SDNs in cloud. The number of the defined metrics is also a scalability bottle-

neck in the measurement infrastructure, because they need to be monitored by passive sensors, which require manual deployment and configuration. We consider, for the purposes of this thesis that any manual configuration of sensors is no optimal and will make the monitoring process error-prone.

3.4.6 Physical to Virtual Mapping

3.4.6.1 in Server Environment

An approach concerning the mapping of low level physical metrics to the high level SLA-relevant ones is discussed in [26]. For our research, the two most interesting moments are the host monitor and the metrics mapping. The host monitoring is interesting because the distributed approach utilized using broadcasts. Whereas distributed monitoring is not possible in the current state of the SDNs, mapping of the physical resources to the virtual by classification of mappings in simple and complex is applicable and relevant to this thesis. However, the main concept in the article is providing better and most accurate automatic SLA management in clouds, which is the high-level purpose of similar mapping in the software-defined datacenters.

3.4.6.2 in Overlay Networks

One of the few existing algorithm aiming to solve the placement of the virtual resources in relation to the physical infrastructure is presented in [61]. As already discussed, the algorithm is evaluated based on limited number of nodes. Related to the algorithm proposed in this thesis, mapping the results of this algorithm are always in the 1:N or 1:1 (physical:virtual) case. In this thesis, the mappings are a result of the network monitoring on a low level under different traffic conditions. Nevertheless, the achieved conclusions about the link capacities are taken into consideration in the proposed mapping implementations.

3.4.7 Controller Placement

[41] focuses on the controller placement problem. It proposes evaluation of the optimal number of controllers and their optimal placement in the network. We consider that the conclusions drawn from this research depend greatly on the metrics chosen by the authors. In case of different metrics for evaluating the experiment, the optimal number and placement of the controllers are most likely to be different. Also, the experiments focus on a network of middle size, therefore, the conclusions are likely to be not relevant for large-scale datacenter networks.

CHAPTER 4

Design

In the following section, we provide clarification of the conceptual design of all parts of this work. The section consists of three main subsections: firstly, we motivate the idea of the thesis with a hypothetical example; secondly, we present the basic Floodlight architecture and the design of the plug-in and, finally, we clarify the physical-to-virtual resource mappings design. The conceptual ideas of the plug-in and its incorporation in Floodlight are main focus of this section, and the implementation details will be explained later in Section 5.

4.1 Google SDN Example

In the middle of 2012 on the *Open Networking Summit* [84] Google has introduced its implementation of a SDN strategy for operating its large network infrastructure. The company claims to have its largest production network operating on OpenFlow and to be with improved manageability. After Google has announced using SDN to implement its data center networks, and to connect one another, many of the other big players in IT industry began to investigate SDN as an opportunity to optimise their networking costs as well.

The main reason for the SDN to be introduced for Google is to simplify the backbone management and costs. Another reason is to solve other problems such as lack of fidelity in the test environment, inability to emulate in software large backbones and inflexible end-to-end path management. Their current infrastructure consists of two large networks: Internet-facing backbone (user traffic) and datacenter backbone (internal traffic). In the middle of 2012, both are rolled out on centralised traffic engineering via optimized routing based on application-level properties and globally optimised flows. Additionally, a proprietary implemented scheduler is responsible for implementing deadline scheduling of large data copies through an OpenFlow controller.

As a motivating example let us assume Google is operating over a Floodlight controller and has also a proprietary scheduler implementing network configuration of VM migrations. Ideally, the network is already separated respectively for all different tenants (in this case Google services). Each tenant is able to configure its own virtual network and, eventually, to set QoS

policies. At a certain point, two VMs are identified to build large traffic between each other, according to measurements in the virtual network. Both VMs are placed on different virtual switches, which are mapped to more than two physical devices geographically placed in Europe and in the USA. Thus, the traffic between the two VMs utilises the physical datacenter backbone. Due to the data gathered for the VMs' traffic, the proprietary migration scheduler is able to choose the best possible physical-virtual network resource's mapping existing for the defined VM and then to migrate it. Moving one of the VMs from one datacenter to another requires first on-demand migration, accomplished by the hypervisor and on the networking part only redirection of all its flows. The reconfiguration of the virtual network is easily done by one single machine, without cabling and manual provisioning. After VM migration, all physical devices executing the VM flow are located in Europe and are at a hop distance that is as short as possible. The end-user benefits are faster speed between the two particular VMs, and better service for the other VMs. Utilisation of the datacenter backbone is lower in comparison to the traffic utilisation of the interdatacenter network, where the cost per bit/sec are lower. In this particular case, the overprovisioning of the physical network is minimised, due to the centralized monitoring of both network abstraction levels.

4.2 Floodlight Controller Architecture

The Floodlight controller is an implementation of the SDN paradigm using OpenFlow. As such, it consists basically of two parts: an implementation of OpenFlow protocol, and a custom implementation of the flow management system. As the implementation of the OpenFlow protocol does not affect the design of the proposed plug-in, we will not discuss it in this section; we will rather provide brief implementation details on it in Section 5.1.2.

4.2.1 Architecture

The functions of the controller are setting not only functional but also a great number of non-functional requirements in front of the Floodlight's architectural design. Ideally, the controller's function should not be interrupted, so it needs to be hot-pluggable. Its performance is crucial for achieving the controller's goals. Fault-tolerance is another important requirement in order to the controller to function properly. Also important for the controller's function is security. Fidelity and containment are also very important requirements for the Floodlight's design.

Together with the non-functional requirements, the controller's basic task is to control the virtualisation of the network through flow management. For the purpose of fulfilling this set of requirements, as well as for achieving separation of concerns, the controller's creators have chosen a component-based architecture. Components are called "Modules" in Floodlight and are mainly designed according to `Observer` pattern [37] [11]. Another architectural decision, aiming at loose coupling between the modules, is using REST interfaces for almost all data interactions.

As Figure 4.1 shows, the Floodlight architecture is component-based. One part of the components build the core of the controller. The other part (the three top components in the figure) is responsible for discovering and listening for OpenFlow switch messages. The different com-

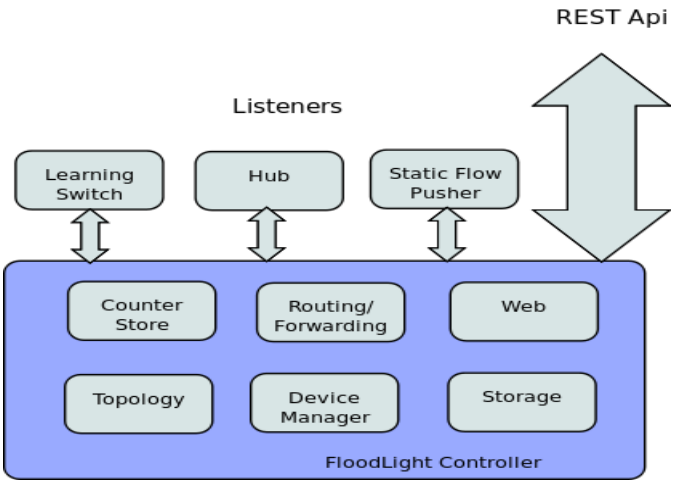

Figure 4.1: FloodLight Conceptual Design

ponents are independent from each other, but they all have same structures in order to meet the standards of the Floodlight module activator. All core components build up the REST interface that provide interface for manual management of the controller itself.

4.2.1.1 OpenFlow Protocol Messaging

The OpenFlow's protocol messages are of three types: controller-to-switch, asynchronous and symmetric [65]. The messaging mechanism [38] in Floodlight is designed according to messaging patterns described in [42]. The solution to separate messages' functions according to their type and sender will be utilised by the design of the proposed plug-in in order to retrieve useful, but not flow-related, information from the switch.

- Controller-to-switch: Sent by the controller, they are used to specify, modify or delete flow definitions, as well as retrieving all kind of switch capabilities and counters.
- Asynchronous messages: The switch sends message when new packet does not match existing flow or on flow remove, or is timed out or on error on the port occurred.
- Symmetric messages: Both the controller and the switch send each other on-switch-startup messages or echo messages.

4.3 Plug-In Design

A number of factors need to be considered in the analysis and in the design of the proposed plug-in. A first step in analysing the plug-in is the definition of the functional requirements. As we have seen in the motivating example, the main function of the plug-in is to identify changes in the virtual layer and to enable us to track them respectively to the physical resources and tenants' ownership. The results need to be made available to the higher level controller as well

as to the administrator in a suitable way. This function need to be carried out in time frame that is small enough to enable accurate and fast feedback.

This section breaks down the design into smaller parts, so that the different problems and specificities can be discussed one after another.

4.3.1 Metrics Analysis

Identifying traffic changes in the conventional networks is done by monitoring a particular set of network metrics. As part of our goal is to monitor the physical network resources, the metrics design starts with analysing which metrics similar tools (such as Ganglia, Cacti, Nagios, etc.) are using.

In conclusion of the monitoring tools analysis, we summarized the following physical metrics:

- Host passive
  - CPU utilisation - percentage of used CPU resources in the physical device
  - Memory utilisation - percentage of total used memory
  - availability - ratio of time the host is functional to the total time it is required to function
- Network passive
  - packets per second per port - number of transmitted and received packets on a specified port for a second.
  - packets per second per switch - number of transmitted and received packets on all ports of a switch for a second.
  - broadcasts per second per port - number of transmitted and received broadcast packets on a specified port for a second.
  - broadcasts per second per switch - number of transmitted and received broadcast packets on all ports of a switch for a second.
  - unicasts per second per port - number of transmitted and received unicast packets on a specified port for a second.
  - unicasts per second per switch - number of transmitted and received unicast packets on all ports of a switch for a second.
  - errors per second per port - number of transmitted and received error packets on a specified port for a second.
  - errors per second per switch - number of transmitted and received error packets on all ports of a switch for a second.
  - bits per second per port - number of transmitted and received bits on a specified port for a second.

- bits per second per switch - number of transmitted and received bits on a specified switch for a second.
- load per port - the percentage of the traffic rate used out of the maximum in defined time period on particular port. Monitoring the port load is useful not only for prevention of eventual oversubscribing, but also for more efficient traffic engineering, and therefore, optimal switch resource usage.
- load per switch - the share of average port load out of the maximum switch speed in percent. This metric is used for monitoring the overall switch health with respect to traffic.
- port maximal speed - as the switches support different types of speeds (advertised, negotiated, supported and peer), the maximum speed of a port is the maximum value of the four types of speeds for a defined time period. Knowing the maximum available traffic rate helps for more accurate port utilisation calculation.
- designed switch maximal speed - for the purposes of this work this metric is defined as the average maximum speed of all switch's interfaces. This metric is useful for calculating the switch load and bandwidth utilisation on high level.

Other metrics such as latency, jitter and other active metrics are considered not relevant, as the end resources are practically non-existing because of the network virtualisation. The metrics analysis is continued by an analysis of the OpenFlow specific metrics. The OpenFlow protocol defines limited number of metrics currently targeting the controller's performance and description of switches. The relevant metrics according to OpenFlow Foundation [65] are:

- Controller
  - connections per second - number of connections established per second in the controller. This shows delays in the controller itself. Connection is defined as the five-tupple:

    $$\{sourceIP\,address,\ destinationIP\,address,\ source\,port,\ destination\,port,\ protocol\} \tag{4.1}$$

    The metric connection per second (c/s) is the rate at which the controller can create the parameters for the new connection.
  - flow arrival rate - rate at which the unmatched packets arrive in the controller for a unit of time. In general, this shows the diversity of flows in the particular network and can be used for predicting how the traffic changes, and, respectively, for performing optimisations in the controller itself for traffic engineering.

    $$AR = \frac{PAK\_IN_n}{\Delta t} \tag{4.2}$$

    where PAK_IN_n is the number of received packets of the Δt time
  - flow setup time - time in seconds for processing packets that cannot be matched in the switch. This is a key metric for the controller performance evaluation.

- Switch
  - maximum concurrent connections - total number of connections for which the device can maintain state simultaneously.
  - number active flows - the number of flows in the switch that are currently processed. It is closely related to the flow arrival rate in the controller, but also interesting for calculating the switch memory usage.
  - flow entry usage - percentage of hits (matched count) in the switch. This criterion is important for predicting and managing the flow table in the switch and respectively the switch performance.
  - match count - number of hits by new flow establishment in the OpenFlow table.
  - transactions per second - complete action of a particular type that can be performed per second. This metric also helps to identify latencies in the messaging processing, either in the controller or in the switch.

It is worth noting that although these metrics are defined in the OpenFlow specification [65], some of them (e.g. match count and transactions per second) do not exist in the FloodLight controller implementation. Neither of the metrics just mentioned provides information about the traffic through the switch. Hence, engineering and calculating metrics that provide information for the traffic on this level of abstraction is necessary. Bits per second per flow and packets per second per flow are the most suitable. As we have the traffic information from the physical metrics and the description of the flows from the OpenFlow metrics, bits and packets per flow can be calculated in the plug-in. Incremental count of the bytes per flow is included in one of the controller-switch message replies, so it is possible within a continuous measurement of this value to calculate the actual bits per flow. Unfortunately, a counter for packets is not implemented in all OpenFlow enabled switches, so the metric packets per sec per flow is not always available. We will have, on the one hand, the traffic rate per flow, and on the other hand, the traffic rate per port. The plug-in is designed to be dependent on two other modules of the Floodlight, and therefore, we have extracted and mapped data, so we know the flow location on physical interfaces. This ensures accurate and robust monitoring of both metric types.

4.3.2 Physical Devices Communication - SNMP

Analysing relevant physical metrics raises a few questions. How can one extract all metrics from the virtual resources, if the OpenFlow protocol is partially implemented for Floodlight? How can one retrieve any metric from the physical device, if the controller itself does not have a permanent management connection to the underlying physical resources and the switch IP information is encapsulated in particular module? Open vSwitch normally comes with NetSNMP installed on it, this enables monitoring from an external tool using SNMP, and we are using this feature to implement monitoring inside the controller. Other reasons for choosing SNMP are its reliability, consistency as well as its compatibility with the other tools, which use it in analysing the metrics.

Implementing it together with the other part of the monitoring is not an efficient design, therefore, another module for maintaining SMNP communication with the physical devices is

designed. An independent SNMP module as a design decision preserves the separation of concerns, as well as keeps the current architecture clean. For performance purposes, the connection to each switch is established on data request, as no additional iterations for establishing connections are needed. Maintaining a large number of open connections is considered inefficient.

For simplicity, in this module no *Object Identifiers* or IPs are stored and handled. This functions is delegated to the monitoring module. The SNMP module provides interface for SNMP commands, as well as to other modules as direct invocation service. The implementation of this service follows the `Proxy` design pattern.

4.3.3 Architecture

The main functional requirements imply reliable and transparent communication with the underling physical devices, as well as incorporation in the current controller function. In addition, the performance is significant. It is required to perform the task of measuring concrete metrics without slowing down the overall function of the controller and adding overhead to the flow processing. Another key issue is the designing of interfaces for the data retrieved and how it can be stored. Managing the connection with the physical devices and discovering new ones while collecting the corresponding flows is also considered to be a difficult task with regards to data cashing.

From architectural point of view, the design of the proposed plug-in is modular. It also uses a the `Observer` pattern in order to have access to OpenFlow messages, but in the same time has singleton manager for each device statistic. Considering the large amount of data that is gathered periodically, only part of it is cached in the plug-in. Defining a threshold for the amount of time for which the data is cached is defined after a series of experiments. A reasonable trade-off between having very fast access to the data and slowing flow processing needs to be found. The older statistics are logged in human and machine readable format.

Figure 4.2 shows the conceptual design of the plug-in and how it relates to other components in the system. The plug-in has four main services. Two of them retrieve the statistics from the sources - OpenFlow messages and the actual devices through SNMP, respectively. The service responsible for switch handling is the *Switch Store*. It intercepts events for learning new switch and deleting switch. All information from the *Device Manager*, the *Topology* and from the *Virtual Network Filter* are handled there. The manager module is the orchestrator, it also controls the functions of the other three parts and intercepts any statistics request from the cloud controller.

Due to the specificities of the Floodlight implementation discussed in Section 5, the *Switch store* is designed following the `Singleton` pattern. For this reason, the switch reference data is consistent and up-to-date despite of the fact that it is editable from the three other modules. In Floodlight, the information about physical and virtual devices is encapsulated in different modules, that are independent and also triggered differently, which is why it is inaccessible from the other modules. As a result, the way we gather the information is by providing one and the same instance to those modules and store their data simultaneously. There we are able to extract relations between the virtual and physical resources with the help of timestamps, IPs and port mapping. The switch information used for SNMP requesting is also managed in the *Switch store*.

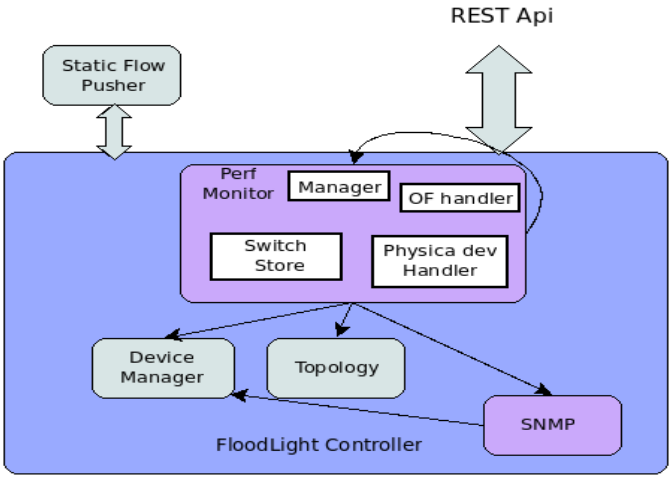

Figure 4.2: FloodLight Plug-In Performance Monitor Conceptual Design

4.4 Physical-Virtual Resources Mappings Design

The evaluation of the proposed Floodlight extension is done in a series of experiments with different physical-to-virtual network resources mappings. The main goal of the experimentations is to suggest a scenario that would eventually behave optimally in a cloud environment.

Slicing the network infrastructure in abstraction layers, we end up with a physical layer, where the forwarding happens, and the virtual layer above it, dedicated to controlling. The virtual network is abstracted due to the controller. Looking at the network from the cloud controller point of view above the virtual network, there are different networks, separated according to the needs of each tenant. At the top abstraction level any cloud provider needs to fulfil SLAs on different client actions, such as VM initialisation and migration, application traffic requirements and others. Another point is that fulfilling those SLA has to be done by using minimal resources in the datacenter. Therefore, provisioning the tenants' VM sequentially on each switch or in a round-robin manner is neither cost- nor SLA effective. The key to building robust and reliable infrastructure is knowing how to scale and provision the virtual network with regards to the physical resources

Going back to our motivating example, let us assume a new VM needs to be set up. Let us also assume we have three virtual switches, each with four interfaces connected in full mesh and two tenants with defined networks, and each owns one VM. Intuitively, the best solution is to define a network connected on the third switch and place the new VM on one of its interfaces. The physical network, on the other hand, is based on two switches with twelve ports. This is the situation where we have 1:N physical to virtual mapping. Considering the expected traffic if we know what is the best proportion between the two kinds of resources, when adding the VM we will be able to provide best possible conditions for it. Let us say that for the expected traffic range the 1:1 mapping is best suited, then we probably would want to add new physical switch to the network.

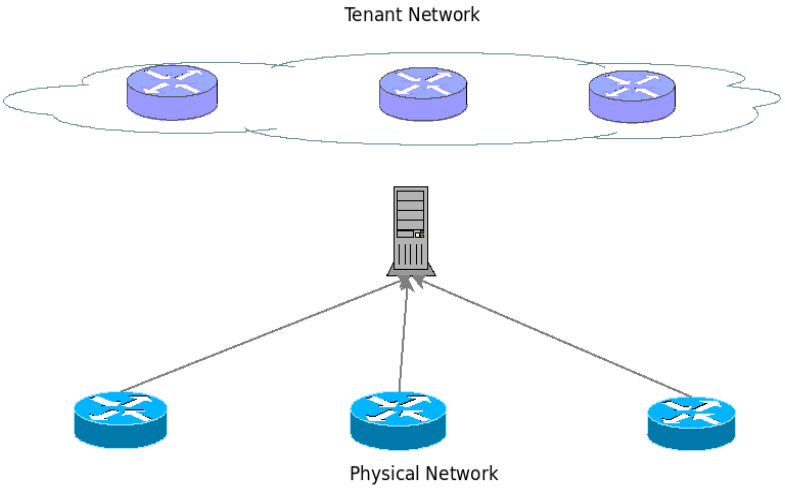

Figure 4.3: Basic 1:1 Set-Up

1:1

The principal mapping 1:1 is shown in Figure 4.3. As the figure depicts the number of physical devices is the same as the number of the virtual ones. According to the SDN concepts, the connections are performed in the physical layer. This is the reason why the traffic between the virtual resources goes not only through the virtual devices, but also through the physical ones. Having this kind of mapping does make sense, since the number of ports that are used for virtualization is obtained from the number of virtual devices raised to the second power. Variable count of the tenants will be used for proving the concept of this mapping.

1:N

Figure 4.4 shows the concept of 1:N mapping. One physical device is opposed to many virtual devices. In this case, even more interfaces of the physical infrastructure are left free. Nevertheless, this network infrastructure is useful for different conditions of the underlying physical device. When designing this mapping, we consider that the tenants' network could benefit from the fact, that no forwarding takes place on the Layer 2 outside the device.

N:1

Figure 4.5 clarifies the N:1 proportion of network resources. Many physical entities are abstracted into one virtual entity. Similar to the other mappings, this design has the great advantage of reliability of the physical infrastructure.

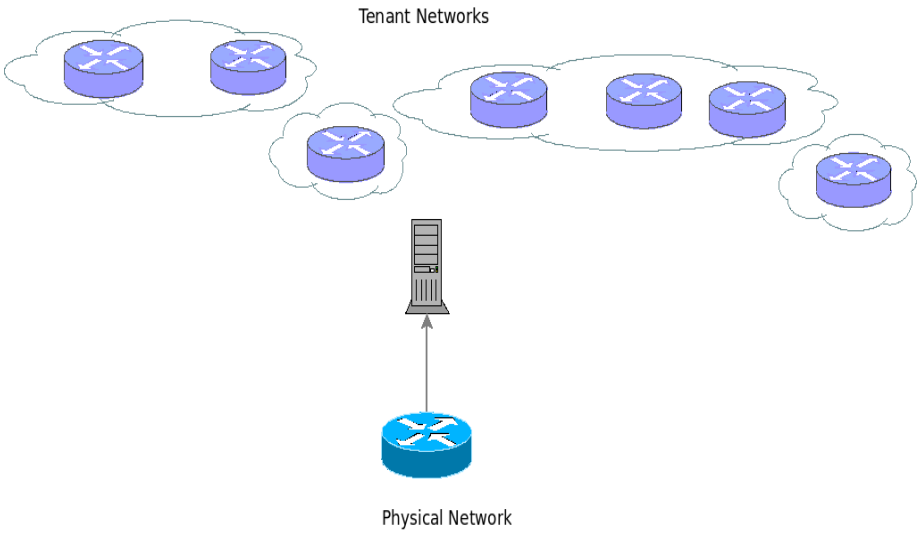

Figure 4.4: 1:N Mapping

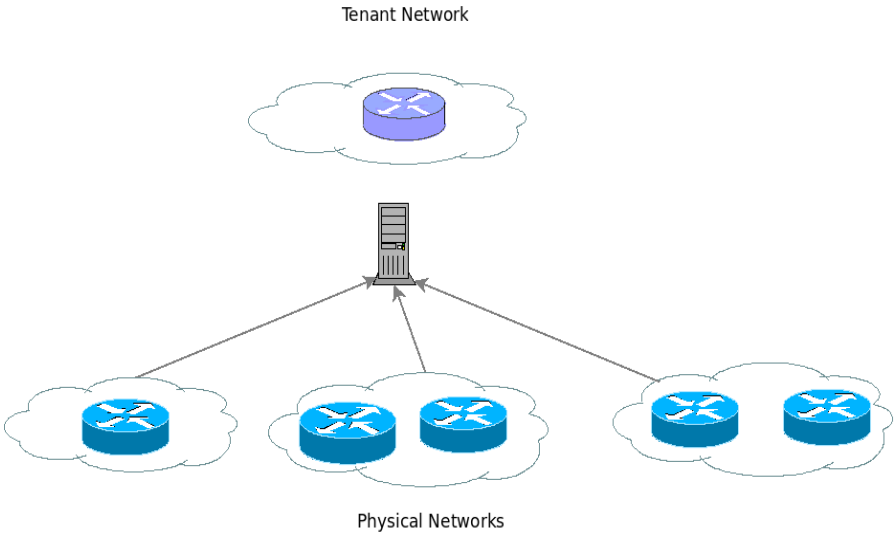

Figure 4.5: N:1 Mapping

N:M

Figure 4.6 shows a N:M network prototype. M virtual devices are built on N physical devices. This mapping can overlap with some of the other cases, depending on the particular setup and configuration. This is the most promising, as long as reasonable trade-off between reliability and datacenter costs is found.

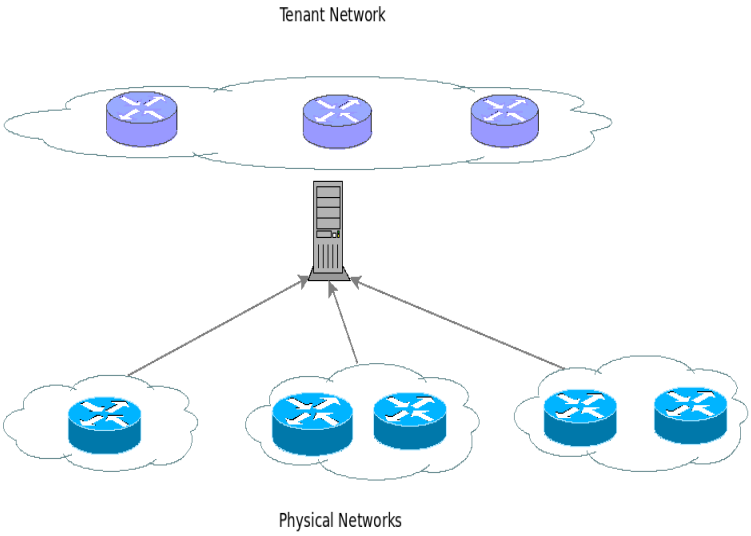

Figure 4.6: N:M Mapping

4.4.1 Simulation Design

The simulation of the network topologies is done using MiniNet [58] [73]. MiniNet is a network simulator used to create scalable SDNs using Linux processes and thus it creates a realistic virtual network by running a real kernel switch. Mininet is used in this work mainly to simulate the topology of measuring traffic flows. The topology is defined using the Python language, which is then created in MiniNet. After that, traffic flows are received from a remote OpenFlow controller.

Due to the specificities of the used tool, the physical switches are tagged as physical and then simulated with Open vSwitch. Usually, MiniNet is used to prototype only the virtual network, that is why in this work we will use the workaround to label the switches differently, namely: physical switch and not labelled switch. With the simulated topologies we cover all of the aforementioned mappings in cloud network scale. The scale of the proposed topologies will be related to the scale of public cloud datacenters. This will conclude our hypothesis for the most optimal mapping, which we aim to confirm with real test-bed.

4.4.2 Test Bed Design

For the purposes of experimentation, a sample network is set up consisting of three to five OpenFlow switches and Floodlight. The switches, in turn, will be connected to hosts with traffic generators on them. In this way, respective traffic is created on each node where a number of VMs is supposed to be placed. This emulates real production network. Further perfections with respect to this are not considered to be essential for evaluating the success of neither the monitoring plug-in nor of the mappings. As research will be limited to exploring the benefits of monitoring the physical infrastructure from inside the OpenFlow controller in a lab-emulation

of a production network. The results from this study can then be extended to real production networks.

The test-bed also comprises of all of the designed mappings. Its scale differs greatly from the simulation's scale. Nevertheless, we expect that the identical topologies will show similar results and, thus, will confirm the validity of the results obtained from monitoring the simulations of the topologies.

CHAPTER 5

Implementation

The following section briefly discusses the implementation specifics of all components that are subject of analysis and development in this thesis. First, to set the basis, the implementation characteristics of the controller are presented, which are relevant to the implementation of the plug-in, followed by the description of the plug-in implementation. Finally, we discuss the technologies used and the implementation of the simulations and the tested.

5.1 Floodlight Controller Implementation Interfaces

For better understanding the reasons for the later discussed plug-in implementation decisions, we present the main points in the Floodlight architecture, that are important for this thesis.

5.1.1 Technology

Floodlight is Apache-licensed, Java-based controller. Together with Beacon [43], are the only controllers written in Java. Floodlight is actually forked from Beacon, originally developed in Stanford. The core of a commercial controller product from Big Switch Networks is built on top of Floodlight.

It is based on Java 6, and Eclipse version 3.8 is recommended for developing. It uses mainly REST technology for exposing its interfaces. The official repository of the open source project is hosted on GitHub [82]. Another reason for choosing Floodlight for basis of our research is that it is being extensively tested. Unit testing is made using to EasyMock [39], and, additionally, integration tests are provided from the community.

5.1.2 Implementation

The main goal of the controller is to replace the controller plane for the underlying switches. That is why it needs to communicate very intensively with the devices on one hand, and with northbound APIs on the other hand. The decision that the communication on the devices side is

implemented with messaging pattern puts the controller in most cases in the receiver role. As explained before, the implementation of this part of Floodlight controller's interfaces follows mainly the `Observer` pattern. The services that are responsible for initial message processing are named "OpenFlow Services" in Figure 5.1 [21]. On the right-hand side of the OpenFlow services is the *Module Manager*, that initiates all services in the controller and also dispatches the OpenFlow messages. The Module Manager is also a container of all modules instances. It also controls all storage capabilities of the controller. The Module Manager and the OpenFlow services build the *Floodlight Core Services*.

Above the core services in Figure 5.1 are the *Internal services*. They are generally independent of the Core services, but they are bounded to the Module Manager. Although they do not directly reference the core modules, they fill in the basis of knowledge, built by the core services, in order to provide complete information for the *Floodlight Module Applications*. The internal services are responsible for the correct processing of each new OpenFlow Message, while the Applications are those who take the decisions, specific for the control plane. The Module Applications have the same structure as the other services, so the Module Manager can manage them, but they do not intercept OpenFlow messages.

From the figure it is clear that most of the modules expose a REST interface. Only the modules classified as OpenFlow services and Internal services are also listeners for switch messages. The module "Hub" has neither REST interface nor is listener in the current available Floodlight version. By design, all modules are independent of others, and thus are easy to change without effecting others' functions. Adding new module is also made easy by the provided interfaces. Keeping this in mind, we build modules that follow the same structure as the others in order to be recognisable and manageable by the module manager.

5.2 Plug-In Implementation

In the following section, the implementation of all parts of the proposed plug-in are described and how they are incorporated in the current Floodlight implementation. The section is concluded by examining briefly the test's characteristics.

5.2.1 SNMP Module

After analysing the available images of OpenFlow enabled switches, we consider that using SNMP is possible. All of those images have enabled NetSNMP and are on Linux, which means that the *Object Identifier (OID)* [1] for the different counters would be the same. In SNMP different OIDs are used for retrieving different metrics.

In case we are using mainly implementations based on Linux, more of the OIDs for the specified metrics are the same and are provided in the module. In other cases, a configuration file for other, specific ones, is available.

The SNMP Module itself follows the implementation model of every other module in Floodlight. It does not intercept OpenFlow messages and has no REST interface. As the functions it accomplishes are related to maintaining connection to each switch, semantically it is placed in the Core modules of Floodlight. Therefore, it can be managed by the Module Manager, such that

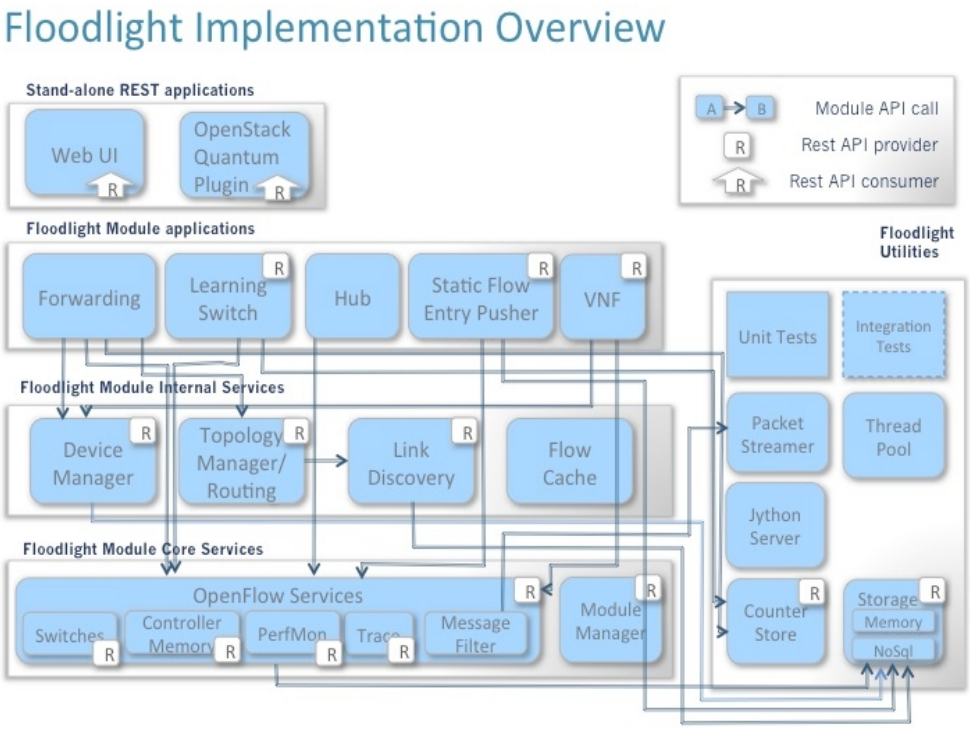

Figure 5.1: Basic Floodlight Architecture (taken from [21])

a *proxy* of the SNMP module is created at run time to create the illusion that the other modules can access it directly and simultaneously keep different calls in isolation. In actuality, the Module Manager will invoke it with the correct connection properties for the switch specified. The collected values are stored in the *Counter Store* Module, so they are available for Application Modules.

5.2.2 Monitoring Module

The *Performance Monitor* is implemented according to the already discussed standard in implementation of Floodlight Modules. It implements the `Observer` pattern, so that it can receive the OpenFlow messages as well. The module also exposes not one but two interfaces - REST and file-based. In Figure 5.2 it is shown where semantically the new modules belong. As the Performance Monitor has an encapsulated functionality as the Module Applications, but also is a listener for the OpenFlow messages as the Internal Modules, it should be placed between both. Nevertheless, it is placed above between the *Quantum plug-in* and the Module Applications, because its function is mainly oriented to cloud usage of the controller.

First, specific in the implementation of the module is that unlikely the other modules it always gives the Module Manager the same instance. This is particularly useful on message

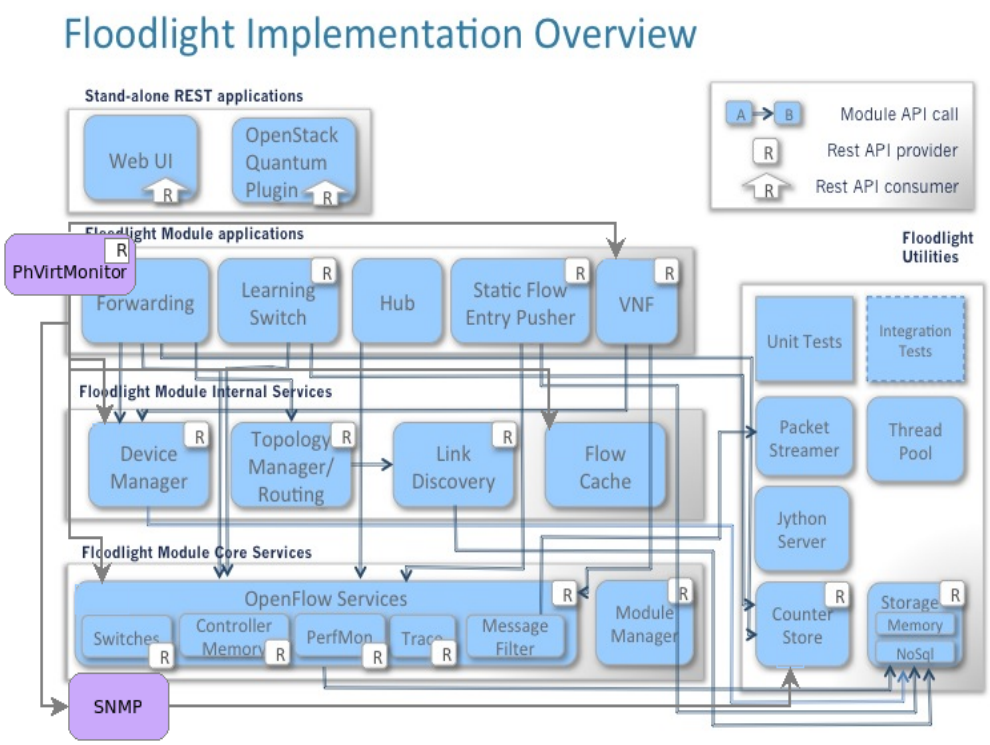

Figure 5.2: Plug-in and Floodlight Interoperability

processing. When a new message is received, it is processed sequentially from all modules that implement the `Observer` pattern. One of the metrics gathered in the Monitor is the Flow Processing Time. Therefore, a naive mechanism for a stopwatch is implemented by manipulating the Module Manager to notify the Performance Monitor on receiving and on finishing with processing the OpenFlow message. This requires the same instance for the specified message.

Another particularly interesting implementation decision is the management of switch instances. After deep analysis of the Floodlight code, it is found that the same switch, in different modules has different identifiers. They are generated and associated by the particular module on switch discovery. Therefore, the Monitor module implements also an `Observer` pattern for the *new switch event*. Each time a new switch is discovered, deleted, connected or disconnected to the controller, the Monitor module is notified with the respective identifiers of the module. This way, a store with the common characteristics, such as IP, MAC address and the specific identifiers is built. Also, each switch is stored with the ports that are connected and used in the moment.

For the purpose of calculating different metrics, which cannot be extracted directly, utility classes are used.

The Performance Monitor Module, similarly to the controller Manager, implements a timer task that invokes the Monitor itself after specified time period. In order to constantly monitor all

metrics, the Monitor needs to gather them periodically. The specified period can be defined in a configuration file, if any slowdown in the entire controller's function is noticed. Such periodical invocation is not typical behaviour for the other controller modules, so further actions are taken that none of this invocation coincide with an invocation from the Module Manager.

5.2.3 Tests

In Floodlight, there is an established convention for unit and integration testing. As mentioned in the previous section, integration testing is done by scripts developed by the developers community, which verify that the newly implemented module does not effect current behaviour of the controller in regard of speed and robustness. Therefore, no contributions are made in the integration testing, similarly to other modules, which don't change the high-level architecture of the controller, this test scripts are used for verification.

The unit tests on the other hand, are extended to cover both new modules. In the tests, the EasyMock framework is used on top of new virtual machine with an Open vSwitch software on it. This way, the network with one switch instance and one controller instance is emulated. This is the established testing strategy used in each Floodlight Module, and we follow this convention for testing the monitoring module. It has well-defined and encapsulated functionality and such an end-to-end unit test is reasonable and possible.

As far as it concerns the SNMP Module, another testing methodology is needed. The tests are considering the machine on it they are running, in this case the developer's machine, where an SNMP daemon is installed. Here also an implementation of another Module Manager test is needed to verify the `Proxy` pattern implementation. An emulation of SNMP Agent is provided in the test, and then invoked outside the Module Manger, the way it also happens run time.

5.3 Minimet Simulation

The MiniNet simulations that are implemented in this thesis aim to cover all the mappings described in Section 4.4. Therefore, for the different mapping types implementations, the following number of physical switches: 1, 2, 3, 10, 50 are chosen. As discussed, they are mapped according to four different types of mappings. In case of one physical switch only 1:1 and 1:N are possible. Further the N:M case coincide for many of the cases, therefore the simplest cases for two, three and for ten switches are provided. This means for N:M mapping for two switches only one mapping, more precisely 2:3, whereas in case of three physical switches two different mappings are built: one with more than three and one with less then three virtual switches. Analogously, for ten switches the topologies are also two. All together, nineteen different topologies are built to cover the conceptual four types of mappings.

5.3.1 Configuration

The simulated topologies in MiniNet are implemented via Python scripts. An example of the simplest possible script is presented in Listing 5.1. First, an unique identifier is given to each entity, and then each entity is added as a node, specifying its type. Finally, the links between the resources are defined, and the topology is marked as enabled. In this case the topology

consists of one physical (**pSwitch**), one virtual (**ovSwitch**) and a host. The other topologies for the MiniNet simulation are implemented in similar way.

```
host = 1
ovSwitch = 2
pSwitch = 3

# Add nodes
self.add_node( ovSwitch, Node( is_switch=True ) )
self.add_node( pSwitch, Node( is_switch=True ) )
self.add_node( host, Node( is_switch=False ) )

# Add edges
self.add_edge( host, ovSwitch )
self.add_edge( ovSwitch, pLeftSwitch )
self.add_edge( pLeftSwitch, pSwitch )

# Consider all switches and hosts 'on'
self.enable_all()
```

Listing 5.1: Simple Topology Configuration

5.3.2 Topologies

After running the MiniNet simulator and configuring a particular topology, it is ran by the emulator using a command line tool. The network resources are created and connected with the Floodlight controller on port **6633**. The command in MiniNet is shown in Listing 5.3. The output from this operation can be seen on Figure 5.3.

The MiniNet simulation is based on an actual Linux instance, so traffic generation can be carried out by any Linux traffic generation tool. For the purposes of this thesis, we use *iperf* [14]. Iperf is commonly used for generating UDP and TCP traffic in order to measure the throughput. In this case it is used to exhaust the links between the simulated hosts aiming at more accurate estimation of the mapped topology. The generation in the simulator is triggered as shown on Listing 5.2.

```
minimet>source_host_name iperf −c targer_host_name
```

Listing 5.2: Iperf Simulation

```
mn −−custom fiftySwManyH.py −−topo fiftySwManyH −−mac
−−controller=remote −−ip=10.0.2.2 −−port=6633
```

Listing 5.3: MiniNet Command for Configuration Loading and Controller Connecting

After creating each topology, it is monitored at least six hours under constant traffic generation. This provides at least twenty five completed cycles of measurement and respectfully gives

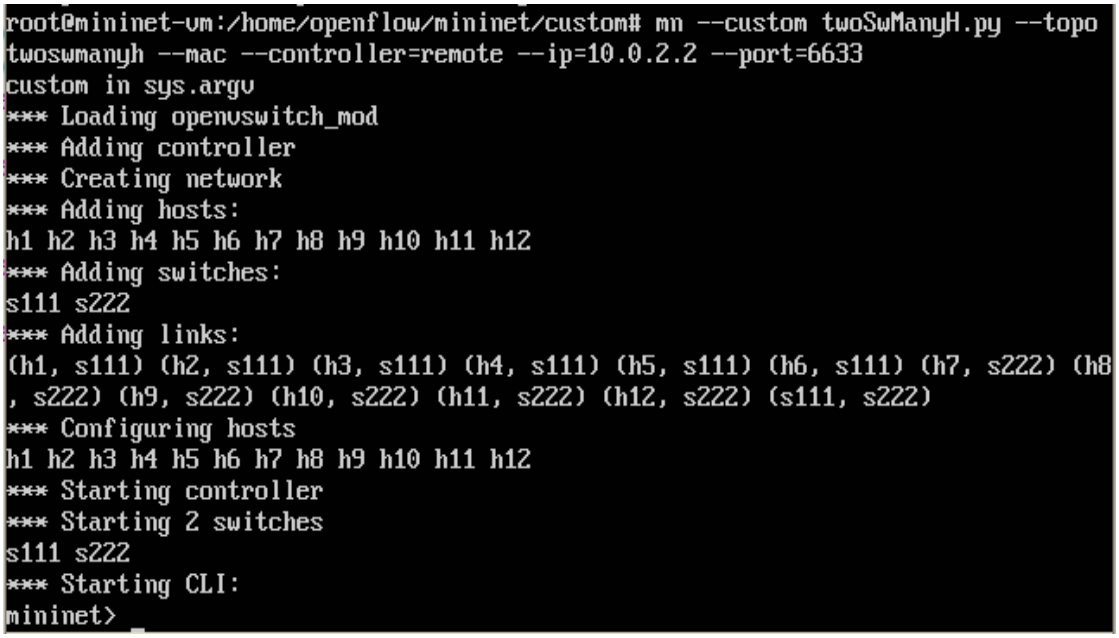

Figure 5.3: Command Output

enough data to make comparisons between the different mappings. Further measurement time will be provided if and when the results for different mappings are too similar or even equal, and no estimation of the correspondence between the physical-to-virtual resources is possible.

In order to keep the measured environment equal for all the proposed mapping, the tenants on the virtual cloud network are always created one after another until they are the same number as the underlying switches. After their continuous creation, they are destroyed the same way until zero tenants use the infrastructure. Depending on where and when a tenant network is created, on the respective hosts is generated traffic. This provides a close emulation of a production cloud network.

5.4 Test Bed Implementation

Here, the implementation of the test bed is discussed. The purpose of building a test bed is to verify the results and the conclusions drawn from the MiniNet simulated mappings. Firstly we introduce the used hardware.

5.4.1 Technology Overview

As the testbed architecture is complex and includes different hardware components, we provide a brief overview of the used technologies and how they cooperate.

5.4.1.1 Hardware

The hardware platform of the physical switches is a *TL-WR1043ND Wireless N Gigabit Router* [87]. The device has 4 Gigabit ports. The switch creates wireless network with a speed of up to 300 Mbps on each interface. The device also provides a USB interface. It has 32 MiB RAM and 400 MHz CPU Speed and it also includes Gigabit switch Realtek RTL8366RB. This are the features that concern our needs in this work, the other characteristics of the device are available in the specification [86].

5.4.1.2 OpenWrt

OpenWrt [31] [15] is an operating system based on Linux, that is mainly used in network devices. OpenWrt provides the user with the possibility to write files and install software and a set of scripts called UCI (unified configuration interface) intended to unify and simplify the configuration of the entire system. It also has build-in network switch and VLAN capability. OpenWrt enables the user to configure the entire network using features such as load-balancing, IP tunnelling and others. It is actively developed and also recently is equipped with, extensive Web interface.

For the purposes of this thesis, we need an OpenFlow enabled switch. Currently, building such a device is accomplished via *Pantou* [29]. After analysing many of the implementations of OpenFlow networks, we consider using Pantou as it is most stable. Pantou is a patch on OpenWrt that enables an OpenFlow application to run over the operating system. Reconfiguring the above specified device has its difficulties, as long as its version is newer than the latest release of Pantou. Further details of the process of re-flashing the devices are considered not particularly interesting for the thesis topic and will not be discussed further.

5.4.1.3 Open vSwitch

The Open vSwitch setup in the testbed is version 1.4.5 and is ran on Linux machine via KVM [47]. The host machine is with the 4G RAM and 2.5 Ghz Core duo processor.

5.4.2 Topologies

The topologies created within the testbed aim to provide confirmation of the simulations results, therefore, part of the MiniNet simulations designs are used. As described in the design of the test bed, the physical switches that we have available are two. That is why the mappings created are the corresponding: 1:1 and 2:2 , 1:N (in this case 1:2), N:1 (2:1), N:M (2:3). The topologies differ in minor details, therefore, here is described the second one, where we have one physical and two virtual switches (cf Figure 5.4).

The figure shows the physical switch connected to a server farm, where two virtual switches are hosted. There is located the machine with the Floodlight controller. Physical connection between the physical switch, Floodlight, and both virtual switches exists. With the provided capabilities of Floodlight, two tenant networks are built. In each of this networks a number of VMs are placed (in this case generated traffic will emulate their existence, as we are not interested in the machine itself). The figure depicts the scenario, where one VM communicates with a VM in

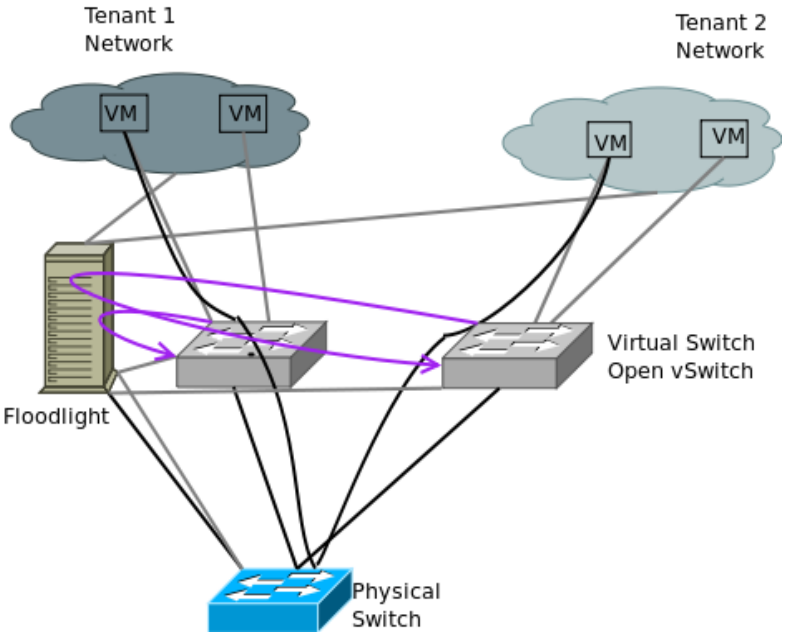

Figure 5.4: Test Bed for the 1:N Mapping

another tenant's network. Firstly, the two virtual switches talk with the Floodlight controller to set-up the flows between the machines (the purple line). In this case, the traffic from the source machine goes though the virtual switch that is logically connected to the physical switch, then from the physical switch to the other virtual switch, and finally to the destination VM. Here, we have as physical resources first the physical switch itself, second the machines which host the virtual switches. In Floodlight, which has connections with these both, we measure the network relevant performance either of the switch or of the host. At the same time traffic metrics for the virtual switches are also gathered. This lead us to the same result set as we have by the simulations, so comparing both results is possible.

CHAPTER 6

Evaluation

In this section we perform an evaluation of the work carried out in the thesis, as well as provide some conclusion about the benefits and problems of the presented mappings. The evaluation focuses on the design decisions discussed in Section 4 as well as the concrete implementation presented in Section 5, of both the plug-in and the mappings. The proposed mappings can hardly be compared to any existing similar experiment. However, our evaluation is based on the efficiency and the performance of the system, as viewed as a single unit. The main parameters used are the latency in the controller function and accuracy of the data gathered. In case of mapping evaluation, the criteria are throughput and actual physical behaviour.

6.1 Plug-In Evaluation

The evaluation of the implemented plug-in is performed while implementing and gathering the metrics proposed in Section 4.

A parallel measurement of the bytes and packets per second with SNMP directly on the physical switches is performed after running the integration test scripts on a system including the Monitoring plug-in. Figure 6.1 compares the results gathered on the the same switch from the plug-in and from SNMP. The figure shows that the deviation of the plug-in results is less than 1% compared to the SNMP results. Additionally, while gathering the mapping data, the plug-in has been invoked 4258 times per switch and only 11 failures of measurement occurred (mostly due to the network connection between the controller and the switches). The error percentage in this case, by switch metrics is 0.26%. Therefore, we consider that the implemented prototype of the designed plug-in performs well and is reliable. There is no difference in the error percentage for either types of different set-ups, namely MiniNet and the physical setup. Nevertheless, there is delay in the MiniNet responses, which occurs by topologies with a large number of switches. As MiniNet's topology is running on one physical machine, in case of large number of switches, all of them and the corresponding benchmarks are being executed on a corresponding VM, which may the cause the aforementioned delay.

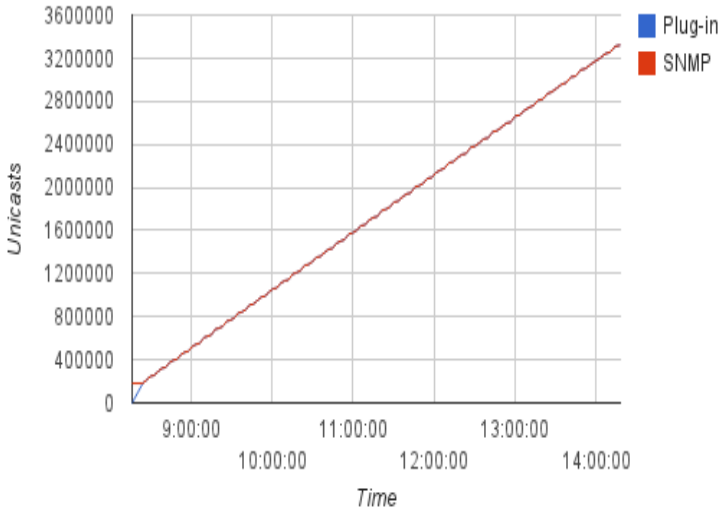

Figure 6.1: Comparison of the Measured by the Plug-In and the SNMP for the Unicasts

6.2 Simulation Results

In this section, we discuss the results of the mappings implemented in MiniNet. The specific topologies that implement the mappings are evaluated according to their physical resource usage and the corresponding throughput. The OpenFlow criteria used for this evaluation are the flow arrival rate and the metrics that describe the switch memory (active flows and flow entry usage). For MiniNet topologies, the metrics of the flow setup time are not present because of the limitations of the underlying implementation of the OpenFlow protocol in the Open vSwitch.

6.2.1 One-to-One Mapping

The topologies that implement this kind of mapping are the 1:1, 2:2, 3:3, 10:10, 30:30 physical to virtual resource mapping, respectively. The resources used by the controller are all the same for the five topologies under consideration. It is also interesting to observe that there is a negligible difference in the usage of the CPU in the first two scenarios (with one or two switches). Considerably more CPU usage occurs when the physical switches are 10 or more. The most interesting, in this topology type, is the case with 3 switches, where the resources used seem to be far less. In this particular case, this behaviour is most probably due to simulation resource usage specifics. Another possible explanation, could be that the simulated traffic is bridged not from the MiniNet, but from another process and, therefore, is not accounted by the plug-in. Figure 6.2 and Figure 6.3 are graphical representation of the comparison, whereas the Figure 6.3 shows in details, the tendencies development in the end of the measured time period.

The other notable criterion is the throughput of the switch. Figure 6.4 demonstrates the

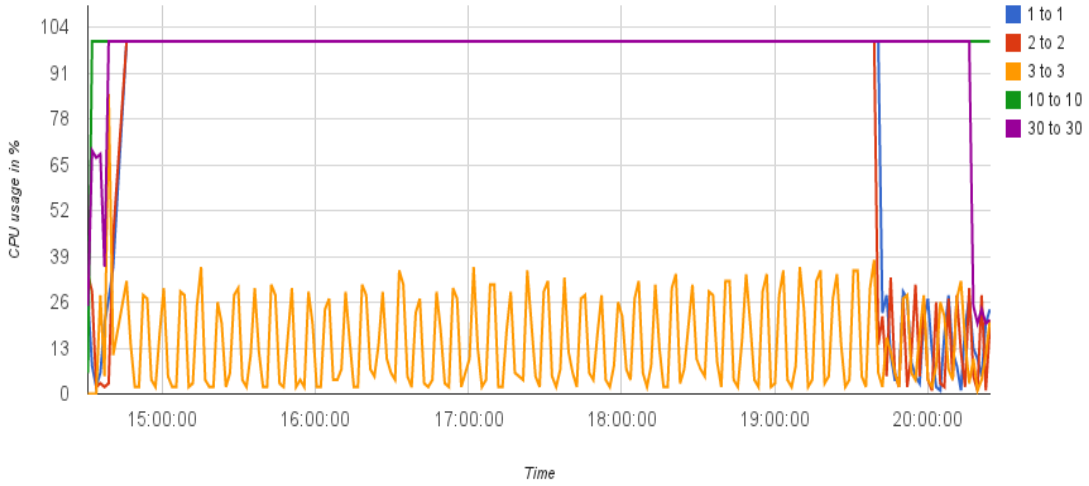

Figure 6.2: CPU Consumption Comparison Between the Implemented 1:1 Topologies

average load on the interfaces of the switch. This metric shows the percentage of the throughput capacity that is taken at the moment. From the figure it can be observed that the topology with 10 switches has optimal behaviour - it has no drastic peaks, and is relatively smooth which means that eventually the topology's load would be easy to predict. It is also important to notice that there are no slow-downs or speed-ups observed in the controller's flow arrival rate. We consider this to prevent possible failures and problems in the controller function as well in the OpenFlow tables in the switches.

6.2.2 One-to-Many Mapping

This kind of mapping includes the topologies with the following physical and virtual switches: 1:2; 1:3; 1:10; 1:30. It can be noticed from the results that the tendencies in the resource usage and loads are similar in all cases. During forwarding, the CPU and the RAM consumption are on the upper limit, and after that they sharply fall to between 5% and 10%. On the controller side, we did not observe any slow-down, but flow arrival rate decreases as time passes. This is an abnormal behaviour, because as the time passes, the OpenFlow tables in the switches should have information for the larger part of the flows. In the other described cases of mappings, such as 1:1 and N:M, no such anomaly is observed. After comparing the Flow Entry Usage in each physical switch (cf. Figure 6.5), we consider that this behaviour may be due to OpenFlow table saturation with active flows. Interestingly, the virtual switches' tables are empty. Therefore, the physical switch in this type of mapping gets overloaded by caching the virtual switch's flows as well. However, another cause for this behaviour could be the fact that the OpenFlow tables

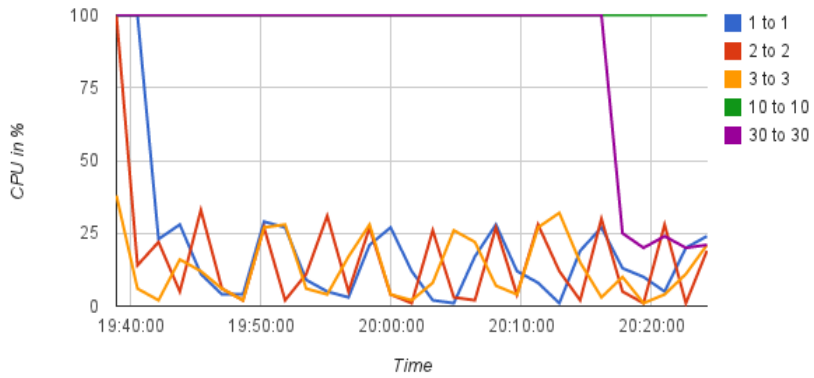

Figure 6.3: CPU Consumption Comparison Between the Implemented 1:1 Topologies (after 19:00)

are cleared too often and the switch cannot gain knowledge about the network. The fact that the physical switches maintain large tables, lead us to the conclusion that the cause lies in the implementation of the virtual switch.

Nevertheless, if similar behaviour is observed in production network and the virtual OpenFlow tables function correctly, this may be due to fast traffic changes. Depending on the traffic engineering policies such a situation may be desirable or treated as a security threat.

According to these results we consider that using such a correlation between physical and virtual resources would be suitable for networks with a smaller number of virtual switches. As shown by our experiments, the number is less than 10 Open vSwitches. However, the results are likely to be different if we consider other implementations of the virtual switch.

On the one hand, the mapping is intensive for the physical switch, but on the other hand, it performs well for the controller hardware, which enables a fast control function on the topologies. Nevertheless, the CPU and the RAM usage are critical for the implementation of this mapping.

6.2.3 Many-to-One Mapping

For the purpose of evaluating this type of mapping, we have implemented the following topologies: 2:1; 3:1; 10:1; 30:1. Similarly to the other topologies' results, the controller is performing well under the executed benchmarking, and the underlying hardware does not show deviation from the expected behaviour. Unlike the aforementioned mapping type, there is no abnormality in the flow arrival rates: as time increases the time interval for new flows becomes greater, and a new flow rarely comes. Therefore, we consider that the behaviour of the OpenFlow tables in the switches to be normal, and reaching a point of saturation is less likely. The RAM usage in the switches, therefore, is not expected to be critical. However, the CPU consumption of each switch does not show a stable behaviour. There are sharp peaks and drops of the CPU percent-

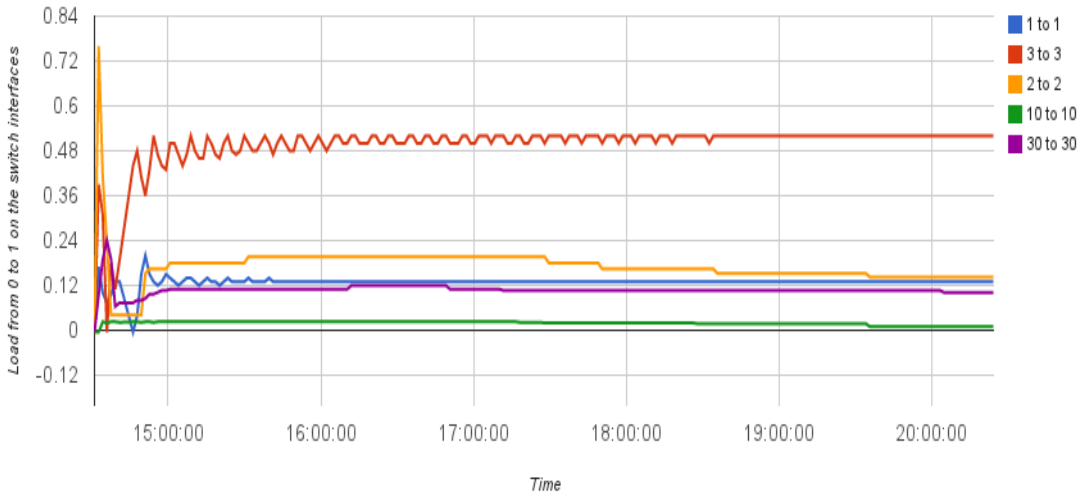

Figure 6.4: Load Comparison Between the Implemented 1:1 Topologies

age. Both, the peaks and the drops are to be observed at the same time in the physical as well as in the virtual switches. In this case, contrary to the other two mappings described earlier, there is a difference between the throughput of physical and the corresponding virtual switches. This difference is shown in Figure 6.6 for 30 virtual switches and on Figure 6.7 for 2. From the figure comparison, we see that the behaviour of the larger topology is more appropriate for real-life implementation. In case of 2 switches there is a slight difference in the loads' trends of the physical and the virtual switches, which is not desirable, because no prediction is possible. We should further notice that in specific time intervals, the physical load is less than the virtual. In the case of 30 physical switches, there are no sharp peaks and changes in the trend, and in addition the physical switch load is with 16% more than the virtual one. In this topology, this is a suitable behaviour, because the physical switches need to compensate the load of the virtual, and the virtual one is not likely to get overloaded. What is also important is that the behaviour of the virtual switch is predictable - there are no sharp changes in the load trend.

6.2.4 Many-to-many Mapping

In this case two sub-cases are discussed:

M Less Than N This includes the implementation of 2:1; 3:2; 10:5 topologies. The observed results show that the gained throughput is optimal for the number of physical switches. Although some physical switches appear to be unused (especially in case of 10 switches), the topology continues to perform well.

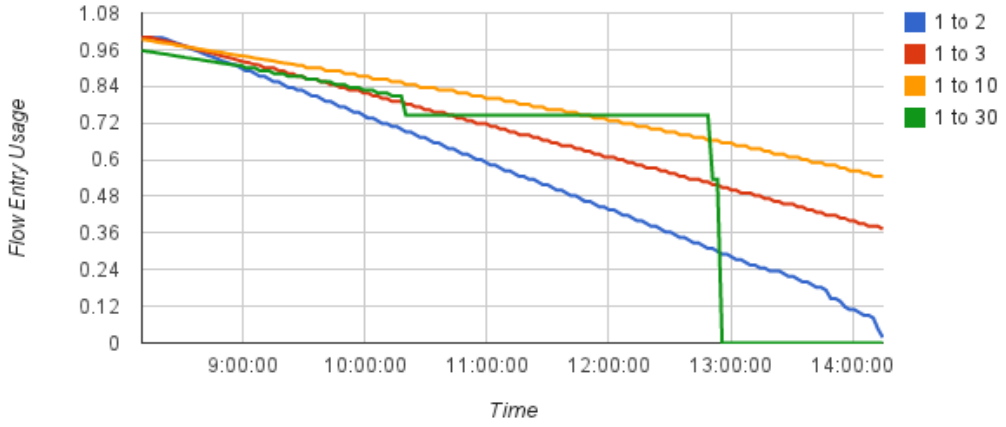

Figure 6.5: Flow Entry Usage of the OpenFlow Table in Average Switch in 1:N Mapping Type

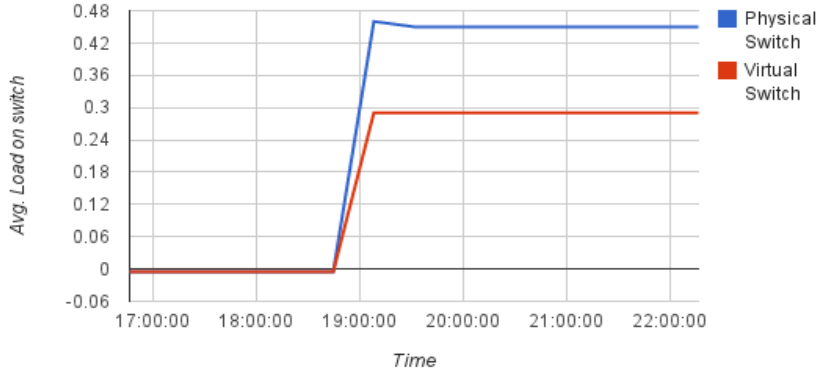

Figure 6.6: Load Comparison Between Physical and Virtual Switches in Case of 30:1 Topologies

M Greater Than N The topologies that build this experiment are: 2:3; 3:4 and 10:15. The virtual switch throughput in this case is worse compared to the previous one. There are physical as well as virtual switches that show overhead. They are under-provisioned and only burden the topology.

Due to that the controller is not performing as good as in the previous three mappings types. In conclusion, we consider the case where the number of virtual resources is less than the number of physical ones to be more suitable for usage in cloud environments. As Figure 6.8 shows, the load on the switches is more stable compared to the load in the case when the virtual resources

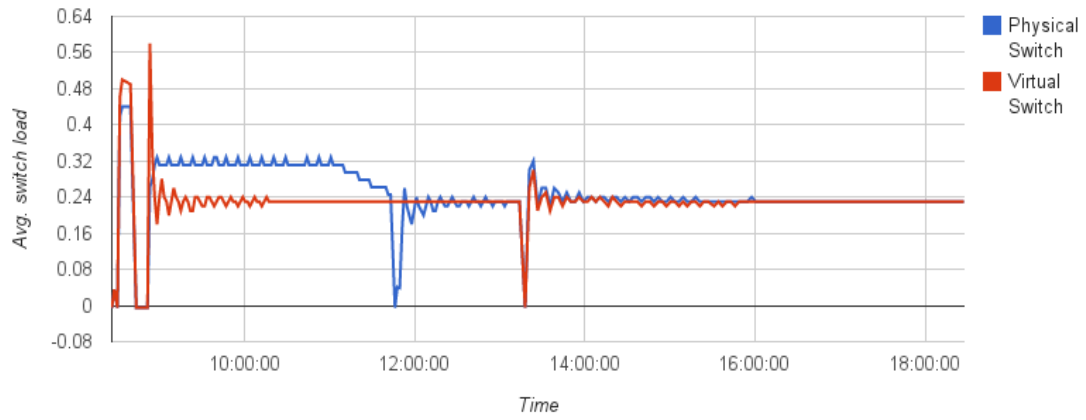

Figure 6.7: Load Comparison Between Physical and Virtual Switches in Case of 2:1 Topologies

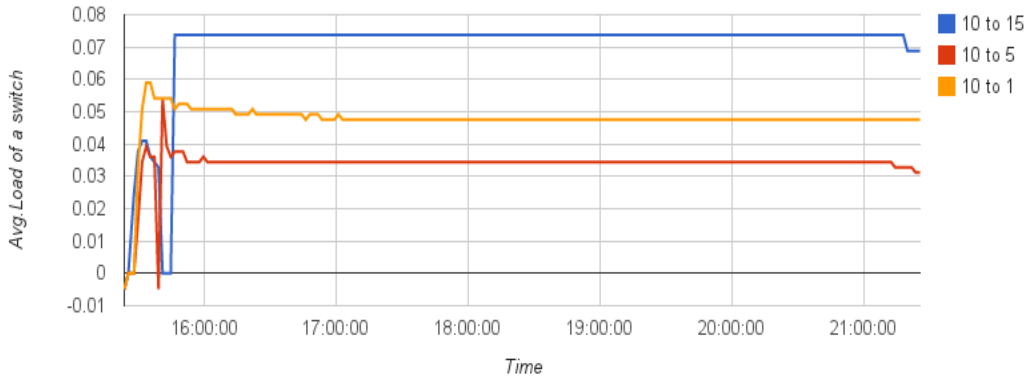

Figure 6.8: Comparison of Average Load of Both Physical and Virtual Switches in N:M and N:1 Mappings

are more than the physical. Additionally, there are less physical switches with very little or no load, which means that the correlation between the physical and the virtual resources is better.

6.2.5 Conclusion

After closely examining the benefits and the weaknesses of all types of mappings, we consider that depending on the number of the tenants and the degree of easy provisioning that the network need to provide, different physical to virtual resource correspondences are optimal. For small networks, a good design decision would be to use 1:N, as the trade-off between the load on the physical switch and the overhead of the increased CPU usage is reasonable. In the case of larger

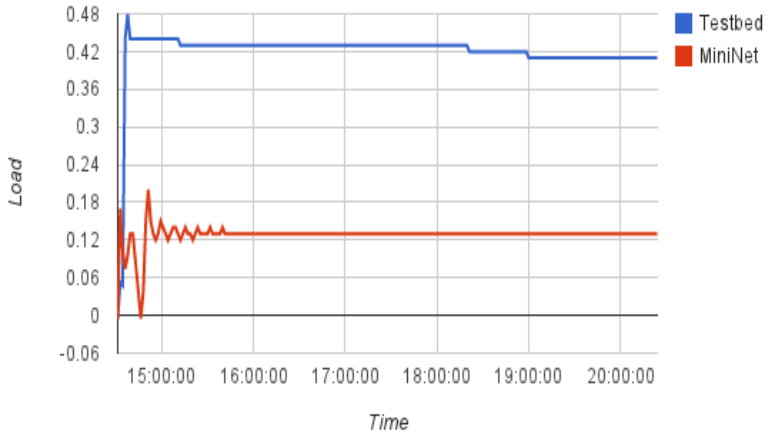

Figure 6.9: Comparison Between 1:1 Mapping - Test Bed and Mininet Simulation

networks with more tenants, and for networks with a rapid increase of the tenants, the optimal mapping would be N:M, where M<N. It is expected to result in a better and more efficient provisioning, as well as in faster forwarding, due to the minimization of the physical resource overhead.

6.3 Test Bed Results

6.3.1 One-to-One Mapping

The testbed that follows one-to-one mapping confirms the results from the simulation: the controller's resources are stable and the load behaves similarly to the simulated physical and virtual resources. In Figure 6.9, the main differences in the loads are shown. Respectively to the OpenFlow tables, the flow arrival rate is considered normal. Comparable to the other test bed topologies there are no sharp peaks or drops in the load. This is a tendency not only for this mapping, but also for all four types. We consider that this may be due to MiniNet's implementation rather than due to the topology changes.

6.3.2 One-to-many Mapping

The one-to-many mapping in the test bed is represented by the 1:2 topology. As by MiniNet's, we did not observe anything special in the resource usage: during intensive forwarding, CPU and RAM are nearly 100% utilized, whereas by decreasing traffic, this percentage sharply drops. The tendency of the average load of the virtual switches is the same. Notable is that the average values are with 11.2% greater than these of MiniNet. As the results in the simulation show, this particular topology behaves not optimally for large traffic loads and large number of flows

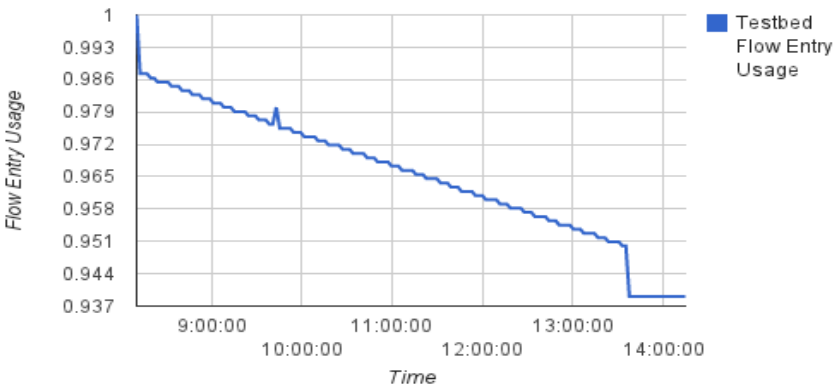

Figure 6.10: Flow Entry Usage for Mapping Type 1:N (Test bed Measurements)

according to the flow entry usages. Therefore, we look closely at this metric in this case in Figure 6.10. As the time passes it is expected that the OpenFlow tables in the switches contain more and more forwarding destination addresses and, therefore, the switch itself will have more information about the network. According to this consideration, as the time passes the *Flow Arrival Rate* in the controller is expected to decrease and the *Flow Entry Usage* of the switch table to increase. In the test bed as well as in the simulation, there is a decreasing trend in the *Flow Entry Usage*. There is a difference in the decreasing rate for both. In the simulation, the metric values decrease much faster than in the test bed case. This could be due to the fact, that the simulated Open vSwitches have different OpenFlow table capacities than the test bed ones. Another possible explanation for this behaviour is that the caching of the OpenFlow tables is inadequate, in this case the cache interval could be very small. In a production network such behaviour can be due to fast changeable flows. In the provided experiments it is not the case, but if such conditions are observed and none of the above described OpenFlow table issues are present it is likely that the network is subject of an attack.

6.3.3 Many-to-One Mapping

The testbed for validating the results of this kind of mapping simulation consists of two physical and one virtual switch. The only major difference between the simulation and test bed is that, in the testbed results, there are no sharp peaks or drops in any of the metrics. In the simulation, we observed unstable behaviour of the CPU usage, but in the test bed it is always 100%. More important in this case is that the tendencies in the switch loads are the same. Here, like in the simulation, in case of two physical switches tendencies of the physical and the virtual switch loads are the same. As the load of physical switch increases, so does the load on the virtual. However, a difference of 12% between the testbed's and simulation's results, in case of physical switches is observed, where in case of the virtual switches this value is 19%. As Figure 6.11 shows, the difference between the resources types is present and it is also notable that when a

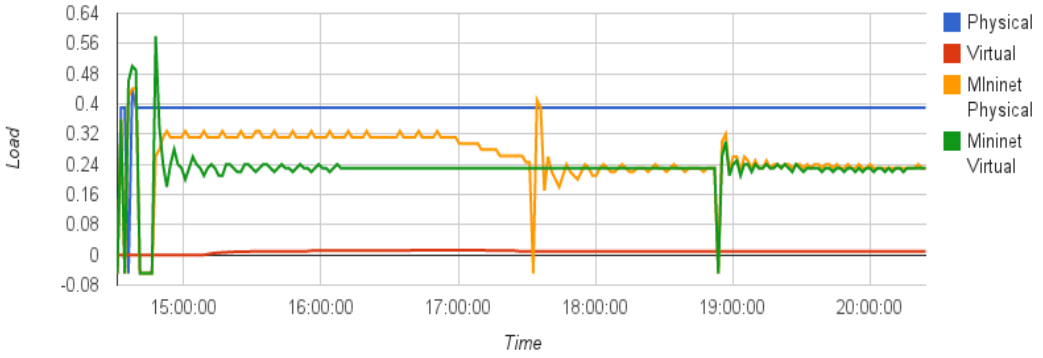

Figure 6.11: Physical and Virtual Resources in the Different Experiment Setting by Topology N:1

change in the physical switch load is present it is then also observed in the virtual.

6.3.4 Many-to-Many Mapping

Evaluating this kind of topology model includes configuring and running a network with two physical and three virtual switches. Surprisingly, here we are able to observe that the physical switches maintain constantly more load on their interfaces compared to the MiniNet's ones. The comparison of the average loads on the switches in the two different cases is shown on Figure 6.12.

Overall, our conclusion that there are some virtual switches that burden the bandwidth and cause overhead has been confirmed. For the sake of reliability, this topology was connected in full-mesh manner. This resulted in a number of unused interfaces in the virtual switches that needed to be actually supported by the physical ones and managed by the controller. Therefore, this kind of mapping would better apply to networks for which reliability is more important than throughput or scalability.

Although we did not observe any slow-downs in the controller function, we consider such situation to be possible in networks with more physical switches, as the simulation shows.

6.3.5 Conclusion

After evaluating the aforementioned mappings, we consider that a network implemented according to the mapping where the virtual switches are less than the physical provides better scalability and easier VM provisioning compared to the other mappings. Moreover, considering reliability as the most important factor, an implementation of one-to-one or many-to-many mapping is reasonable, because of predictability of resource usage. Another notable observation of the experiments is that with all networks based on a single physical switch, the controller as

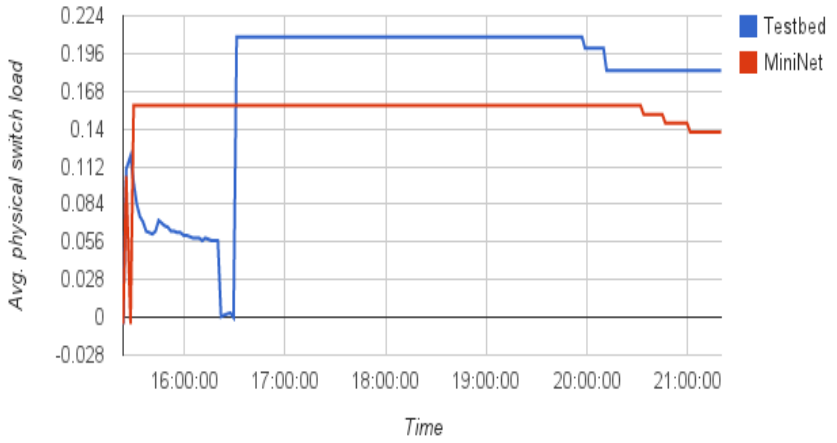

Figure 6.12: Many-to-many Load Comparison for Test Bed and Mininet

well as the switch function is disturbed by OpenFlow table saturation. Anyway, this fact can be useful in other cases, such as home networks, but not cloud networking.

CHAPTER 7

Conclusion and Future Work

Virtualization has always been a building block of cloud computing. Recently, it also found its place in the cloud network through SDNs and OpenFlow. However, at present their implementation in the cloud only refers to management. As the management is centralized and placed in a controller, monitoring the architecture in a distributed manner outside the controller becomes error-prone and does not provide optimal feedback to the management entity. In addition to that problem, the question of how to monitor the virtual switches persists: which metrics are important and how are they related to the traditional ones. Another open question in the context of SDN implementation in the cloud is how to design the best virtual architecture on top of existing physical one, such that a number of requirements related to elasticity and scalbility can be fulfilled. Another goal of the centralised management and monitoring is the easier and more efficient VM provisioning.

In this thesis, we proposed a methodology to monitor the physical and the virtual resources in a SDN implementation based on OpenFlow. Monitoring was performed within the management entity (the OpenFlow controller) centralized for the network. The relation between the traditional and the OpenFlow metrics within the monitoring plug-in was defined and calculated for virtual resources. Some of the OpenFlow specific metrics have strict relationship to the traditional metrics and, therefore, we were able to derive them, and to use them to monitor the virtual entities of the network. However, the SDNs can be described through other metrics, which are not a subject of this thesis. Through the years, there have been only few attempts for centralised management and monitoring of the network. In this thesis, we had shown that this approach is not only possible, but it also performs well in cloud conditions.

The data gathered from a series of experiments with different implementations of SDN topologies allowed us to analyze the correlation between the physical and the virtual resources. Placing physical and virtual switches in different relations and testing these topologies in near-real-life cloud conditions provided us with data to compare the different scenarios behaviour. We discover that a different number of physical and virtual devices benefits different requirements of the environment.

7.1 Future Work

Even though the current implementation of the monitoring plug-in is functional, the development is not considered finished. Some further improvements and extensions would be helpful for the users.

- The scope of the designed plug-in includes only the vital metrics for any monitoring tool. However, for the sake of completeness, a number of other metrics can be considered for implementation. Such metrics include the metrics belonging to the active network metrics class, such as latency and end-to-end jitter. Implementation of this kind of metrics is supposed to happen at the OpenFlow level and in the plug-in. This will improve not only the network application performance, but also the end-use experience.

- As the evaluation of the plug-in shows, it is lightweight and causes negligible delays in the routing function. While testing none fast increase in the delay interval of the routing is observed. In the tests used for evaluation, the largest number of switches used was 30 and in this case the delay was less than one time greater than in the case of 10 switches. Further, a case with large number of switches is present in the integration test scripts. Therefore, we believe that the plug-in will not cause failures in the function of the controller even in the case of considerably larger number of switches. Nevertheless, we see a potential for optimisation in this direction. Applying different optimisation heuristics within the plug-in is considered useful.

- In the current implementation, the REST interface is implemented in the context of the last measurement, and no data from the previous measurements is exposed. It is considered helpful to provide a caching feature of the gathered metrics.

- For the mapping evaluation, we have focused on the case when the network is implemented using one controller. However, the gathered and analysed data have shown that in some cases that is not the optimal setup. Further experiments with more controllers will clear these issues and will further help to define the optimal mapping between physical and virtual resources unambiguously.

- Furthermore, for the mapping implementation, it is considered useful to create and execute experiments with a larger number of switches. In addition, for more precise conclusions, it is better to use a recorded real-world traffic that corresponds to the number of the switches rather to use a benchmark tool.

Bibliography

[1] Management Information Base for Network Management of TCP/IP-based internets: MIB-II.

[2] Integrating the Physical and the Virtual Relevance to SLA Cloud. Technical report, Network Visibility Solutions, 2012.

[3] Bulut Ahmet, Koudas Nick, Meka Anand, Singh Ambuj K., and Divesh Srivastava. Optimization Techniques for Reactive Network Monitoring . *IEEE Transactions on Knowledge and Data Engineering*, 21:1343 – 1357, 2009.

[4] T. Anderson, L. Peterson, S. Shenker, and J. Turner. Overcoming the Internet impasse through virtualization. *Computer,*, 38:34–41, 2005.

[5] Sergio Andreozzi, Augusto Ciuffoletti, Antonia Ghiselli, Demetres Antoniades, Michalis Polychronakis, Evangelos P. Markatos, and Panos Trimintzios. *Integrated Research in GRID Computing*, chapter On the Integration of Passive and Active Network Monitoring in Grid Systems, pages 147–161. Springer US, 2007.

[6] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, David Patterson Gunho Lee, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A View of Cloud Computing. *Communications of the acm*, 53:50–58, 2010.

[7] F.M. Aymerich, G. Fenu, and S. Surcis. An approach to a Cloud Computing network. *Proceedings of the First International Conference on the Applications of Digital Information and Web Technologies*, pages 113–118, 2008.

[8] Ed. B. Davie and J. Gross. A Stateless Transport Tunneling Protocol for Network Virtualization (STT) draft-davie-stt-01.

[9] Aymen Belghith, Bernard Cousin, Samer Lahoud, and Siwar Ben Hadj Said. Proposal for the Configuration of multi-domain Network Monitoring Architecture. *Proceedings International Conference on Information Networking.*, 2011.

[10] Big Network Controller. http://www.bigswitch.com/products/sdn-controller.

[11] Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt. *Pattern Oriented Software Architecture: On Patterns and Pattern Languages*. John Wiley & Sons, 2010.

[12] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. *Future Generation Computer Systems*, 2009.

[13] Rajkumar Buyya, Saurabh Kumar Garg, and Rodrigo N. Calheiros. SLA-Oriented Resource Provisioning for Cloud Computing: Challenges, Architecture, and Solutions. *2011 International Conference on Cloud and Service Computing*, 2011.

[14] Canonical Ltd., http://manpages.ubuntu.com/manpages/lucid/man1/iperf.1.html. *iperf - perform network throughput tests*, 2010.

[15] M. Chmielewski, M. Szyszkowski, and P. Piechowiak. Application of IP multicast in embedded systems with OpenWRT. *8th International Symposium on Communication Systems, Networks & Digital Signal Processing (CSNDSP)*, pages 1–5, 2012.

[16] Yu-Hunag Chu, Yao-Ting Chen, Yu-Chieh Chou, and Min-Chi Tseng. A simplified cloud computing network architecture using future internet technologies. In *2011 13th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2011.

[17] Cisco, http://www.cisco.com/en/US/docs/switches/lan/catalyst6500/ios/12.2SX /configuration/guide/vlans.html. *VLANs*.

[18] Cisco. Cisco Open Network Environment: Network Programmability and Virtual Network Overlays. *Cisco Public Information*, 2012.

[19] Augusto Ciuffoletti. Monitoring a Virtual Network Infrastructure An IaaS Perspective. *SIGCOMM Computer Communication Review*, 40:47–52.

[20] D. Clark, B. Lehr, S. Bauer, P. Faratin, R. SamiI, and J. Wroclawski. Overlay Networks and the Future of the Internet. *Communications & Strategies*, 3:1–21, 2006.

[21] Floodlight Is An Open SDN Controller. Floodlight, 2012.

[22] Paolo Costa, Matteo Migliavacca, Peter Pietzuch, and Alexander L. Wolf. NaaS: Network-as-a-Service in the Cloud. *2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE'12)*, 2012.

[23] Ed. D. Allan N. Bragg D. Fedyk, Ed. P. Ashwood-Smith and P. Unbehagen. IS-IS Extensions Supporting IEEE 802.1aq Shortest Path Bridging. Technical report, 2012.

[24] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef. Web Services on Demand: WSLA-Driven Automated Management. volume 43, pages 136–158, 2004.

[25] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud Computing: Issues and Challenges. *24th IEEE International Conference on Advanced Information Networking and Applications*, pages 27–33, 2010.

[26] Vincent C. Emeakaroha, Ivona Brandic, Michael Maurer, and Schahramn Dustdar. Low Level Metrics to High Level SLAs - LoM2HiS Framework: Bridging the Gap Between Monitored Metrics and SLA Parameters in Cloud Environments. *International Conference on High Performance Computing and Simulation*, pages 48 – 54, 2010.

[27] Emulex. NVGRE Overlay Networks: Enabling Network Scalability for a Cloud Infrastructure.

[28] Fernando Farias, Joao Salvatti, Igor Furtado, and Eduardo Cerqueira. LegacyFlow: A Hybrid SDN Solution to Bring Openflow to Legacy Network Environment. In *http://www.cpqd.com.br/files/pdf/CPqD-WNRP_07_FernandoFarias_LegacyFlow.pdf*, 2011.

[29] Pantou : OpenFlow 1.0 for OpenWRT Accessed: 2012-10. http://www.openflow.org/wk/index.php/pantou_:_openflow_1.0_for_openwrt.

[30] I. Foster, Yong Zhao, I. Raicu, and S. Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *Grid Computing Environments Workshop*, 2008.

[31] Open Wrt Wireless Freedom. https://openwrt.org/. Accessed: 2012-12-27.

[32] M. Gerola. Enabling Network Virtualization in OpenFlow Networks through Virtual Topologies Generalization.

[33] Srinivas Govindraj, Arunkumar Jayaraman, Nitin Khanna, and Kaushik Ravi Prakash. OpenFlow: Load Balancing in enterprise networks using Floodlight Controller. 2012.

[34] ONF Hybrid Working Group. https://www.opennetworking.org/working-groups/hybrid.

[35] E. Haleplidis, S. Denazis, O. Koufopavlou, J. Halpern, and J.H. Salim. Software-Defined Networking: Experimenting with the Control to Forwarding Plane Interface. In *European Workshop on Software Defined Networking (EWSDN)*, 2012.

[36] Evangelos Haleplidis, Spyros Denazis, Odysseas Koufopavlou, and Joel Halpern. Software-Defined Networking: Experimenting with the control to forwarding plane interface. *European Workshop on Software Defined Networking*, 2012.

[37] Jan Hannemann and Gregor Kiczales. Design pattern implementation in Java and aspectJ. *Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 161 – 173, 2002.

[38] Jens Happe, Holger Friedrich, Steffen Becker, and Ralf H. Reussner. A pattern-based performance completion for Message-oriented Middleware. *Proceedings of the 7th international workshop on Software and performance*, pages 165–176, 2008.

[39] Elliotte Rusty Harold. Easier testing with EasyMock Imitate interfaces, classes, and exceptions with an open source mock-object framework. Technical report, IBM, 2009.

[40] Zongjian He and Guanqing Liang. Research and Evaluation of Network Virtualization in Cloud Computing Environment. *2012 Third International Conference on Networking and Distributed Computing*, pages 40–44, 2012.

[41] Brandon Heller, Rob Sherwood, and Nick McKeown. The Controller Placement Problem. *Proceeding HotSDN '12 Proceedings of the first workshop on Hot topics in software defined networks*, pages 7–12, 2012.

[42] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns:Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional, 2004.

[43] Beacon Home. https://openflow.stanford.edu/display/beacon/home. Accessed: 2012-10-25.

[44] http://www.cisco.com/en/US/products/ps6601/products/ios/protocol/group/home.html. Cisco IOS NetFlow. Accessed: 2012-10.

[45] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. *5th international conference on Emerging networking experiments and technologies*, page 1–12, 2009.

[46] Gunjan Beaty Khanna, Kirk A., Gautam Kar, and Andrzej Kochut. Application Performance Management in Virtualized Server Environments. In *Network Operations and Management Symposium*, 2006.

[47] KVM. http://www.linux-kvm.org/page/mainpage. Accessed: 2013-01-13.

[48] Liboftrace. http://www.openflow.org/wk/index.php/liboftrace. Accessed: 2012-11-09.

[49] libvirt: The virtualization API. http://libvirt.org/.

[50] Junjie Liu, Yingke Xie, Gaogang Xie, Layong Luo, Fuxing Zhang, Xiaolong Wu, Qingsong Ning, and Hongtao Guan. Building a Flexible and Scalable Virtual Hardware Data Plane. *IFIP International Federation for Information Processing*, pages 205–216, 2012.

[51] Yan Luo. Cloud Computing Network I/O Virtualization for Cloud Computing. *IEEE Computer Society*, 2010.

[52] Yan Luo. Network I/O Virtualization for Cloud Computing. *IT Professional*, 12:36–41, 2010.

[53] Maestro-Platform. http://code.google.com/p/maestro-platform/.

[54] Jon Matias, Borja Tornero, Alaitz Mendiola, and Eduardo Jacoband Nerea Toledo. Implementing Layer 2 Network Virtualization using OpenFlow: Challenges and Solutions. *European Workshop on Software Defined Networking*, 2012.

[55] R. McIlory and J. Sventek. Resource Virtualisation of Network Routers. *2006 Workshop on High Performance Switching and Routing Date of Conference:*, 2006.

[56] Ling Mei, Yiduo Liu, Xing Pu, and Sankaran Sivathanu. Performance Measurements and Analysis of Network I/O Applications in Virtualized Cloud. *IEEE 3rd International Conference on Cloud Computing*, pages 59 – 66, 2010.

[57] P. Mell and T. Grance. Draft nist working definition of cloud computing. 2009.

[58] MiniNet. http://mininet.github.com/. Accessed: 2012-09-29.

[59] Yukihiro Nakagawa, Kazuki Hyoudou, and Takeshi Shimizu. A management method of IP multicast in overlay networks using OpenFlow. *Proceedings of the first workshop on Hot topics in software defined networks*, pages 91–96, 2012.

[60] Nicira. http://nicira.com/.

[61] Márcio Nogueira, João Melo, Jorge Carapinha, and Susana Sargento. Virtual Network Mapping into Heterogeneous Substrate Networks. *Computers and Communications*, pages 438 – 444.

[62] NOX. http://www.noxrepo.org/nox/about-nox/.

[63] Karsten Oberle, Manuel Stein, Thomas Voith Georgina Gallizo, and Roland Kübert. The Network Aspect of Infrastructure-as-a-Service. *14th International Conference on Intelligence in Next Generation Networks (ICIN)*, pages 1–6, 2010.

[64] T. Okumura and D. Mosse. Virtualizing network I/O on end-host operating system: operating system support for network control and resource protection. *IEEE Transactions on Computers*, 53:1303–1316, 2004.

[65] ONF. OpenFlow Specification version 1.2. Technical report, Open Network Foundation, 2011.

[66] OpenVZ. http://en.wikipedia.org/wiki/openvz.

[67] Oprofile. http://oprofile.sourceforge.net/news/.

[68] John Panneerselvam, Lu Liu, Richard Hill, Yongzhao Zhan, and Weining Liu. An Investigation of the Effect of Cloud Computing on Network Management. *2012 IEEE 14th International Conference on High Performance Computing and Communications*, pages 1794 – 1799.

[69] Ch. Z. Patrikakis, Y. Despotopoulos, A. M. Rompotisand, N. Minogiannis, A. L. Lambiris, and A. D. Salis. An Implementation of an Overlay Network Architecture Scheme for Streaming Media Distribution. *Proceedings of the 29th EUROMICRO Conference "New Waves in System Architecture"*, 2006.

[70] Ben Pfaff, Justin Pettit, Teemu Koponen, and Keith Amidon Martin. Extending Networking into the Virtualization Layer. *8th ACM Workshop on Hot Topics inNetworks (HotNets-VIII).New YorkCity,NY*, 2009.

[71] Zhikui Wang Sharad Singhal-Kang G. Shin Pradeep Padala, Xiaoyun Zhu. Performance Evaluation of Virtualization Technologies for Server Consolidation. Technical report, Enterprise Systems and Software Laboratory HP Laboratories, 2007.

[72] Multilayer Open Virtual Switch Production Quality. http://openvswitch.org/.

[73] Mininet:network protopyping. http://yuba.stanford.edu/foswiki/bin/view/openflow/mininet.

[74] L. Richardson and S. Ruby. *RESTful web services*. O'Reilly, 2008.

[75] Aris Cahyadi Risdianto and Eueung Mulyana. Implementation and Analysis of Control and Forwarding Plane for SDN. In *7th International Conference on Telecommunication Systems, Services, and Applications (TSSA)*, 2012.

[76] Piotr Rygielski and Samuel Kounev. Network Virtualization for QoS-Aware Resource Management in Cloud Data Centers: A Survey. In *PIK - Praxis der Informationsverarbeitung und Kommunikation*, 2012.

[77] Peter Sempolinski and Douglas Thain. A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus. *2nd IEEE International Conference on Cloud Computing Technology and Science*, pages 417–426, 2010.

[78] Hideyuki Shimonishi, Shuji Ishii, Yasunobu Chiba anf Toshio Koide, Masahiko Takahashi, Yasuhito Takamiyaand, and Lei Sun. Helios: Fully distributed OpenFlow controller platform. Technical report, Global Environment for Network Innovations (GENI).

[79] Myung-Ki Shin, Nam Ki-Hyuk, and Hyoung-Jun Kim. Software-defined Networking(SDN): A Reference Architecture and Open APIs. In *ICT Convergence (ICTC), 2012 International Conference on*, 2012.

[80] Simple Network Access Control (SNAC). http://www.openflow.org/wp/snac/.

[81] Aameek Korupolu Singh, Madhukar R., and Dushmanta Mohapatra. Server-Storage Virtualization: Integration and Load Balancing in Data Centers. In *High Performance Computing, Networking, Storage and Analysis*, 2008.

[82] Build software better together. https://github.com/. Accessed: 2012-09-08.

[83] M. Steinder, I. Whalley, D. Carrera, I. Gaweda, and D. Chess. Server virtualization in autonomic management of heterogeneous workloads. In *10th IFIP/IEEE International Symposium on Integrated Network Management*, 2007.

[84] April 2012 Open Networking Summit. http://www.opennetsummit.org/archives-april2012.html.

[85] Omesh Tickoo, Ravi Iyer, Ramesh Illikkal, and Don Newell. Modeling virtual machine performance: challenges and approaches. *Newsletter ACM SIGMETRICS Performance Evaluation Review*, 37:55–60, 2009.

[86] TP-LINK. *TL-WR1043ND Wireless N Gigabit Router*.

[87] TP-LINK. Ultimate Wireless N Gigabit Router TL-WR1043ND. Technical report, TP-LINK, http://www.tp-link.com/en/products/details/?model=TL-WR1043ND, 2011.

[88] D. Turull, M. Hidell, and P. Sjödin. libNetVirt: the network virtualization library. *Proceedings of the Communications (ICC)*, pages 5543–5547, 2012.

[89] Oracle VM VirtualBox. https://www.virtualbox.org/.

[90] VMware. http://www.vmware.com/.

[91] M.A. Vouk. Cloud computing — Issues, research and implementations. In *Proceedings of the 30th International Conference on Information Technology Interfaces*, 2008.

[92] Guohui Wang and T. S. Eugene Ng. The Impact of Virtualization on Network Performance of Amazon EC2 Data Center. *IEEE INFOCOM 2010*, 2010.

[93] Art Wittmann, Lorna Garey, Andrew Conry-Murray, and Heather Vallis. Monitoring and Measuring Cloud Provider Performance. Technical report, InformationWeek Reports, 2012.

[94] Xen. http://www.xen.org/products/xenhyp.html.

[95] Praveen Yalagandula, Sung-Ju Lee, Puneet Sharma, and Sujata Banerjee. Correlations in End-to-End Network Metrics: Impact on Large Scale Network Monitoring. *INFOCOM Workshops*, pages 1–6, 2008.