

UNIVERSITÀ DI PISA
Scuola di Dottorato in Ingegneria “Leonardo da Vinci”



Corso di Dottorato di Ricerca in
INGEGNERIA DELL'INFORMAZIONE

Tesi di Dottorato di Ricerca

Software Defined Networking
for resource allocation and monitoring:
virtualization and hardware acceleration

Lisa Donatini

Anno 2016

UNIVERSITÀ DI PISA

Scuola di Dottorato in Ingegneria “Leonardo da Vinci”



Corso di Dottorato di Ricerca in
INGEGNERIA DELL'INFORMAZIONE

Tesi di Dottorato di Ricerca

Software Defined Networking for resource allocation and monitoring: virtualization and hardware acceleration

Autore:

Lisa Donatini _____

Relatori:

Prof. Stefano Giordano _____

Ing. Davide Adami _____

Anno 2016

Sommario

Le reti di telecomunicazioni sono presenti in modo sempre più pervasivo nella nostra vita di tutti i giorni, e sempre più persone le usano per un numero crescente di operazioni.

Gli utenti hanno aspettative sempre maggiori per le performance della rete, usandole per diverse applicazioni, con livelli sempre più alti di interattività.

Le reti quindi si trovano ad avere non solo traffici sempre maggiori, e differenti pattern di traffico, ma anche una domanda crescente in termini di prestazioni offerte.

In questo scenario, diviene di fondamentale importanza identificare le aree dove apportare modifiche, e le tecnologie da sfruttare e implementare in questo processo.

In questa tesi, vengono esplorate le possibilità offerte dalle nuove tecnologie di virtualizzazione: nuovi approcci che permettono di virtualizzare le reti, vedendole come risorse fisiche sulle quali costruire funzioni che possono essere indipendenti dall'infrastruttura sottostante, esattamente come già accade con i sistemi operativi per i computer, che offrono all'utente una versione virtualizzata delle risorse hardware disponibili.

In particolare, in questa tesi, ci si concentra sul concetto di Software Defined Networking, e su come questo approccio possa essere usato nella pratica per fornire risposte ad alcune questioni ancora aperte.

Allo stesso tempo, riteniamo che al fine di operare su reti ad alte prestazioni e con throughput di rilievo, ci sia bisogno di basare le considerazioni, le decisioni da prendere, su dati il più possibile precisi, forniti da strumenti in grado di raggiungere alte risoluzioni. Questo tipo di azioni richiedono l'utilizzo di hardware ad alte prestazioni per la misura e il monitoraggio, e anche questo aspetto è stato tenuto in considerazione in questo percorso di ricerca.

Abstract

Communication networks are more and more present in everyday life, as more and more people use them for an increasing number of operations.

Users have growing expectations about network performances, while they use them with different applications, with increasing levels of interactivity.

Networks not only have to deal with higher traffics, different traffic patterns, new demands, but also with higher requirements for performing operations.

In this scenario, it becomes of fundamental importance to identify novel promising technologies, and understand when, where and how to deploy them in the most effective ways.

In this thesis, we explore the possibilities offered by virtualization technologies: novel approaches that allow to virtualize networks, seeing them as general physical resources on which to run functions that can be separated from the real underlying infrastructure, just as it happens with the well-known operating systems for computers, that offer to the user a virtualized version of the pool of resources available.

In particular, in this thesis, we focused on studying Software Defined Networking, and how such technology can be deployed to give answers to some open issues in networking.

At the same time, we have always kept in mind that in order to perform high performance operations on networks that experience high throughputs, we need to base our calculations and decisions on precise data, and have tools that allow to reach higher precisions and resolutions. These kinds of actions require the deployment of high performing hardware for measuring and monitoring, and we have kept also this aspect in consideration in our research.

Contents

1	Introduction	1
1.1	Motivations	1
1.1.1	Virtualization Technologies	1
1.1.2	Software Defined Networking for LTE	2
1.2	Contribution	3
1.2.1	SDN-based orchestration	3
1.2.2	VDI	4
1.2.3	SDN-based solutions for LTE	4
1.2.4	Hardware acceleration	4
1.3	Thesis Organization	4
2	Virtual Desktop Infrastructure	7
2.1	Desktop Virtualization	7
2.1.1	Hypervisor	8
2.1.2	Types of VDI	9
2.1.3	Choosing VDI	9
2.2	VDI system	10
2.3	Infrastructure for measurements	15
2.4	Measurement and traffic characterization	16
2.4.1	Traffic graphs	16
2.4.2	Packet length distribution	24
3	Software Defined Networking for resource allocation and monitoring	35
3.1	SDN-DC Orchestrator	36
3.2	Architecture	37
3.2.1	SDN-DC Orchestrator Design	38
3.2.2	Topology discovery	40
3.2.3	SDN-based Network Monitoring and Load Estimation	40
3.2.4	Resource Selection and Composition Algorithms	43
3.3	System performance	45

3.3.1	Performance evaluation via simulations	45
3.3.2	Simulation Results	47
3.4	Testbed Set-Up and System Validation	47
3.4.1	Testbed Setup	47
3.4.2	Testbed operation and message flow	49
3.4.3	System Validation	51
3.4.4	Evaluation of Traffic Monitoring Strategies	58
3.4.5	Aggregate results	60
3.5	Final remarks	65
4	Software Defined Networking for LTE	69
4.1	Introduction to LTE	69
4.2	SDN for Traffic Monitoring	70
4.3	Testbed	74
4.4	POX Controller Configuration	75
4.5	Experimental results	78
4.6	Practical scenarios	80
5	Hardware acceleration	83
5.1	NetFPGA SUME	83
5.2	Open Source Network Tester	84
5.2.1	Generation	85
5.2.2	Monitoring	86
5.2.3	Final remarks	86
6	Conclusions	93
	References	95

List of Figures

2.1	VMware View Infrastructure	13
2.2	Initialization of connection - Ethernet	17
2.3	Initialization of connection - Wi-Fi, ADSL	17
2.4	Flow graph: messages exchanged when opening a new session	18
2.5	Traffic graphs for CDburner - Ethernet	19
2.6	Traffic graphs for CDburner - Wi-Fi ADSL	19
2.7	Traffic for videostreaming - Ethernet	20
2.8	Traffic for videostreaming - Wi-Fi ADSL	20
2.9	Traffic graphs for LibreOffice Draw - Ethernet	21
2.10	Traffic graphs for LibreOffice Draw - Wi-Fi ADSL	21
2.11	Traffic graphs for LibreOffice Calc - Ethernet	22
2.12	Traffic graphs for LibreOffice Calc - Wi-Fi ADSL	23
2.13	Traffic graphs for LibreOffice Math - Ethernet	23
2.14	Traffic graphs for LibreOffice Math - Wi-Fi ADSL	24
2.15	Traffic graphs for LibreOffice Impress - Ethernet	25
2.16	Traffic graphs for LibreOffice Impress - Wi-Fi ADSL	26
2.17	Traffic graphs for LibreOffice Write - Ethernet	27
2.18	Traffic graphs for LibreOffice Write - Wi-Fi ADSL	27
2.19	Traffic when closing connection	28
2.20	Flow graph: messages exchanged during closing of connection	29
2.21	Packet length distribution for connection initialization	30
2.22	Packet length distribution for CD burner application	30
2.23	Packet length distribution for LibreOffice Draw	31
2.24	Packet length distribution for LibreOffice Calc	31
2.25	Packet length distribution for LibreOffice Math	32
2.26	Packet length distribution for LibreOffice Impress	32
2.27	Packet length distribution for LibreOffice Write	33
2.28	Packet length distribution for video application	33

3.1	DC resource infrastructure	36
3.2	Orchestration for network and computing resources	37
3.3	SDN-DC Orchestrator: system design and orchestration components	39
3.4	Different resource allocation strategies	44
3.5	Evolution of blocking probability	48
3.6	Testbed setup	49
3.7	SDN orchestrator operation	50
3.8	Screenshot of the state of Virtual Machine on one of the Xen Servers	51
3.9	Extract from the flow table of a core switch	52
3.10	Evolution of blocking probability	53
3.11	Number of VMs per server	54
3.12	Percentage of path occupation	55
3.13	Percentage of core link occupation	56
3.14	Blocking Probability for each RSCA	57
3.15	Load with history weight 0.2	59
3.16	Load with history weight 0.5	60
3.17	Load with history weight 0.8	61
3.18	Load with history weight 0.2	62
3.19	Load with history weight 0.5	63
3.20	Load with history weight 0.8	64
3.21	Link load for $\alpha=0.2$ (ND-BF)	65
3.22	Link load for $\alpha=0.2$ (ND-BF)	66
3.23	Link load for $\alpha=0.2$ (ND-WF)	66
3.24	Link load for $\alpha=0.5$ (ND-WF)	67
4.1	Distributed monitoring	72
4.2	Topology	74
4.3	Testbed	75
4.4	Network Segmentation	76
4.5	Traffic Redirect	78
4.6	Reachability test from IP 192.168.1.3 to IP 192.168.2.3	79
4.7	Test for isolation between IP 192.168.2.3 and IP 192.168.2.4	79
4.8	Redirecting of HTTP traffic	80
5.1	NetFPGA SUME	88
5.2	OSNT: architecture for generation	89
5.3	OSNT: architecture for monitoring	90
5.4	Traffic for eNodeB	91

List of Tables

3.1	Simulation parameters	46
3.2	Available paths in the topology	52
3.3	Time for VM setup: per server mean values (ms)	57
3.4	Time for VM setup: per server CI at 95% (ms)	58
3.5	Parameters for randomly generated test patterns	61
3.6	Results with Best Fit algorithms	62
3.7	Results with Worst Fit algorithms	64

Introduction

The motivations for this thesis spread from the observation of the most recent evolutions in the world of networking. It is undeniable that virtualized technologies are changing the way we use and the way we think networks.

These changes involve demands for higher performances of the network, faster processing, higher degree of flexibility and reconfigurability.

Hence, we felt the need to investigate some of the most promising solutions, and design new ways to deploy their potentials.

1.1 Motivations

1.1.1 Virtualization Technologies

Data Centers (DCs) are a good example of a growing technology that plays an ever bigger role in everyday's life. The growing performances of networks, the higher bandwidths, the higher reliability, are enabling the use of such systems.

Virtualization technologies (e.g., VMware [1], XEN [2]) have enabled Data Center functionalities to evolve, allowing them to process and store large amount of data, and to host services in a scalable, flexible, elastic, and seamless way.

Virtualization technologies enable cost-effective utilization of hardware resources, and are putting solid basis for novel changes in the approach to usage of computing resources, favouring the spread of solutions for remote computing.

At the same time, software virtualization is demanding additional requirements from virtualized Data Centers.

In fact, enabling multiple Virtual Machines (VMs) to simultaneously run on a set of servers, accessed by remote users, means dealing with highly dynamic traffic flows generated by the various VMs as they start, expand, reduce, run, migrate. Moreover, VMs can run all kinds of applications, that may generate very variegated kinds of traffic, and with different communication patterns, while using network capacities of switches and link in a concurrent way.

Due to the increasing demand for applications that operate in virtualized DC environments, in the latest years solutions for high bandwidth and low latency have become a really hot research topic [3]. Different network architectures have been designed to improve network performance while trying to maintain scalability of the network infrastructure [4][5][6].

Other works have gone in the direction of studying new routing solutions [7], using optimized transport protocols, in order to address high throughput and low latency requirements [8]. New traffic engineering strategies have been proposed, aiming to handle extremely variable and unpredictable communication patterns [9][10].

It is evident that in order to meet the network requirements of cloud computing and virtualization, we need more advanced network tools, that are able to manage both the computing and the network resources in appropriate ways.

This is why we need an orchestration system that is able to arrange, manage and coordinate the different sets of resources present in the domain [11][12][13].

In the light of these considerations, it is easy to understand how a paradigm such as Software Defined Networking (SDN) can give answers to the needs of better network control of management [14].

SDN uses a centralized and highly programmable model, that separates the control plane from the data plane: a controller manages the whole network, instantiating rules in all switches for each new flow.

Such a solution can bring great advantages to systems such as Data Centers, where all the switches are known, and specific rules can be programmed so that the system evolves depending on the instantaneous conditions.

Moreover, other features of SDN make it an ideal candidate for such use. For example, tools like OpenFlow [15], a protocol that allows to implement in practice the paradigm of Software Defined Networking, provide functions to measure traffic statistics on a per-port base, making available data that can be processed in order to achieve better network management [16][17][18].

The statistics collected by the switches can be used to estimate the state of the network, which can be the starting point to plan and enforce forwarding rules.

The use of SDN-based solutions is discussed and in some cases deployed [19]. However, its effective usability within contexts such as Virtualized Data Centers is still an open research issue [20] [21].

1.1.2 Software Defined Networking for LTE

LTE (Long Term Evolution) is a growing trend, with this technology becoming more and more common.

LTE has the potential to bring new mobile applications to a global audience, offering higher bandwidth than ever before to mobile users, which also causes the spread of new ways to use mobile networks, increasing again the demand of resources.

This increments even more the market of smartphones and tablets, since more actions are now possible with this kind of devices, and this in turn generates again an even more accentuated increase of Internet mobile traffic.

The novelty, the growing demand, the higher throughput, the increase not just in traffic, but in users expectations as well: all these things together present new challenges to operators, that need to deploy resources efficiently, and that need to be able to test their solutions, measure their traffic, and perform analysis on it.

Having a correct overview of the network is important for tuning network parameters and optimizing resource usage to meet users' expectation.

It became immediately clear that Software Defined Networking based approaches could be considered for creating platforms for the monitoring and testing of LTE networks.

SDN has a big potential to enable network innovation and create new possibilities, and thus help the realization of new approaches and address persistent problems in networking [22]. Also, this paradigm gives network operators more control on their infrastructure, allowing ready customization and optimization, thereby reducing the overall capital and operational costs.

To this purpose, we made a plan to study in more depth the challenges that arise from monitoring real deployed LTE network, in order to target specific issues and offer solutions.

1.2 Contribution

This thesis provide the following contribution:

1.2.1 SDN-based orchestration

A main contribution of this thesis is the study, implementation and testing of a solution for an orchestrator for computational and network resources in a Data Center environment, based on a Software Defined Networking approach.

This system is based on functions for resource selection, that base their decisions upon statistic data continuously collected on the devices in the system.

Such system was not only simulated for verification, but a downscaled version was built and tested, in order to obtain realistic data on how changes in parameters of the algorithms would affect the final outcome, in terms of reaching the goals of the orchestration process. The main contribution of this work lies in the comprehensive design and experimental evaluation of an SDN-based orchestrator for the coordinated deployment of both network and computing DC services. An orchestration system has been implemented and experimentally evaluated, exploiting SDN/OF capabilities.

This orchestrator leverages statistics about traffic traversing DC switches and links, collected thanks to functions present in the OpenFlow protocol. Once collected, these statistics are used to derive information about utilization trends, and make decisions about how to manage and coordinate VM placements on physical servers, and about

the forwarding path to/from the VM to be set-up while alleviating possible congestions despite the unpredictability of traffic cloud DCs.

The continuous monitoring of traffic allows to build trend estimations that give valuable indications to the orchestrator of the system, that is able to balance in a more appropriate way the load on the different links, resulting in better acceptance rate for new requests.

1.2.2 VDI

Various sets of measurements have been carried on a real VDI system, built in a project spreading from the collaboration of University of Pisa with associations of local companies.

These tests allowed to have clearer views of what the VDI traffic is like, which also led to improved test patterns for research on Software Defined Networking.

1.2.3 SDN-based solutions for LTE

Thanks to a continuous collaboration with TILAB (Telecom Italia Lab), it has been possible to investigate scenarios of LTE monitoring that needed more performing and customizable solutions.

This thesis offer a contribution to some issues that have arisen in the observation of the requirement for monitoring, and offers tailored solutions based on virtualization

1.2.4 Hardware acceleration

If on one side virtualization offers novel efficient ways to manage networks, a main requirement for managing a network in an effective way is to have a detailed, precise, updated image of the state of the network. Given the fact that network bandwidths keep increasing, and the traffics are becoming more and more variegated, hardware solutions are even more valuable to give best results when it comes to measuring, timestamping, filtering with high resolutions and at high speed. For this reason, during this PhD additional works have been carried involving developing of hardware acceleration projects.

1.3 Thesis Organization

This is how this thesis is organized.

Chapter 2 discusses the characteristics of VDI (Virtual Desktop Infrastructure) systems, and portrays the results we obtained performing measurements on a real infrastructure. It also describes the infrastructure that was built and how it was used.

Chapter 3 describes the study and realization of a resource allocation system based on SDN (Software Defined Networking), for Cloud Data Center environment. This chapter describes the initial premises, the architecture and algorithms developed, and the design

and the sets of specific tests that were carried, both in simulation and in a real testbed that was built.

Chapter 4 offers insights on how SDN-based solutions can find applications in various fields of high interests, like that of LTE (Long Term Evolution) monitoring. Some issues of LTE managing are discussed, and solutions are described.

Chapter 5 summarizes the work that was carried thanks to a research exchange with University of Cambridge: this work focused on hardware solutions for traffic monitoring and testing, and can find several applications, including for example in monitoring for LTE systems.

Chapter 6 finally concludes this thesis, with a summary of the work done and the results obtained.

Virtual Desktop Infrastructure

In this thesis we are studying, developing, testing virtualized solutions for resource allocation and monitoring.

Virtualization can bring several advantages, such as less cost for hardware, greater ease of management, efficiency in resource usage, energy saving. For this reason, in later years virtualization of servers has become more and more used.

2.1 Desktop Virtualization

A recent emerging concept has been that of Desktop Virtualization, a solution where users will receive as a service the access to a PC, virtualized inside a Data Center. One driving force of this new development has surely been the more and more pervasive presence of networks with high bandwidth, along with faster and more secure connections. Moreover, portable devices that can be used as clients are more and more common.

Soon, the development, the more advanced technologies employed in cloud computing, the tangible advantages, has led to an increase of popularity of these solutions, encouraging more and more companies and organizations to turn to cloud solutions, using all those solutions known as "as a service": Infrastructure as a Service, Platform as a service, etc, since the benefits range from the economical ones (less need for dedicated hardware) to the managing ones (less need to think of maintenance, updating, etc).

Today we use various kinds of virtualization, often jointly, as we have virtualization of servers, of storage, of desktop, etc.

A VDI (Virtual Desktop Infrastructure) system aims at offering "Desktop as a Service", using thin clients as terminals of virtual systems, where actually all the processing is completely in the data center, and the client is only used to visualize and interact.

Desktop Virtualization is a software technology that separates the desktop environment and its associated application software, from the physical device that the user uses to access it.

The desktop environment is in fact provided as a service by a server, while the user only needs a "thin client" to access it.

A thin client, sometimes called a lean client, is a low-cost lightweight device, built to connect from remote to a server. There can be many kinds of thin clients, but the general idea is that these clients have and need very few resources, and depend on the server to which they connect for functioning.

For example they may lack completely storage functions (in this case sometimes they can be referred to as zero clients), and usually have very little computational resources on board.

VDI systems feature a complex architecture, that includes authentication system, virtualization of hardware and software components, managing of the network, etc.

From an architectural point of view, the main difference between VDI and a more traditional server-based computing structure (SBC), is the layer of virtualization of the hardware: SBC, invented by Citrix and launched in 1995, basically hosts multiple user sessions on a single operating system. Each server has a single operating system, and each user session runs its own user environment and applications.

In VDI instead, invented by VMware in 2008, each user session runs on a separate operating system, running as a Virtual Machine hosted by a hypervisor. Thus, each server can run different Virtual Machines, either with a same or different operating systems. This solution is perceived by the user as having an own machine to work on, with a total isolation between different users' sessions, as they happen on different VMs.

Somehow, this is one step beyond the regular virtualization of hardware that we experience with laptops, PCs, etc, where we are delivered a desktop that virtualizes the hardware underneath.

The difference, in the case of VDI, is that the hardware is in a remote location, and the applications are running in remote instead of running locally, thus the desktop is presented to the user thanks to RDP (Remote Desktop Protocols).

2.1.1 Hypervisor

A hypervisor (sometimes called VMM, virtual machine monitor) is a component that creates and runs virtual machines. It can be implemented in software, firmware or hardware, or be a combination of more parts.

Hypervisors run on host machines, that can host more virtual machines (also known as guest machines).

The hypervisor is in charge of presenting these guest operating systems with a virtual operating platform, and manages the execution of the guest operating systems.

In a virtualized system, multiple instances of one or more operating systems share the virtualized hardware resources.

There are two main categories of hypervisors, although the distinction nowadays is no longer so clear all the times.

- *Type-1 hypervisor* (also known as *native hypervisor* or *bare-metal hypervisor*): these hypervisors run directly on the hardware of the host machine, thus are meant to be installed before any other thing. The hypervisor itself acts like an operating system,

controls the hardware, and manages guest operating systems. This is the reason why they are sometimes called *bare metal hypervisors*. A guest operating system runs as a process on the host. Some of the most common type 1 hypervisors are Oracle VM Server for SPARC, Oracle VM Server for x86, the Citrix XenServer, VMware ES-X/ESXi and Microsoft Hyper-V 2008/2012.

- *Type-2 hypervisors* (also known as *hosted hypervisors*): these hypervisors run on a conventional operating system, exactly like any other software. Type-2 hypervisors are used in order to abstract guest operating systems from the host operating system. Some of the most common type 2 hypervisors are VMware Workstation, VMware Player, VirtualBox and QEMU.

2.1.2 Types of VDI

There are two kinds of VDI: the server-hosted (or centralized) and the client-side.

Server-hosted VDI

A server-hosted VDI is a dedicated remote desktop solution, that allows remote access to a desktop (usually Linux or Windows).

All the instantiated Virtual Machines work inside the data center. The virtual infrastructure thus increases the safety and robustness of the system, and makes resource deployment easier.

Implementing a server-hosted VDI means that the desktop is no longer bound to any location or workstation, nor to the configuration of the client that is used. Each user has its own customized desktop, and all data processing and applications are in the Data Center. Although for some specific applications some of the processing can happen in the clients as well. This is for example what happens with applications for 2D or 3D graphics: in this case GPUs (Graphics Processor Units) are used in the client, in order to process the images to be accessed from remote. Same for video decoding, where the flow of data is passed to the client, and locally decoded

These examples make it clear that knowing the kind of traffic that we expect to have, is fundamental to understand which is the best solution to deploy.

Client-side VDI

A client-side VDI, instead, is a model where all the desktop resources are encapsulated in a Virtual Machine, that then runs on the peripheral devices. More VMs can run on the same device. A hypervisor takes care that the VMs will be independent from the hardware they run on, and independent from each others as well.

2.1.3 Choosing VDI

A first step thus, was to study VDI, Virtual Desktop Infrastructure, since this technology is evolving and is of particular interest in many environments for its characteristics.

The main advantage of this approach are:

- thin clients are not as vulnerable to malware attacks as regular PC
- thin clients require less maintenance work, less assistance, and can have longer life cycle, since all that is needed is to maintain and update the central servers
- thin clients consume less energy than regular PCs
- thin clients are less expensive

Thanks to virtualization, all the applications commonly found on any computer can be used disregarding the local operating system on the client, the local characteristics of hardware, and without downloading, installing, licensing new applications in each terminal.

In some cases, even in VDI approaches some applications can be run locally, and files can be stored locally if needed (and if the local CPU and storage resources allow for it).

But at the same time there can be "zero clients", i.e. systems where the client is not supposed to have enough resources to perform such operations locally, and only consists of tools for sending inputs (mouse, keyboard, touch screen) and tools for showing the info to the user (generally, a screen).

As for the operating system made available for the user, that can be just as possible: a client can launch and use from remote any complete operating system, that is created remotely starting from a centralized image.

A single image can be used by more users. This gives great advantages, since complete operating systems, with all their applications, can be available for more users in a short time and in a much more secure environment than with local installations.

It is easy to see that, while all these features can be interesting even for home computers and personal users, they make VDI technology particularly even more interesting for companies and enterprises, that may outsource the software and hardware they need, with the advantage of having a safer environment, that is easier to maintain and keep updated.

This is why in order to use VDI it is even more fundamental that the network is able to provide excellent service, since any delay, congestion, wrong sizing of resources, can cause severe degradation in the user experience, resulting in such architectures to be practically unusable.

Our first aim, thus, was to study VDI (Virtual Desktop Infrastructure) traffic: its characteristic and features, in order to be able to understand it and build realistic models of such traffics.

2.2 VDI system

We had the chance to take part in project ARPEGGIO: Architetture di Rete e Processing per virtual desktop: esperimenti, misure e modelli di sistemi VDI (Network architectures and processing for virtual desktop: experiments, measurements and models for VDI systems).

This project was part of a series of initiatives by the Innovation Center INNOPAPER, that was pushed in this direction by the interest expressed by many local companies for these solutions.

In this section, we describe the components in the VDI system that we tested in the project ARPEGGIO.

The project deployed a VMware solution.

VMware Workstation 8 was initially used in the project (later updated to more recent versions). This is a closed commercial solution, but the reason for using it was its advanced functionalities and management tools, for which it can be considered state of the art, and the fact that is one of the most widespread solutions of this kind.

VDI VMware is one of the main component of the VMware pack. Workstation allows to create and manage Virtual Machines both locally, both in remote. It is also possible from one machine to connect to any other where Workstation has been installed, and manage Virtual Machines from there.

VMware provides various remote-desktop capabilities for users who need a virtualization technology, including:

- VMware vSphere for Desktops (this includes ESXi, VMware's hypervisor)
- VMware vCenter Server (for managing the virtualization environment)
- View Composer (advanced View management, with automation and cloning)
- View Manager (for managing the View Environment)
- View Client (for communication between View and the desktop OS)
- VMware ThinApp (application virtualization)
- View Persona Management (for the management of user profiles)
- vShield Endpoint (component for security - e.g.: antivirus)

Workstation supports some of these tools for server side virtualization, such as ESXi (the hypervisor), vCenter Server.

Let us go into some more details about some of the fundamental tools composing the VMware package, and how they were deployed in the ARPEGGIO architecture.

ESXi

ESXi is a type 1 hypervisor, and is the latest version of the *light hypervisor* of VMware, and nowadays one of the most successful softwares for managing of Virtual Machines.

It can be installed directly on a machine, without any operating system previously installed, although it can also run upon an existing operating system, if that better fits with the specific need of the system to build.

ESXi can be installed from an ISO image provided by VMware, and during the set-up it can be configured to decide access data, network settings, etc.

Since ESXi is the virtualization server, all virtual machines and operating systems are installed on it. In order to manage, create, install all these VMs, we need different component of the vSphere package, like vSphere Client or vCenter Server.

VSphere Client

VSphere Client is used by the client machine to connect to ESXi server and perform management tasks, like installing and managing virtual machines on ESXi server.

It represents the transition from ESX to ESXi, as ESX architecture consisted of a virtualization kernel (*vmkernel*), and a management partition known as Console Operating System (CSO). While ESXi does not have the CSO, allowing direct access to the *vmkernel*.

VCenter Server

vCenter server is similar to vSphere client but it is for the server side. It consists of a management interface that offers a centralized tool, providing a detailed image of the VMs and allowing to manage virtual machines and ESXi hosts centrally. For example, you can easily clone existing virtual machine in vCenter server. It also provides tools for fault tolerance of the virtual infrastructure, and can be used in order to control user access.

vCenter server can be installed on Windows Server or Linux Server.

vCenter server provides many additional features that can be very interesting especially for enterprise environment.

In ARPEGGIO infrastructure, vCenter Server was installed on a machine equipped with Linux, and during the installation an SQL Server 2008 R2 instance is automatically installed.

View Manager 5.0

View Manager 5.0 [30] was the tool used to carry the first tests. This includes a series of modules that allow to connect to the system.

VMware View includes the following key components:

- View Connection Server
- View Agent
- View Client
- View Client with Offline Desktop
- View Portal
- View Administrator
- View Composer

View Connection Server

View Connection Server is the component that manages secure access to virtual desktops. It is the connection broker, and provides advanced management capabilities: it receives the connection requests, and forward them to the most appropriate machine to host them.

It can be installed as one of the following instances:

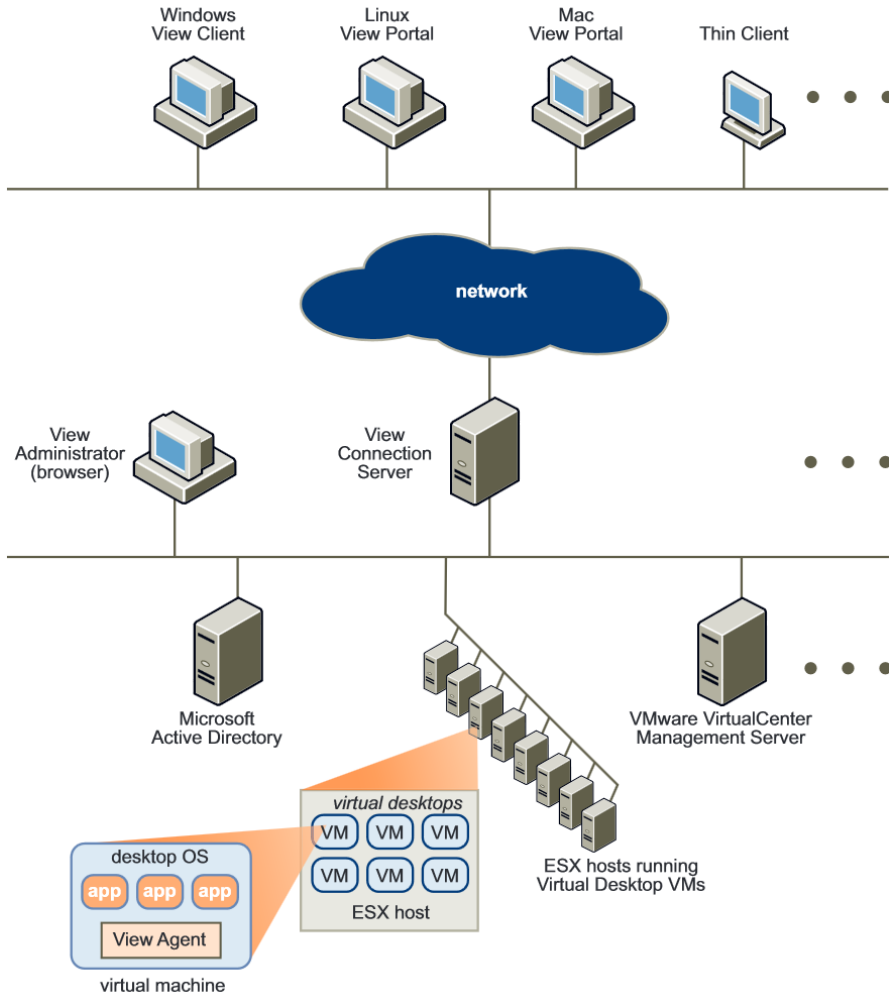


Figure 2.1: VMware View Infrastructure

- *Standard*: this instance can be used as stand-alone
- *Replica*: this instance is installed as a second or subsequent ViewConnection Server in a high-availability group.
- *Security Server*: this instance implements a subset of the View Connection Server functionality. It is only used in specific cases where security is particularly crucial.

The software View Connection Server was installed on a Virtual Machine, equipped with operating system Windows Server 2008 R2 (64-bit).

The instance type is selected during the installation.

View Agent

View Agent runs on each virtual desktop and is used for session management. It can also be installed on a virtual machine template to use for cloning, so that all virtual desktops created from that template will automatically include it.

View Agent also provides, if enabled, *single sign-on* functionality, that allows a user to be automatically logged in when they connect to their virtual desktops, using the same credentials they use to log in to their domain.

View Client

View Client runs on a Windows PC as a native Windows application, and allows users to connect to their virtual desktops through View.

This component connects the user to a View Connection Server, and after logging in, users can select a virtual desktop from a list of authorized ones.

Offline Desktop

Offline Desktop is a tool especially for users who wish to connect from mobile devices: it allows to "check out" an instance of a desktop, and keep on using it on own local resources even in the absence of connection.

View Portal

View Portal is similar to View Client, except that it provides View user interface through a Web browser, and all necessary View software is installed automatically on the client through the Web browser. It is included automatically during the View Connection Server installation, and is supported on Linux and Mac OS/X.

View Administrator

The component View Administrator provides the possibility to administrate the system through a Web browser. It is possible to make configuration settings, and to manage virtual desktops and entitlements of desktops of Windows users and groups. It is also possible to monitor log events.

View Composer

View Composer is used by View to create and deploy cloned desktops from VirtualCenter. It enables View administrators to clone and deploy multiple desktops from a single centralized base image (sometimes called "parent virtual machine"), creating replicas, each with their own unique identifier.

In the infrastructure used in the ARPEGGIO project, View Composer was installed on a Virtual machine created with the client vSphere, where the operating system Window Server 2008 R2 (64-bit) had been installed.

View User Authentication

One thing of fundamental importance in these systems is security, especially to make sure that only authorized people access. There are ways to guarantee this, with the use of credentials, but View also provides an additional tool called *View User Authentication*, that uses RSA SecurID authentication, requiring the use of a SecurID token for each user.

So, each View Connection Server must be joined to an Active Directory domain, in order to allow user authentication.

When users enter their RSA SecurID credentials, View Connection Server communicates with RSA Authentication Manager to verify the information. After the verification of the credentials, View Connection Server requests Active Directory domain credentials from the user and communicates with Active Directory to continue the authentication process

For the infrastructure of ARPEGGIO, the role of Active Directory was not performed by a physical machine, but a virtual machine, managed with the virtual host ESXi.

To do this, an ad hoc virtual machine was created, with operating system Windows Server 2008 R2, 64-bit.

2.3 Infrastructure for measurements

The IT architecture was implemented in the Lucense Data Center, that provided its processing and networking resources for a collaboration with the University of Pisa.

The infrastructure consisted of the following hardware and software components:

- Hypervisor VMware ESXi 5.5
- VMware vCenter 5.5 (initially installed on Windows 2008R2, then on Linux as appliance)
- Horizon View 6.0: Connection Broker, VComposer, Security Server, View Client, HTML Access (WEB version of the View Client)
- Windows 2008R2 as server Active Directory

For the management of the environment, these machines have been installed:

- Active Directory Win2008R2
- Server Composer (Win2008R2)
- Appliance vCenter (Linux)
- View Connection Broker (Win2008R2)
- Security Server connected with View Connection Broker

The clients used for testing were:

- PCoIP View Client Windows/Linux/Android/IOS
- Proprietary device AXEL3000/M80F

Once the infrastructure was created, the last thing to do was creating the virtual machines to use in the pool.

Given the use in an enterprise environment, a VM with Windows 8 (64-bit) was the one used as matrix for all the rest of Virtual Machines to be created.

There are different ways to create the pool of desktops to be used by the system:

- Minimum: a minimum number of desktop is created at the beginning
- Maximum: a maximum number of desktop is set, in order to limitate them
- Available: there are created as many desktop as requested

The pool was created with View Manager, and the "Available" mode was the chosen one, so that it is the vCenter Server adapting the number of VMs in the pool depending on the requests.

2.4 Measurement and traffic characterization

Our main role in the project ARPEGGIO was to perform measurements on VDI traffic from a general purpose client, connected to Internet. We used two machines and performed from each the same sets of measurements and test: the first was a machine connected to the Internet with Ethernet and with public IP, portaying a typical connection from a company. The second was a laptop connected in wireless with a 7Mb/s ADSL line, portaying a situation that is more likely to experience in home or small business environment.

The measurements show that, except for video streaming, the characteristics of traffic were similar in the two scenarios.

In both cases, the machines were equipped with Windows operating system, and access to the remote desktop was realized using the software VMware Horizon View Client [29].

The user would have to use such software, connect to the service, enter credentials, and then there were two possible options:

- connecting to the *VDI VMs external pool*, which offers a whole virtual machine, with various application installed
- using directly one of the application made available (CD burner, LibreOffice Writer, LibreOffice Calc, etc.)

Each service was used for a while and from the two different machines, while the traffic capture software Wireshark was used to capture the traffic, in order to then perform analysis.

2.4.1 Traffic graphs

Figures 2.2-2.19 show the graphs of traffics. For all the graphs: black line is the download traffic (from the VM to the user), and in red the upload (from the user to the VM).

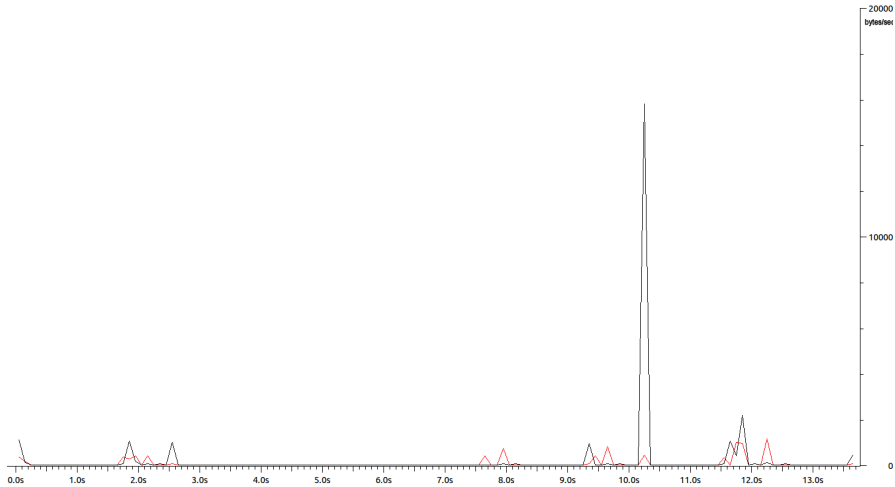


Figure 2.2: Initialization of connection - Ethernet

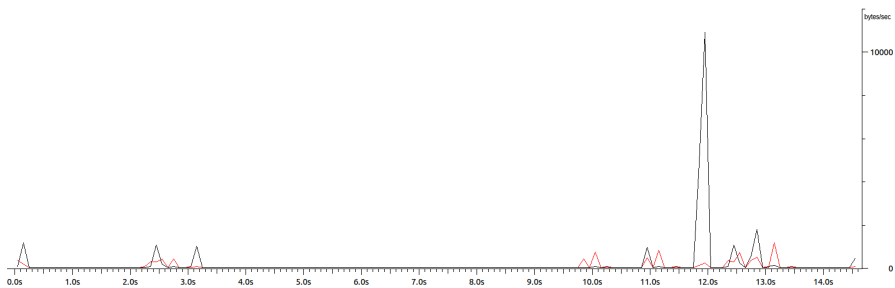


Figure 2.3: Initialization of connection - Wi-Fi, ADSL

The first thing we notice is the peak of traffic at the initialization of the session, that is due to the establishment of the connection and authentication of the user.

The shape of the graphs for 2.2 and 2.3 is similar in the two scenarios, although in the case of Wi-Fi ADSL connection, a little more time is necessary in order to complete the operation.

For further details, in figure 2.4 we show part of the flow graph, that is the sequence of packets really exchanged between our device and Lucenseis server.

Once authenticated in the system, we were able to use the applications available.

CDburner is an application that allows to burn CDs or DVDs. Figure 2.5 and 2.6 portray, respectively, the traffic measured in the scenario of Ethernet connection from Public IP, and the scenario of wireless ADSL connection.

The spikes of traffic in download coincide with moments in which the user performs actions such as selecting an option (copy disc, create data disk, burn DVD, etc), that

Time	VDI_client	VDI_server	
1.745	51374 > https [SYN]		TCP: 51374 > https [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
1.758	(51374) <----->	(443)	
1.758	https > 51374 [SYN,		TCP: https > 51374 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=2 SACK_PERM=1
1.758	(51374) <----->	(443)	
1.758	51374 > https [ACK]		TCP: 51374 > https [ACK] Seq=1 Ack=1 Win=65700 Len=0
1.777	(51374) <----->	(443)	
1.777	Client Hello		TLV1.1: Client Hello
1.822	(51374) <----->	(443)	
1.822	Server Hello, Certf		TLV1.1: Server Hello, Certificate, Server Key Exchange, Server Hello Done
1.878	(51374) <----->	(443)	
1.878	Client Key Exchange		TLV1.1: Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
1.945	(51374) <----->	(443)	
1.945	Change Cipher Spec		TLV1.1: Change Cipher Spec
1.945	(51374) <----->	(443)	
1.945	Encrypted Handshake		TLV1.1: Encrypted Handshake Message
1.945	(51374) <----->	(443)	
1.945	51374 > https [ACK]		TCP: 51374 > https [ACK] Seq=409 Ack=1077 Win=64624 Len=0
1.946	(51374) <----->	(443)	
1.946	Application Data		TLV1.1: Application Data
2.162	(51374) <----->	(443)	
2.162	https > 51374 [ACK]		TCP: https > 51374 [ACK] Seq=1077 Ack=718 Win=64818 Len=0
2.162	(51374) <----->	(443)	
2.162	Application Data		TLV1.1: Application Data
2.379	(51374) <----->	(443)	
2.379	https > 51374 [ACK]		TCP: https > 51374 [ACK] Seq=1077 Ack=1075 Win=64462 Len=0
2.590	(51374) <----->	(443)	
2.590	Application Data		TLV1.1: Application Data
2.590	(51374) <----->	(443)	
2.590	Application Data		TLV1.1: Application Data
2.590	(51374) <----->	(443)	
2.590	51374 > https [ACK]		TCP: 51374 > https [ACK] Seq=1075 Ack=1983 Win=65700 Len=0
2.590	(51374) <----->	(443)	
2.590	Application Data		TLV1.1: Application Data
7.697	(51374) <----->	(443)	
7.697	https > 51374 [ACK]		TCP: https > 51374 [ACK] Seq=1983 Ack=1432 Win=64104 Len=0
7.909	(51374) <----->	(443)	
7.909	Application Data		TLV1.1: Application Data
8.113	(51374) <----->	(443)	
8.113	https > 51374 [ACK]		TCP: https > 51374 [ACK] Seq=1983 Ack=2125 Win=65536 Len=0
9.397	(51374) <----->	(443)	
9.397	Application Data		TLV1.1: Application Data
9.398	(51374) <----->	(443)	
9.398	Application Data		TLV1.1: Application Data
9.398	(51374) <----->	(443)	
9.398	51374 > https [ACK]		TCP: 51374 > https [ACK] Seq=2125 Ack=2825 Win=64856 Len=0

Figure 2.4: Flow graph: messages exchanged when opening a new session

cause new menus to open, thus causing changes on the screen. Let us not forget that the main source of traffics in VDI applications, is the refreshing of the screen itself, which must be fast enough for the user to not perceive delays in usage. These cause many sudden spikes, making the traffic rate extremely variable. Such spikes are usually in the order of 50KB/s.

Smaller spikes we see in the graphs are due to many actions that cause minor or slower changes on the screen, like dragging and dropping files to be copied on CD.

Among the various measurements, the case of the playout of a video flow was particularly interesting.

This was performed using a Virtual Machine, that is a functionality provided by the application *VDI VMs External pool*: this gives to the user a whole VM, of which the user can see the desktop from remote.

Here we can see some differences between the scenario with Ethernet connection and the one with Wi-Fi ADSL, probably due to limitation in bandwidth of ADSL itself: in this second scenario, traffic is limited around 900KB/s, while with Ethernet connection the traffic can reach much higher values, even though with higher irregularities and variations (probably due also to video buffering).

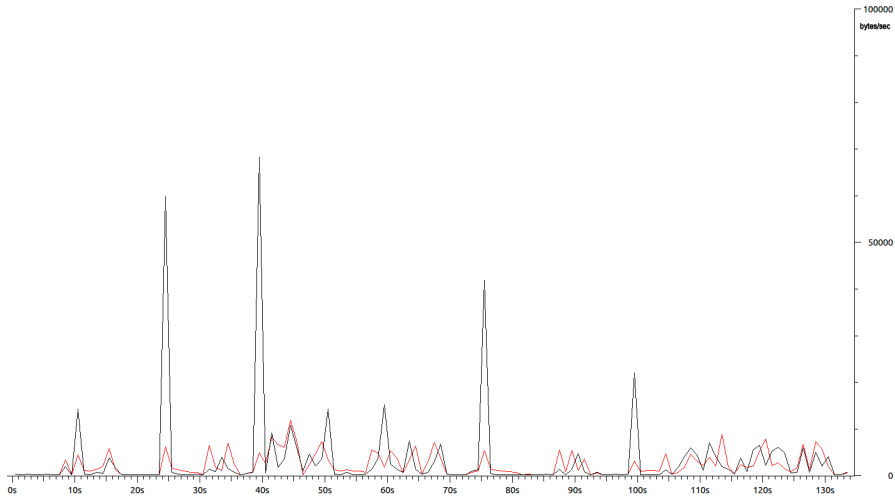


Figure 2.5: Traffic graphs for CDburner - Ethernet

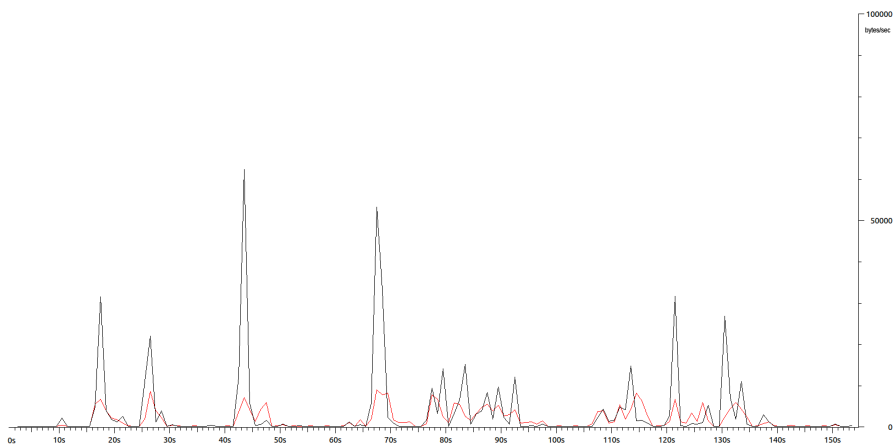


Figure 2.6: Traffic graphs for CDburner - Wi-Fi ADSL

In both cases the video was seen on full screen in the beginning, and shrunk to regular size for the last 15 seconds of visualization, which led to a decrease in the traffic rate in case of Ethernet, while it led to no changes with ADSL.

The spikes of traffic before the start of the video, are due to actions that provoke important instantaneous variations on screen, like opening a directory, moving a window. These actions provoke peaks in the order of 200KB/s.

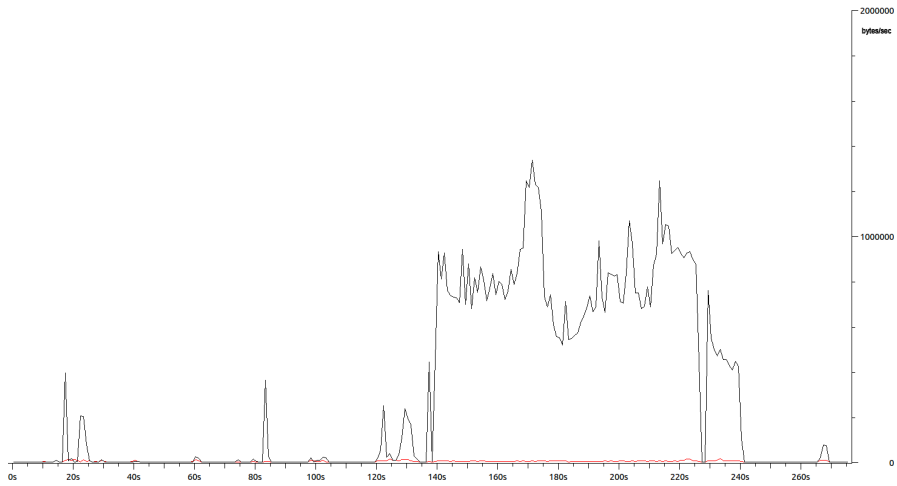


Figure 2.7: Traffic for videostreaming - Ethernet

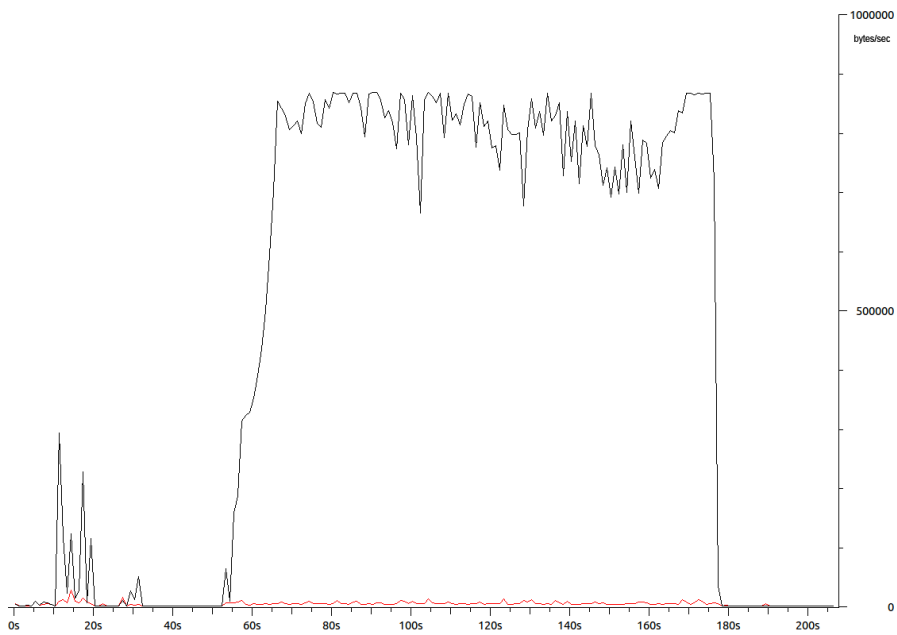


Figure 2.8: Traffic for videostreaming - Wi-Fi ADSL

The VDi system we are using also makes available some LibreOffice applications (Draw, Calc, Math, Impress, Writer). In the next paragraphs we show the traffic measurements we obtained with them.

2.4. MEASUREMENT AND TRAFFIC CHARACTERIZATION

Figures 2.9 and 2.10 show the traffic measured while using LibreOffice Draw.

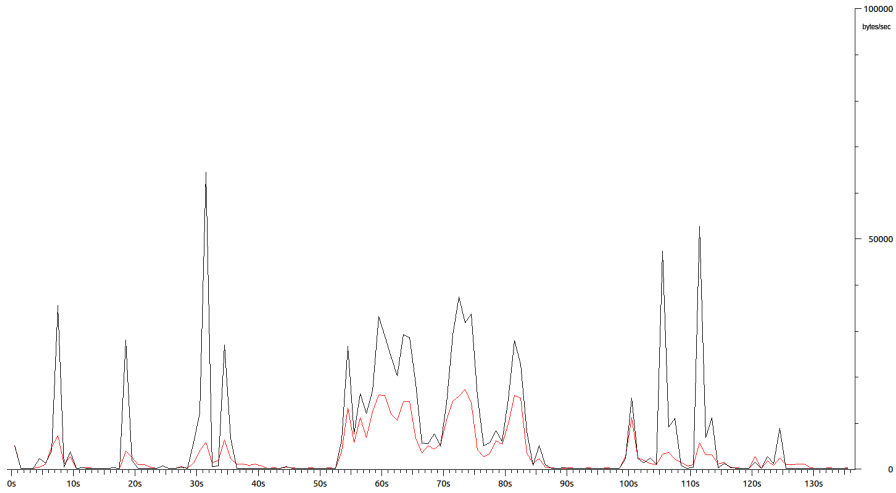


Figure 2.9: Traffic graphs for LibreOffice Draw - Ethernet

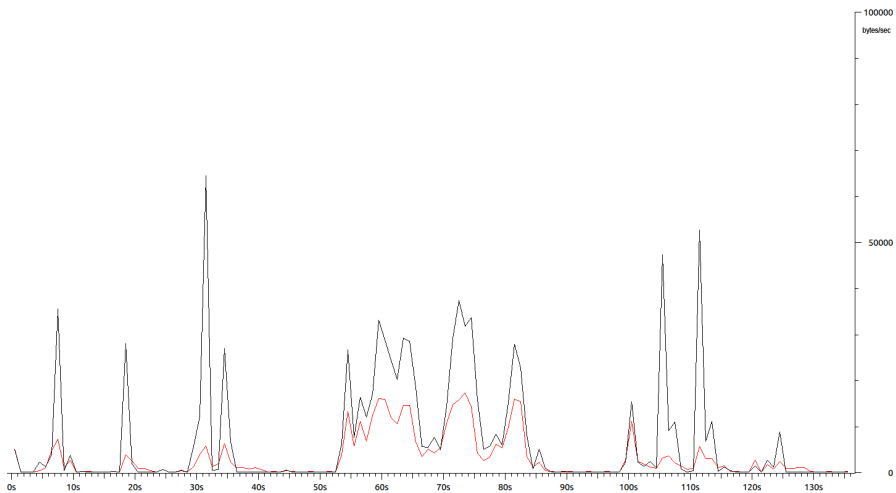


Figure 2.10: Traffic graphs for LibreOffice Draw - Wi-Fi ADSL

The capture starts right after authentication. Ten seconds after starting the capture, we launched the application LibreOffice Draw, and once opened we started drawing lines.

Traffic is very variable, due to the fact that in some moments there were sudden variations of the images shown on screen. At second 70, we stopped to save the file (the two traffic spikes are due, respectively, to opening the menu from File tab in order to select the Save command, and to the appearing of the window that browses where to save the file).

These same actions have been performed in the case of ADSL connection. The values and shapes of traffics are similar, and traffic peaks at 60KB/s, while when there are no actions performed, the amount of traffic is negligible.

Figure 2.11 shows the traffic captured with Ethernet connection while using LibreOffice Calc.

5 seconds after the start of the capture, we launched the application, and after 30 seconds we started to use it. The spikes in the upload traffic (in red) that are already present in the meanwhile are due to actions such as moving the position of the mouse.

While using the application, around the seconds 90-100, we can see again many spikes of download traffic, due to opening some of the tab menus, that cause the transmission of bursts of packets, since whole area of the screen have to change fast.

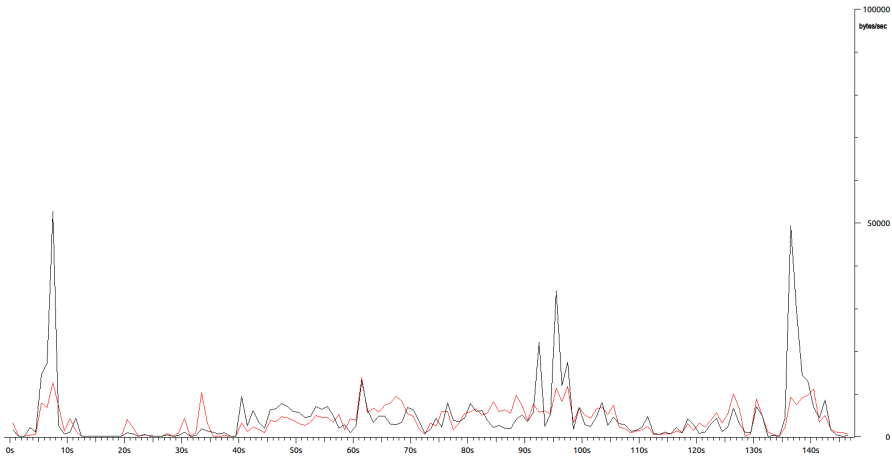


Figure 2.11: Traffic graphs for LibreOffice Calc - Ethernet

With the ADSL connection (figure 2.12), the values of traffic remain substantially similar, and also similar to what we saw for LibreOffice Draw, around 60KB/s.

Figures 2.13 show the captured traffic while using LibreOffice Math with the machine connected via Ethernet. We can easily spot the differences between moments in which the application is being used (thus traffic is being generated) and instants (seconds 85-105) where no action is performed. VDI traffic is often characterized by sudden abrupt variations.

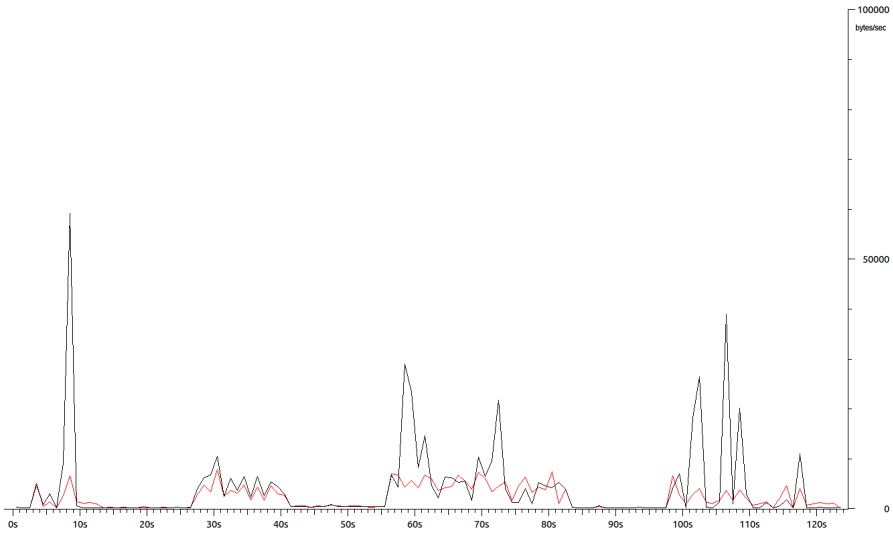


Figure 2.12: Traffic graphs for LibreOffice Calc - Wi-Fi ADSL

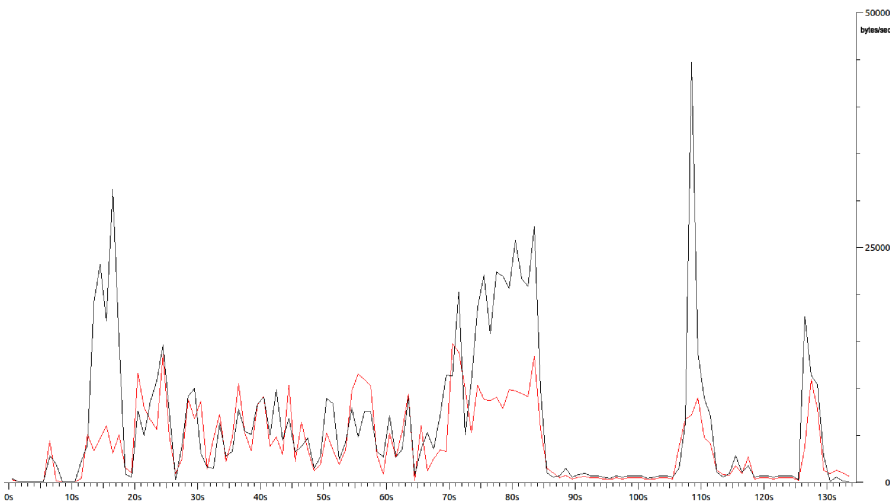


Figure 2.13: Traffic graphs for LibreOffice Math - Ethernet

In the case of ADSL connection (figure 2.14) again we had a period of inactivity (seconds 55-80), and that reflects on almost complete absence of traffic in those moments. The spikes at the second 85 are due to saving the file, while the final spike is the closing of the application, which takes us back to the homepage of VMware Horizon View Client, causing a big change on screen.

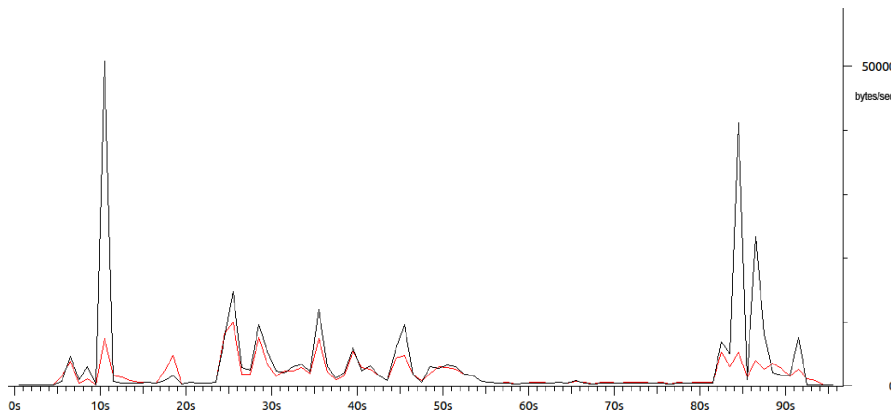


Figure 2.14: Traffic graphs for LibreOffice Math - Wi-Fi ADSL

LibreOffice Impress has a more graphic nature, which means that, as shown in figures 2.15 and 2.16, the traffic spikes are bigger than what we have seen with other applications of the LibreOffice pack. The traffic spike at the second 80 in figure 2.15, for example, is due to inserting a FontWork writing, while that at 100-120 is due to changing layout. The upload and download traffic between 130 and 140, was caused by moving the mouse (which, obviously, implies that packets are both sent and received).

Similar actions have then been performed when connected with ADSL (figure 2.16).

Last, we used LibreOffice Writer, and figures 2.17 and 2.18 show the captured traffic. We can see that writing continuously generates a stable traffic, while spikes are due to actions such as opening the application, inserting a hyperlink in the text (since it causes a new window to open), saving the file (which causes two consecutive spikes, one for the opening of the menu to save, and the one for the window to save the file), closing the application (which again causes big changes of the context displayed on the screen).

The last figure (2.19) shows, finally, the traffic that is generated when we close VMware Horizon View Client.

In figure 2.20 is the flow graph associated with ending the connection of VMware Horizon View Client.

2.4.2 Packet length distribution

With the traces we collected, we proceeded to perform additional analysis on the VDI traffic.

Particularly important are the statistics on the packet length distribution, since they give a better image of the communication, and can provide hints to improve the performances of the system.

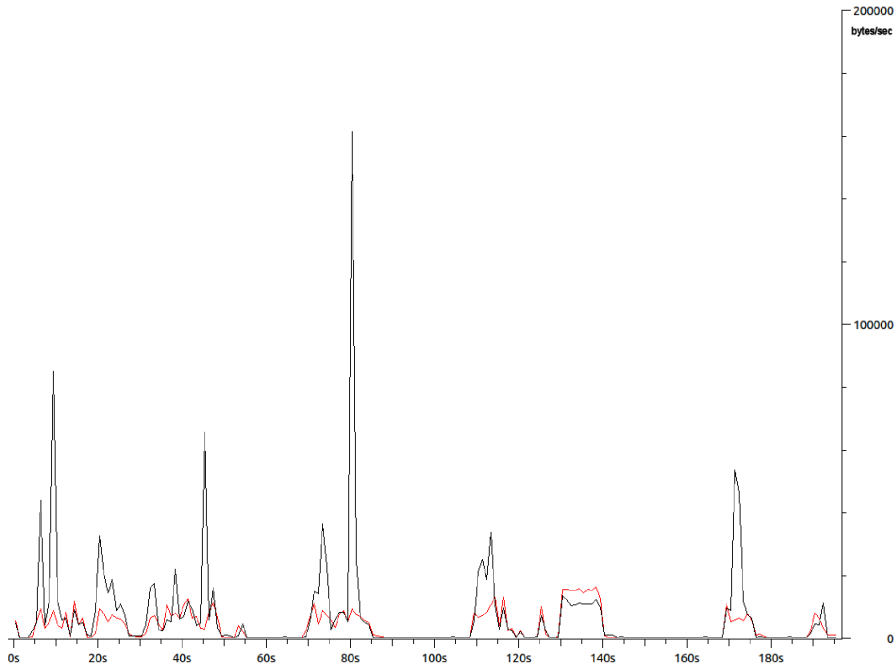


Figure 2.15: Traffic graphs for LibreOffice Impress - Ethernet

In the following figures, we show the graphs we obtained for the various applications, separating upload traffics (from the user to the data center) and download traffics (from the data center to the user).

On the x axis we have the length (in byte) of the packets. On the y axis, the occurrence.

In the first two figures (2.21) we can see the packet length distribution recorded for the initialization of the connection (that is: launching Horizon VM Client and authenticating to the server).

One of the applications made available by the system is the CD burner application, to burn CD or DVD. Figure 2.22 shows the packet length distribution for download traffic and upload traffic, respectively.

Some of the applications belong to the LibreOffice package (Calc, Draw, Math, Impress, Write) and the relative packet length distributions are shown in figures 2.23-2.27.

We can easily see how the upload traffic consists of smaller packets, as they just contain information about mouse position or keyboard inputs. The download traffic instead consists of images to display, thus packets are longer.

For all applications belonging to LibreOffice, we can see the characteristics of the traffic are similar.

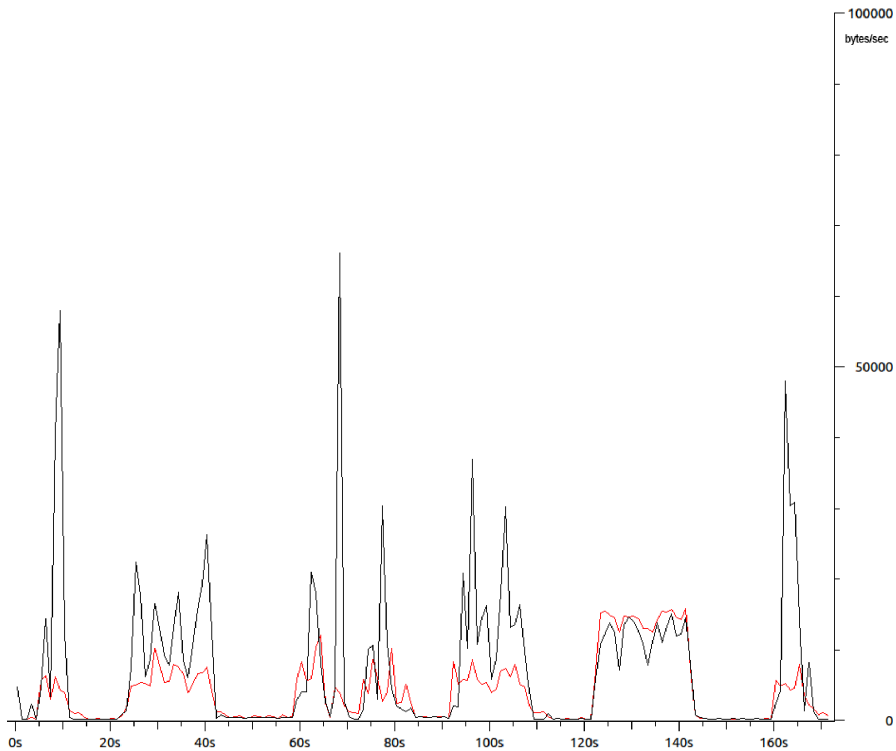


Figure 2.16: Traffic graphs for LibreOffice Impress - Wi-Fi ADSL

Traffic generated by video applications (for example, video streaming) tends to be CBR (constant bitrate), as once the video codification is set, the rate of transmission and the characteristics of packets will not change, but will be stable.

In figure 2.28 we have a clear representation of this: we can clearly see how in the download traffic we have a massive presence of packets with same characteristics (packets bringing video information).

As for the upload traffic, instead, that is similar to that of other applications, as it again depends mostly on mouse and keyboard inputs.

2.4. MEASUREMENT AND TRAFFIC CHARACTERIZATION

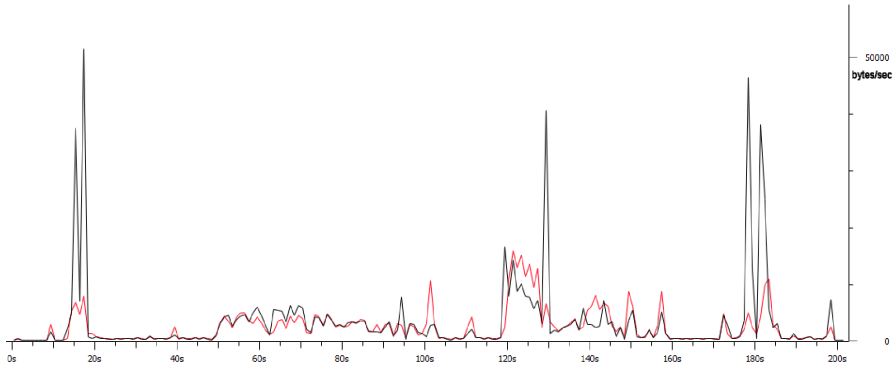


Figure 2.17: Traffic graphs for LibreOffice Write - Ethernet

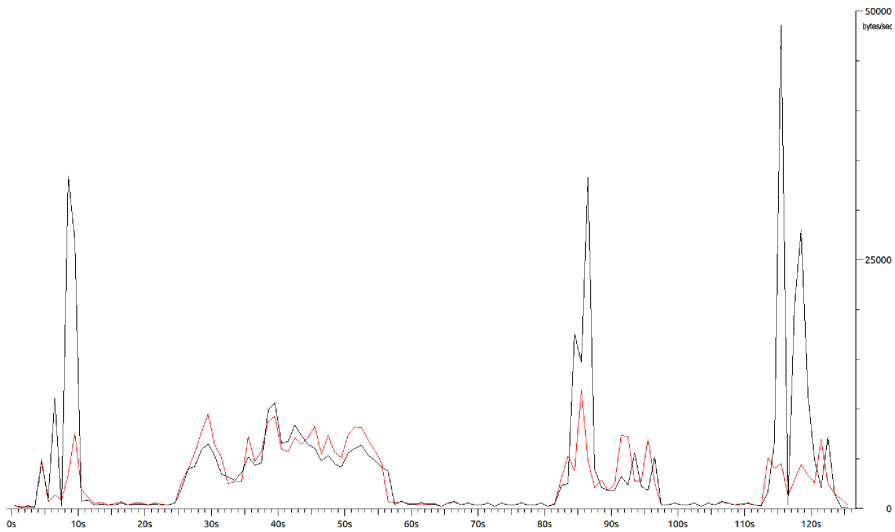


Figure 2.18: Traffic graphs for LibreOffice Write - Wi-Fi ADSL

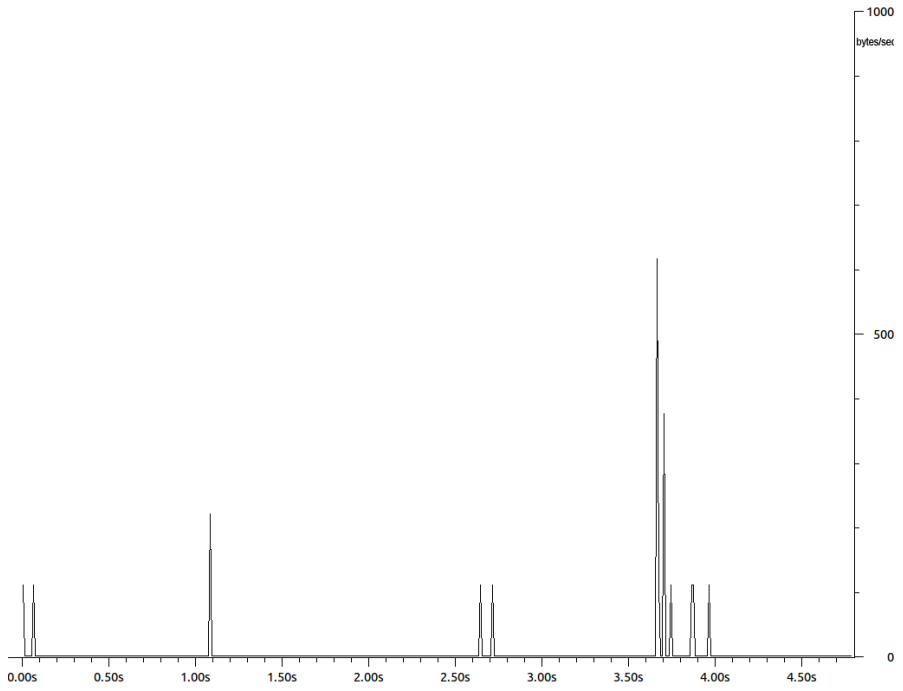


Figure 2.19: Traffic when closing connection

2.4. MEASUREMENT AND TRAFFIC CHARACTERIZATION

Time	VDI_client	VDI_server	
0.000	(50002) Source port: 50002	-----> (4172)	UDP: Source port: 50002 Destination port: pcoip
0.066	(50002) Source port: pcoip	-----> (4172)	UDP: Source port: pcoip Destination port: 50002
1.081	(50002) Source port: pcoip	-----> (4172)	UDP: Source port: pcoip Destination port: 50002
1.082	(50002) Source port: 50002	-----> (4172)	UDP: Source port: 50002 Destination port: pcoip
2.649	(50002) Source port: 50002	-----> (4172)	UDP: Source port: 50002 Destination port: pcoip
2.718	(50002) Source port: pcoip	-----> (4172)	UDP: Source port: pcoip Destination port: 50002
3.664	(50002) Application Data	-----> (4172)	TLSv1.1: Application Data
3.664	(49608) Application Data	-----> (443)	TLSv1.1: Application Data
3.666	(49608) Encrypted Alert	-----> (443)	TLSv1.1: Encrypted Alert
3.666	(50116) 50116 > https [RST,	-----> (443)	TCP: 50116 > https [RST, ACK] Seq=54 Ack=1 Win=0 Len=0
3.666	(50116) Encrypted Alert	-----> (443)	TLSv1.1: Encrypted Alert
3.666	(50117) 50117 > https [RST,	-----> (443)	TCP: 50117 > https [RST, ACK] Seq=54 Ack=1 Win=0 Len=0
3.670	(50002) Source port: 50002	-----> (4172)	UDP: Source port: 50002 Destination port: pcoip
3.703	(49608) https > 49608 [ACK]	-----> (443)	TCP: https > 49608 [ACK] Seq=1 Ack=187 Win=32441 Len=0
3.706	(49608) Application Data	-----> (443)	TLSv1.1: Application Data
3.706	(49608) Encrypted Alert	-----> (443)	TLSv1.1: Encrypted Alert
3.706	(49608) https > 49608 [FIN,	-----> (443)	TCP: https > 49608 [FIN, ACK] Seq=107 Ack=187 Win=32441 Len=0
3.706	(49608) 49608 > https [ACK]	-----> (443)	TCP: 49608 > https [ACK] Seq=187 Ack=108 Win=4329 Len=0
3.748	(50002) Source port: pcoip	-----> (4172)	UDP: Source port: pcoip Destination port: 50002
3.860	(50002) Source port: pcoip	-----> (4172)	UDP: Source port: pcoip Destination port: 50002
3.876	(50002) Source port: pcoip	-----> (4172)	UDP: Source port: pcoip Destination port: 50002
3.966	(50002) Source port: pcoip	-----> (4172)	UDP: Source port: pcoip Destination port: 50002
4.846	(49608) 49608 > https [RST,	-----> (443)	TCP: 49608 > https [RST, ACK] Seq=187 Ack=108 Win=0 Len=0

Figure 2.20: Flow graph: messages exchanged during closing of connection

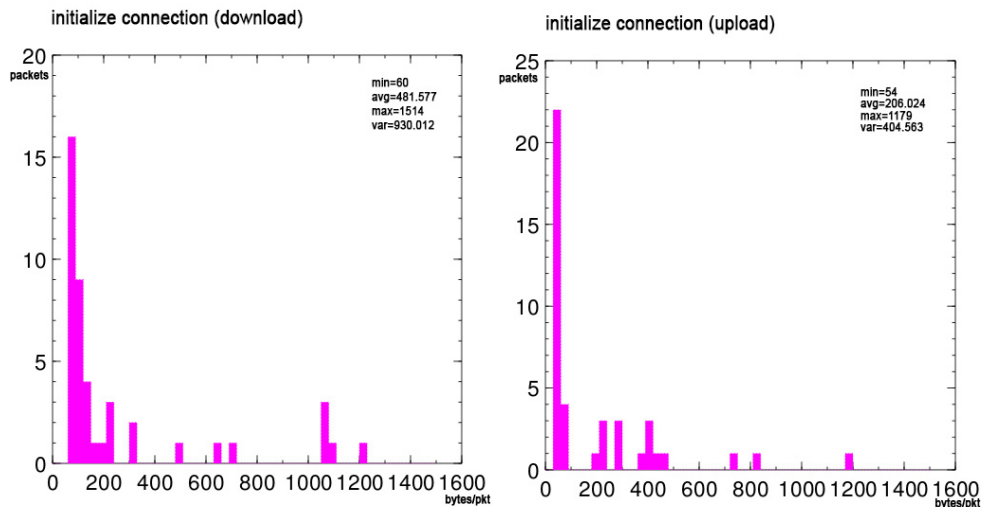


Figure 2.21: Packet length distribution for connection initialization

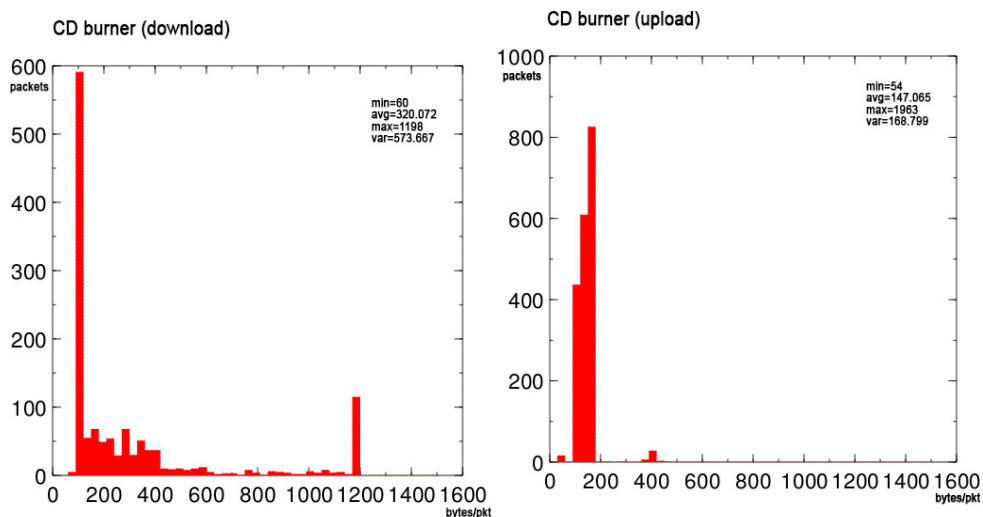


Figure 2.22: Packet length distribution for CD burner application

2.4. MEASUREMENT AND TRAFFIC CHARACTERIZATION

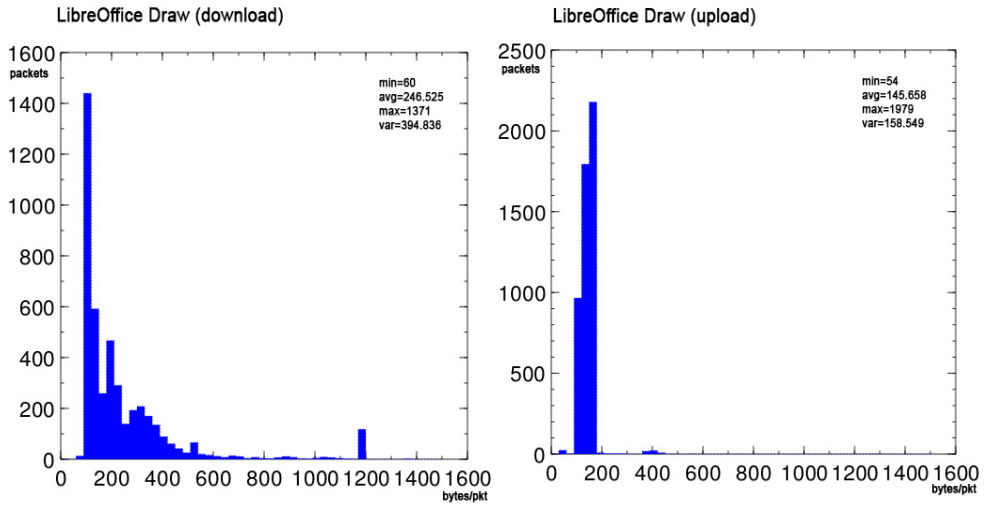


Figure 2.23: Packet length distribution for LibreOffice Draw

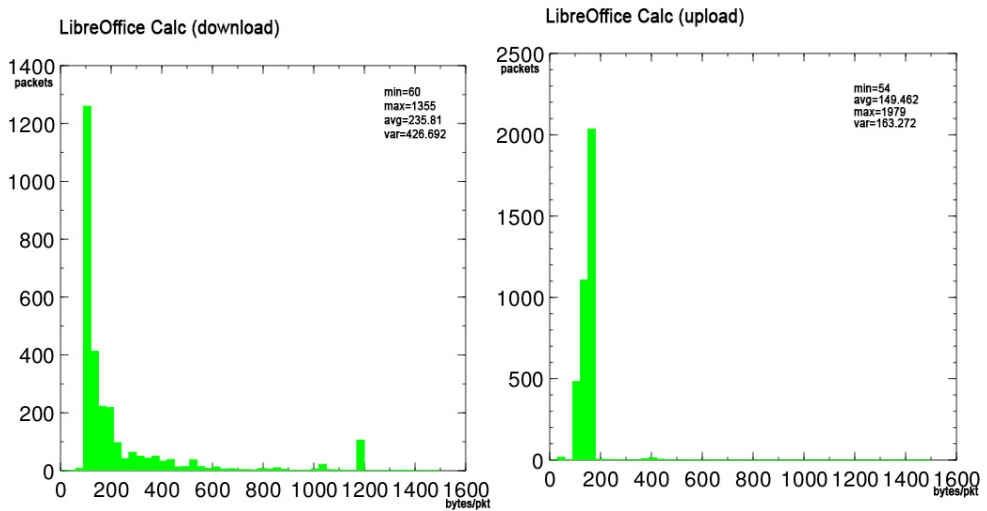


Figure 2.24: Packet length distribution for LibreOffice Calc

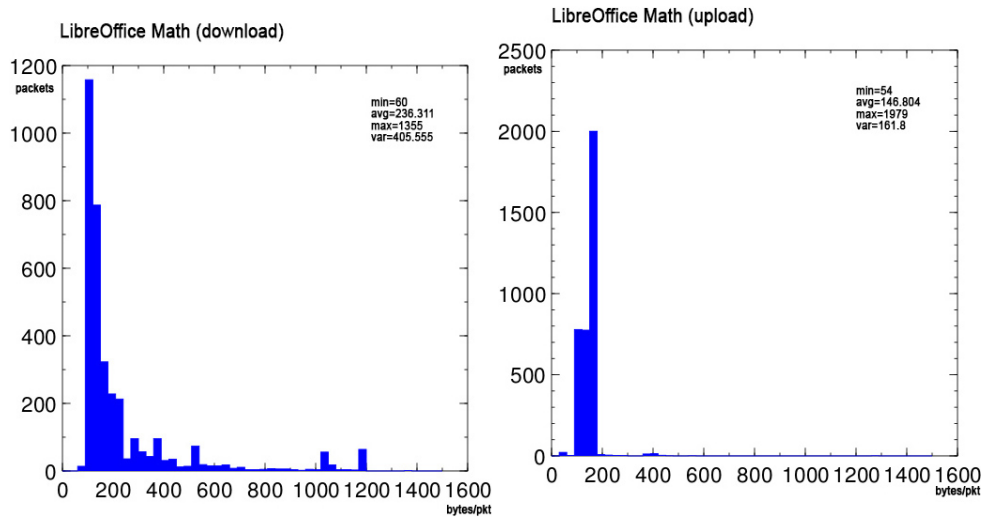


Figure 2.25: Packet length distribution for LibreOffice Math

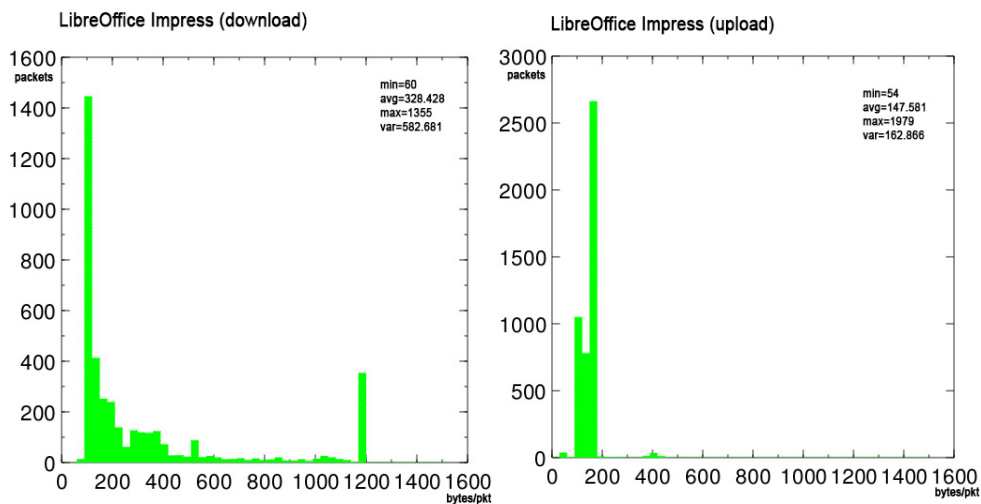


Figure 2.26: Packet length distribution for LibreOffice Impress

2.4. MEASUREMENT AND TRAFFIC CHARACTERIZATION

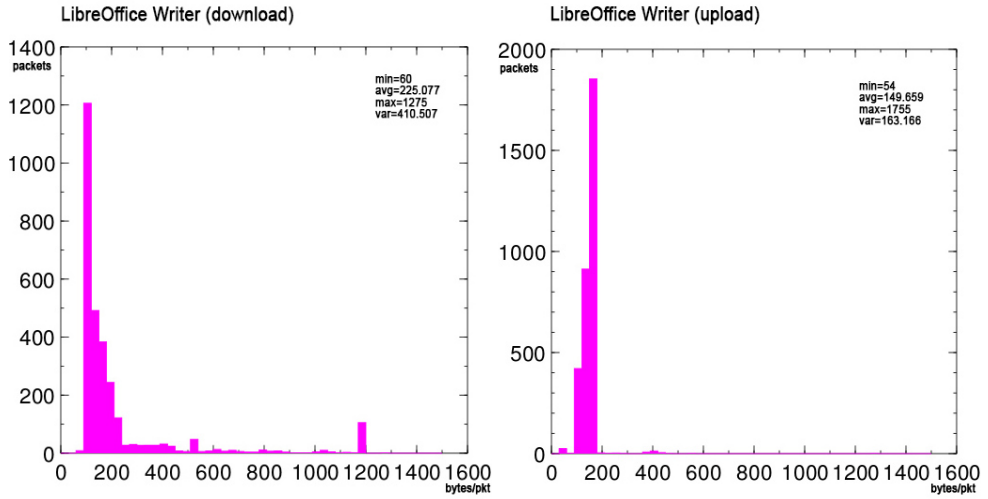


Figure 2.27: Packet length distribution for LibreOffice Write

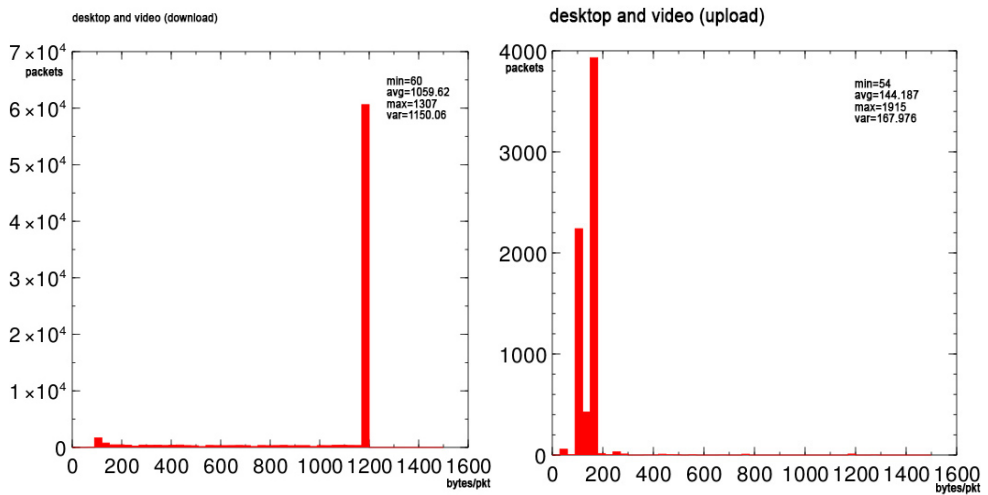


Figure 2.28: Packet length distribution for video application

Software Defined Networking for resource allocation and monitoring

This chapter discusses solutions for Cloud Data Centers, based on Software Defined Networking.

Software-defined networking (SDN) is a novel approach to networking that aims at providing an abstraction of higher-level functionalities of the network, in order to enable better management.

The core idea is to split the network in two separate planes: on one side there is the part that makes decisions about forwarding (control plane). Decisions are taken by a central element (controller) that has a global view of the whole network and enforces rules for the switches. Switches do not take decisions: if they see a packet belonging to a flow for which they do not have rules, they just forward it to the controller.

This control plane works above an underlying system (data plane) that just forwards regular traffic to the selected destinations.

Our solutions will leverage this approach in order to have a higher degree of control of the network, while harmonizing the usage of resources, creating an orchestration between the computational resources and the network resources deployed in a generic data center environment.

Cloud Data Centers are becoming more and more strategic nowadays, and great efforts are put both in research and in industry to increase efficiency and performances of these systems, aiming at optimizing VM placement across servers in a DC.

There are both commercial tools (VMware Capacity Planner [31], IBM WebSphere [32]) and research initiatives ([33][34][35]) focusing on new solutions for resource management in virtualized environments. A main issue with most solution though is they do not fully consider all the elements that are present in a DC and concur to affect overall service performance, such as the state of the network. In order to better manage the risk of congestions or traffic latency [36] derived from the oversubscription on one side, and in order to make sure that all resources deployed are consistently used on the other side, we need new approaches in Cloud DC networks that also consider the network as a resource to be taken into account during the VM placement.

Our solution, named "SDN-DC orchestrator" is meant to dynamically coordinate the instantiation and management of Virtual Machines across servers and, accordingly, establish a delivery path throughout switches.

3.1 SDN-DC Orchestrator

The general structure of a Data Center is portrayed in figure 3.1.

Basically, we have a set of computational resources, and a network connecting them.

We can imagine that different users from outside the system will want to access the servers, and all these user requests are a challenge for the system, as each user will need both computational resources, and network resources as well.

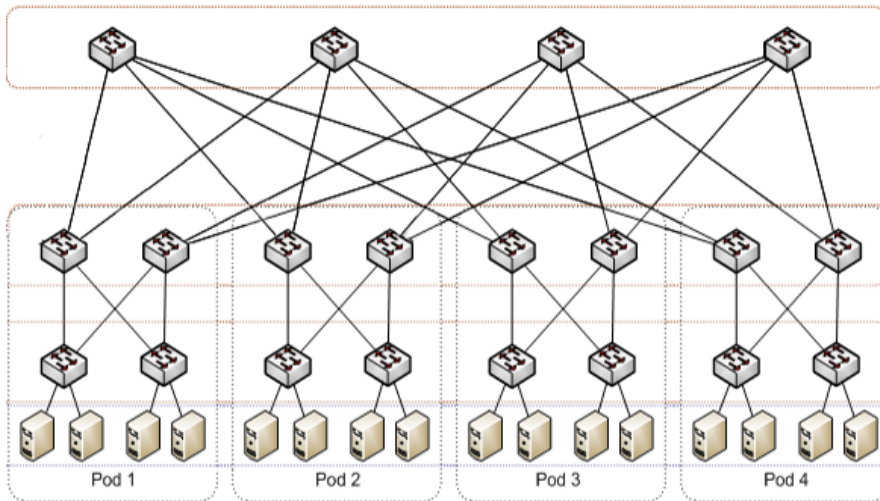


Figure 3.1: DC resource infrastructure

There are many challenges in managing and allocating resources in such a system, and different solution can be proposed.

In our concept, we consider the system as consisting of two pools of resources, the computational resources (servers), and the network elements (switches and links). And we consider each request from the users to be basically a request for allocation of resources from both pools. These two aspect have to be considered together, as considering only one or the other may lead to the problems we see in figure 3.2: some servers may still have available computational power, while the switches that lead to them may not be able to serve any new traffic.

In the next sections, our architecture is explained and described, and the experimental results are shown.

3.2 Architecture

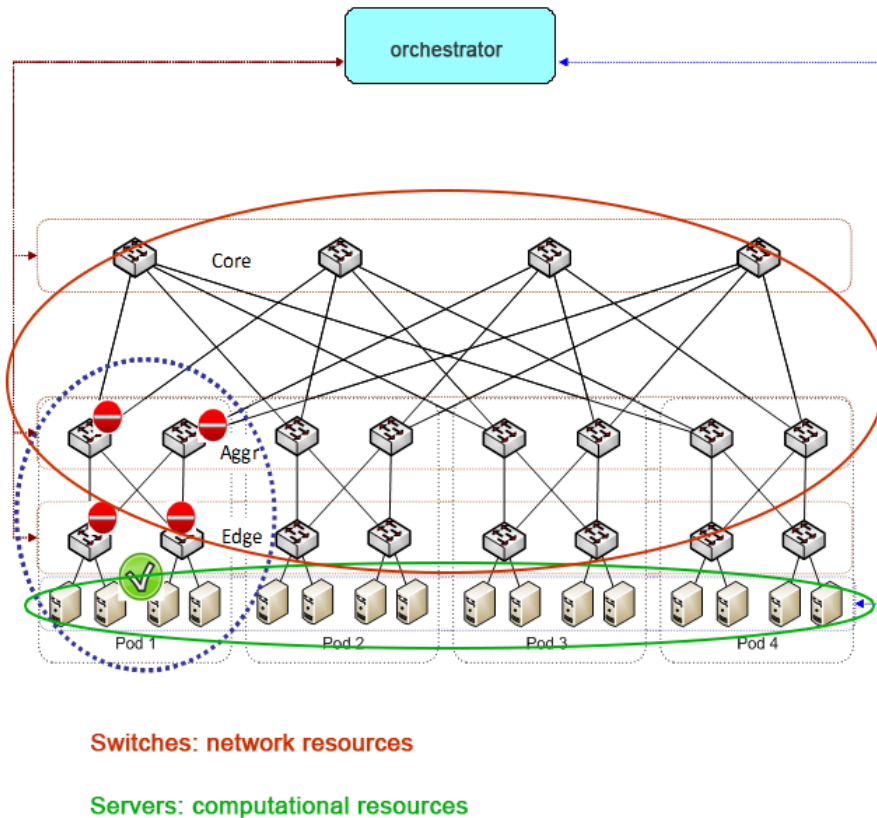


Figure 3.2: Orchestration for network and computing resources

The SDN-DC orchestrator consists of operational components that interact realizing the orchestration process.

In order to reach our goal, we need on one side to have a constantly updated picture of the system, keeping track of the allocated resources, and measuring the traffic that is actually passing in the system, and how and if that is changing. On the other side, we need to integrate and process all the data we get, and use them in algorithms that give us the most efficient results.

The next paragraphs will describe the main functionalities related to mechanisms for monitoring and for load estimation, and the algorithms for the coordinated selection of computing and network resources.

3.2.1 SDN-DC Orchestrator Design

First of all, let us give a big picture of how our theoretical system is made and works: there is a POX OpenFlow controller, that allows to manage the network according to SDN principles.

The network consists of switches divided in three hierarchical levels: core, aggregation, edge.

The edge switches are the ones connected to the servers, that have the computational resources that the users want to use.

The core switches are connected to a gateway that connects the Data Center with the outside the world.

Using a web-based GUI, end-users ask for VMs with specific configuration settings (CPU, RAM, disk, bandwidth, expiration date).

The Web Server waits for VM requests, and interfaces with the VM Request Handler which, parsing incoming requests, retrieves VM configuration settings.

The orchestrator, the heart of the system, consists of many modules, as shown in figure 3.3, each performing specific operations.

The data collected by the VM Request Handler are used by the Resource Selection and Composition Engine (RSCE) to decide if there are enough resources to accept the new request. If so, the RSCE is able to find a server to host the VM, and a network path from the server to the DC gateway.

The selection and composition of computing and network resources is performed by the RSCE using one of the algorithms described in paragraph 3.2.4. Such algorithms need input data that is retrieved from the Network Topology Database (NTDB), Network Statistics Database (NSDB) and Server Statistics Database (SSDB).

NTDB keeps up-to-date information about the DC topology, including network links, switches and servers. Such information is provided by the Topology Discovery service, that relies on functions made available by OpenFlow controller POX.

NSDB is populated by the Network Statistics Handler and keeps up-to-date information about the measurements and estimation of traffic load on each network switch/link, gathered from OF statistics.

SSDB is populated by the Virtual Machine Manager (VMM) Handler and contains information about the current availability of computational (CPU, RAM) and storage (disk) resources for each server as result of previous VM allocations.

When a new VM allocation request is accepted, the Virtual Machine Manager (VMM) creates the new Virtual Machine on the server selected by the algorithms, and once the Virtual Machine has received its IP address (from the DHCP server), appropriate forwarding rules are enforced by the controller on all the switches along the selected

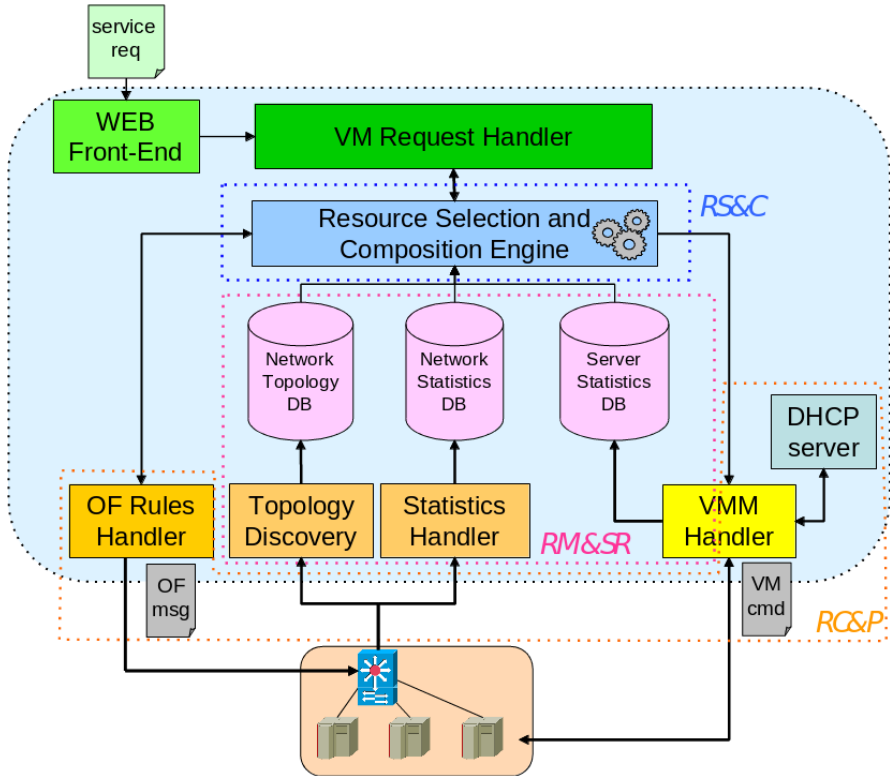


Figure 3.3: SDN-DC Orchestrator: system design and orchestration components

forwarding path (each server has its own IP, and each VM will receive an own unique one upon being created).

In figure 3.3, the three main conceptual components of the orchestration system are highlighted.

The *resource selection and composition* (RS&C) function performed by the RSCE relying on resource status stored in the three aforementioned databases (NTDB, NSDB, SSDB).

The *resource configuration and provisioning* (RC&P) function is provided by OF Rule Handler and VMM Handler blocks.

The *resource monitoring and status registration* (RM&SR) function is provided by the Topology Discovery, Statistics Handler and VMM Handler blocks, that will store their data in the three databases NTDB, NSDB, SSDB respectively.

3.2.2 Topology discovery

The topology discovery service provides and updates knowledge about the elements present in the system, both network elements (switches and interconnection links) and computational element (servers).

As the SDN-DC orchestrator was built on top of a POX SDN controller, the topology discovery service relies on the software components that are provided by POX: the *openflow_discovery* and the *host_tracker*.

The *openflow_discovery* is the key element to network discovery. It is in charge of sending Link Layer Discovery Protocol (LLDP) packets to all the connected switches through *packet_out* messages. When a switch receives this kind of message, it sends the LLDP packets out from all its ports. If the device that receives it is an OF switch, it will perform a flow table lookup (i.e.: will check if the packet matches one of the entries of its own flow tables). Obviously the switch will not have a flow entry for this LLDP packet, so it will send it to the controller via a *packet_in* message. When the controller receives such message, it analyzes the packet and creates a connection in its discovery table for the two switches. All the remaining switches in the network will similarly send their own *packet_in* messages to the controller, that will create a complete map of the network topology. LLDP messages are also periodically exchanged during regular activity. Moreover events are raised at the controller when links go up or down, or when new links are added or old ones are removed. All the information retrieved on switches and links is maintained in the NTDB.

The *host_tracker* keeps track of the servers in the network (where they are, how they are configured, their MAC/IP addresses). When a change occurs, the software component *host_tracker* raises an event. *host_tracker* examines *packet_in* messages, and learns MAC and IP addresses. The orchestrator periodically pings with ARP messages the servers, to see if they are still up and running.

The data collected is stored in the SSDB, that also keeps track of each server (IP and MAC addresses) and the switch it is connected to (datapath ID, port number).

3.2.3 SDN-based Network Monitoring and Load Estimation

For the purpose of orchestration, it is necessary to have some monitoring function constantly running in the background, in order to obtain updated data about the utilization of the servers and network switches/links, and in order to support decisions on the admission of VM allocation requests as well as on the selection and composition of resources. A continuous assessment of traffic load is a necessity in order to prevent concurrency issues, and avoid significant impacts not only on the acceptance rate of VM allocations, but also on the quality of user experience. Such assessment in Cloud DCs is a pretty challenging task due to the high dynamicity of traffic flows generated by VMs, featured with many different time-scale behaviors and bandwidth requirements: very different type of traffic, such as elephant flows, mice flows ([37][38]) co-exist in a same network.

OpenFlow offers tools for statistic collection, that combined with estimation mechanisms allow the *resource monitoring and status registration* component of the SDN-DC orchestrator to perform its operations.

For the collection of traffic statistics and monitoring data of the DC network through the OF protocol, the switch counters [39] keeps track of the following traffic statistics:

- per-flow received/transmitted packets
- per flow received/transmitted bytes
- per-flow duration
- per-port received/transmitted packets
- per-port received/transmitted bytes
- per-port received/transmitted errors

The *resource monitoring and status registration* component periodically collects OF traffic statistic data from switches interfaces, and then the Statistics Handler stores such data in the NSDB.

First of all, the Statistics Handler calculates the average traffic rate (data throughput), at each interface. OpenFlow switches only provide counters for transmitted (*Tx_Bytes* or received *Rx_Bytes*), that corresponds to the number of bytes that have been transmitted or received on each interface since the system started. So we must also keep previously collected data, as the instantaneous value is calculated as a difference.

If the replies to two consecutive OF statistics on the switch s requests are received at the two consecutive instant t_{i-1} and t_i , the average input and output rates for the interface connected to link l in the time interval $(t_i - t_{i-1})$ will be:

$$traffic_rate_in_i(l) = \frac{Rx_bytes(l, t_i) - Rx_bytes(l, t_{i-1})}{t_i - t_{i-1}} \quad (3.1)$$

$$traffic_rate_out_i(l) = \frac{Tx_bytes(l, t_i) - Tx_bytes(l, t_{i-1})}{t_i - t_{i-1}} \quad (3.2)$$

The *resource monitoring and status registration* component elaborates statistics to derive estimations on switch/link load for the purpose of resource selection and composition.

Due to the high variability of traffic patterns, current values of traffic rates are also highly variable over time. But in order to have a reliable estimation of the link load, it is desirable to consider trends in order to base selection decisions on more solid points.

These considerations lead to two questions:

1. How often should the orchestrator retrieve statistics from the switches? If the polling of statistic data does not take place often enough, the orchestration may make allocation decisions based on a traffic load of the network that is out-of-date. If the statistics are retrieved too often, that can create a considerable overhead for the orchestration system without guarantees of improved estimations.

2. Should the orchestrator make decisions based on the newest statistics only (i.e., instantaneous values) or should it take into account also past statistics (i.e., historical values)? How and how many historical values should be taken into account? If the orchestrator does not consider historical values of statistics at all, statistic data could be little significant, as the monitoring component might sample a moment where there is a temporary burst of packets, or a moment where the traffic has temporarily decreased: that may give a distorted image of the actual network load. On the other hand, giving too much weight to historical statistic data may prevent the orchestration system from keeping pace with the evolution of the traffic, with the risk of basing decisions on a wrong image of what is actually happening in the network, far from being a negligible risk given the high volatility and high variability of most of the traffic in Cloud DCs.

We thus defined an estimation procedure, and proceeded to implement in the SDN-DC orchestrator, for the purpose of assessing the orchestrator performance in the context of Cloud DCs.

For load estimations we use a formula where an average value is calculated using a number of historical measurements, so that the influence of past sample decreases exponentially fast.

The estimated upstream and downstream traffic loads for link l , with bandwidth B , are estimated as:

$$link_load_in_i(l) = \frac{(1 - \alpha)traffic_rate_in_i(l) + \alpha \sum_{k=i-M}^{i-1} \frac{traffic_rate_in_{i-1}(l)}{M}}{B} \quad (3.3)$$

$$link_load_out_i(l) = \frac{(1 - \alpha)traffic_rate_out_i(l) + \alpha \sum_{k=i-M}^{i-1} \frac{traffic_rate_out_{i-1}(l)}{M}}{B} \quad (3.4)$$

where α is the "history weight", a parameter that allows to assign a weight to the average of the past M samples (collected at time t_{i-1} down to t_{i-M}), against the newest one (at time t_i), with samples collected every T seconds.

The traffic rates, and consequently the estimated link loads, are calculated every time the orchestrator gets new statistics from the switches.

So, to summarize, the parameters that must be set to tune in the formulas are:

- α (*history weigh*), affecting the relevance of past events;
- M (*window size*), affecting the number of past values that are considered in the estimation;
- T (*polling time*), affecting the rate of measurements collected from switches and, ultimately, how the estimations are up to date (at the cost of overloading the system with the exchange of more packets).

3.2.4 Resource Selection and Composition Algorithms

As previously stated, in Cloud Data Centers computational power and network resources can be seen as virtual resources, provided by pools of physical resources. Upon the demand for the allocation of a new Virtual Machine, with specified CPU and bandwidth requirements, the RSCE aims at finding a server S with enough computational power to host the requested VM, and a network path P with enough available bandwidth to satisfy the user requirements.

In our model, network switches are interconnected according to the typical tree-like 3-layer DC network topology, used to connect servers. As shown in figure 3.4, the edge layer provides physical connectivity to the servers in the data centers, while the aggregation layer connects together edge layer switches. Similarly the core level of networking is used to interconnect aggregation layer switches.

Multiple redundant links connect together pairs of switches at all layers, thereby enabling high availability, even though at the risk of forwarding loops.

Different Resource Selections and Composition Algorithm (RSCA) can be adopted for the selection and composition of resources in DCs.

There can be different sets of algorithms, that rely on either optimizations through ILP formulations or on heuristics to reduce the computational burden.

In this project, a number of RSCA algorithms based on heuristics are envisioned, designed, tested for the purpose of assessing the orchestrator performance in the context of Cloud DC infrastructures.

The RSCE provides two categories of online RSCAs, diversified by the order computational and communication resources are selected [40][41]:

- Server-Driven (SD): first attempts to find a server and then the network path through-out a sequence of switches/links.
- Network-Driven (ND): first attempts to find a network path and then a server where to deploy the VM.

Using such heuristics in a DC tree-like topology, the selection of the server constrains the choice of the network path and vice-versa. A case of selection is described for each algorithm category in the DC example topology shown in figure 3.4.

On the left side, we see a Server-Driven algorithm, that first selects a server (in red), and then selects a path (in blue) that connects that server to the outside world.

On the right side, we see a Network-Driven algorithm: first the most appropriate path (in red) for the new request is selected, then one of the servers reached by that path is chosen (in blue).

But what are the "best" server and the "best" path to choose?

That greatly depends on the goal we want to achieve with resource managing. For example, we may want to keep the resource usage balanced, spreading the connections

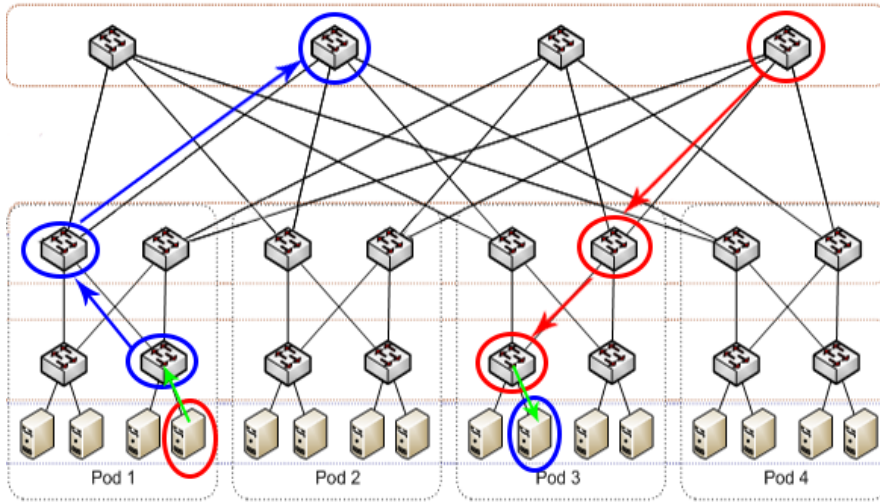


Figure 3.4: Different resource allocation strategies

as much as possible on all links, and using all servers without loading too much any of them.

Or the other way, we may want to "consolidate" resource usage, in order for example to be able to keep some of the servers and switches off when not in use, in order to save energy. This again can be done in many ways: we may implement some heuristics to use some of the resources with specific features, or we may just establish a set to use more often, offloading some of the computational load for the orchestrator.

We summarized these possible scenarios in three main heuristics to use.

- First Fit (FF) chooses the first indexed available resource (server or switch/link at each network layer) that has enough availability to meet the requirements of the request.
- Best Fit (BF) chooses the resource that is fullest, provided that it is able to accommodate the request.
- Worst Fit (WF) chooses the least loaded resource, i.e., the resource that has the most remaining availability to accommodate the request.

The chosen heuristics affects the overall usage pattern of resources, as FF and BF tend to consolidation, while WF tends to spread resource usage.

Since there are three possible heuristics, and we are working with SD and ND strategies, we can picture six possible algorithms, from the combination of these strategies: SD-FF, SD-BF, SD-WF, ND-FF, ND-BF, and ND-WF.

In all cases the available capabilities need to be obtained to decide for resource selection.

In particular, the selection of switches and links is done basing on the available bandwidth at the link and available throughput at the switch the link is connected to. At each step the following conditions must be satisfied:

1. For edge, aggregation and core links (In and Out directions):

$$\frac{link_load_i(l) + VM_B}{link_bandwidth(l)} < link_load_threshold(l) \quad (3.5)$$

2. For edge, aggregation and core switches:

$$switch_load_i(s) < switch_load_threshold(s) \quad (3.6)$$

where

$$switch_load_i(s) = \sum_l link_load_in_i(l) + \sum_l link_load_out_i(l) \quad (3.7)$$

All these parameters need to be calculated for both directions of the traffic.

The $link_load(l)$ is the estimated load at the link/switch port l . $link_load(l)$ (input and output) is calculated as explained in 3.3 and 3.4.

The $link_load_threshold(l)$ and $switch_load_threshold(s)$ are preconfigured values in the interval $[0, 1]$. Values equal to 1 mean that we allow the link to work to the theoretical full capacity. In some cases, it may be worthy to keep the value below 1 to minimize the probability of traffic congestion or quality degradation occurrences.

3.3 System performance

In order to assess the effectiveness of our solutions, we carried out various kinds of tests.

First we ran simulations, in order to be able to test highest amounts of traffic and requests on a bigger system.

We also built a testbed in our facilities, so that we could see a real behavior of the system.

3.3.1 Performance evaluation via simulations

We implemented a custom-built, event-driven Java simulator to evaluate the performance of our heuristics. The simulator reproduces one of the most common topologies, VL2.

Table 3.1 summarizes the values of some default parameters used in our simulation.

T_h is the admission threshold: when choosing the links where to allocate the traffic of a new VM, the system checks that summing the *estimated traffic load* on such link and the bandwidth requested by the new VM, the result does not exceed such threshold. We chose a value of 1.0, meaning the sum can go up to 100% of the nominal bandwidth of the link.

Parameters	Values
Number of servers	320
Number of edge switches	16
Number of aggregation switches	8
Number of core switches	4
Total number of requests	10000
α	0.5
T_h	1.0
ΔT	180 s
Edge link speed	100 Mbps
Aggregation link speed	1 Gbps
Core link speed	1 Gbps

Table 3.1: Simulation parameters

The general structure of the set is the one already showed in figure 3.2, where we can identify core switches, aggregation switches, edge switches.

Our aim is to test the behavior of our network solutions, so we designed tests so that the bottleneck would be the network, and not the computing resources.

This is how we built our simulations: VM allocation requests follow a Poisson distribution, where the inter-arrival time is exponentially distributed, with an average value of $1/\lambda$. The “holding time” (requested duration of a VM) also follows a Poisson distribution, with an average of $1/\mu$.

In order to provide a realistic representation of traffic, we use a model that is composed of three different kinds of traffic: elephant, mice, hybrid.

- Elephant traffic is characterized by heavy loads and long duration, is the traffic we have for streaming, for file uploading/downloading, etc. VMs that simulate this kind of traffic have $1/\mu$ equal to 42 hours, and requested bandwidth uniformly distributed in range [80, 100] Mbps, with a step of 5.
- Mice traffic is short-lived and of lesser volume, is the kind of traffic we observed when performing routine operations in VDI environments, so $1/\mu$ is set to 13 minutes, and requested bandwidth is uniformly distributed in range [1, 5] Mbps, with a step of 1.
- Hybrid traffic has some hybrid features, between elephant and mice, so $1/\mu$ is 85 minutes, and requested bandwidth is uniformly distributed in range [10, 80] Mbps, with a step of 5.

The parameter $1/\lambda$ is in a range [1.3, 7600] seconds, and serves to tune the load of traffic: the lesser the interarrival time, the more intense the flow of requests is, and thus the amount of VMs the system will have to simultaneously deal with is higher.

3.3.2 Simulation Results

When testing our algorithms, for simplicity we decided to only test the subset given by SD-BF, SD-WF, ND-BF, ND-WF, that is excluding the FF (First Fit) heuristics. This choice was due to simplifications of operations, reduction of time to carry out all tests, and most of all the consideration after some preliminary test that First Fit and Best Fit in practice overlap, since both of them tend to consolidate resource usage in a similar way.

In the beginning, we tested all our algorithms in two different scenarios: in scenario *A* the parameters of the simulations were tuned so that we would have an average load of 1000 VMs simultaneously, causing an amount of average traffic load of about 50 Gbps.

In scenario *B* instead the average load was about 3000 simultaneous VMs, and an average traffic load of 160 Gbps.

A first feature we tested was the blocking probability. Blocking probability [42] is the probability of having a "block" (that is a situation where a new request cannot be accepted), and is calculated as the number of rejected requests over the total number of requests received.

Figure 3.5 presents the evolution of the Blocking Probability (BP) in time.

When the amount of requests and traffic is bigger, obviously the blocking probability is higher. But in both cases we can see that it is the Worst Fit algorithms, be it Server Driven or Network Driven, that obtain the best results (lesser blocking probability). That is, using a balanced approach in resource allocation seems to be beneficial.

3.4 Testbed Set-Up and System Validation

As aforementioned, not only did we carry simulations, but we also built in our premises a real testbed, in order to have a real environment where to see our algorithms at work.

3.4.1 Testbed Setup

In the framework of the EU FP7 OFELIA project [43], we deployed an experimental testbed reproducing a down-scaled Cloud DC. As shown in figure 4.3, the testbed consists of the following facilities:

- 7 OF switches (general-purpose devices equipped with Open vSwitch 1.4 [44])
- 6 servers equipped with Xen Cloud Platform (XCP) [2]. S1 and S2 are Intel Xeon Quad Core 2.33 GHz with 14 GB RAM, S3, S4, S5 and S6 are Quad Core CPU 2.4 GHz with 4GB RAM.
- 1 server for the SDN-DC orchestrator, connected to all the OF switches and XCP servers.
- 1 HTTP web server, which receives VM allocation requests and redirects them to the SDN-DC orchestrator. This server also uses an SQL database for tracking VM allocations.

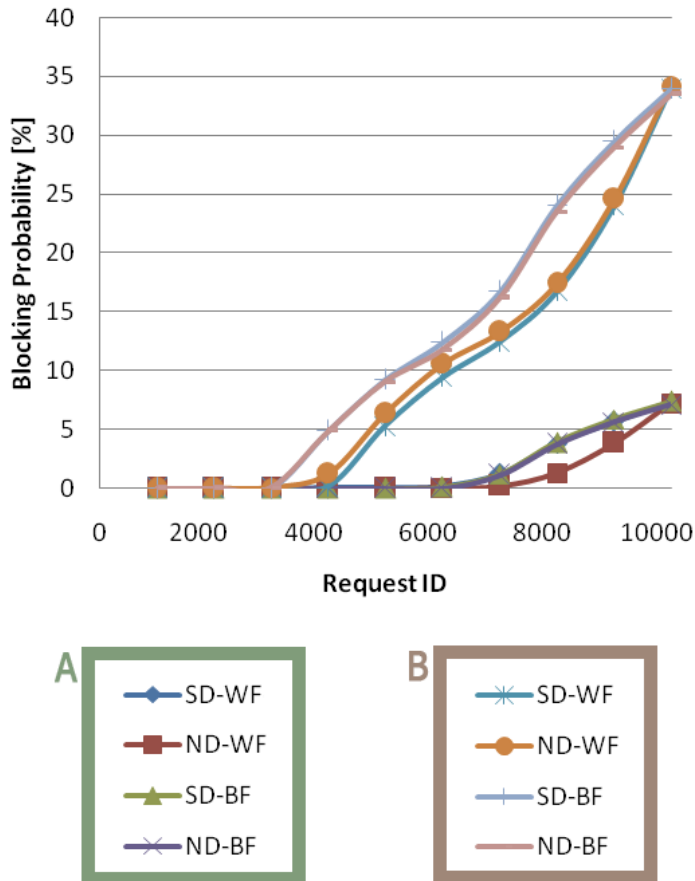


Figure 3.5: Evolution of blocking probability

- All links are 100 Mbit/s. Core links are 200 Mbit/s.

On all servers there is one “original” VM that is turned off, but is cloned to generate new VMs with the requirements requested by the user.

For the sake of simplicity, figure 3.6 only shows the data network (white interfaces), and not the control plane. The SDN-DC orchestrator is directly connected with all devices (switches and servers) through a physically separated control and management network (red interfaces).

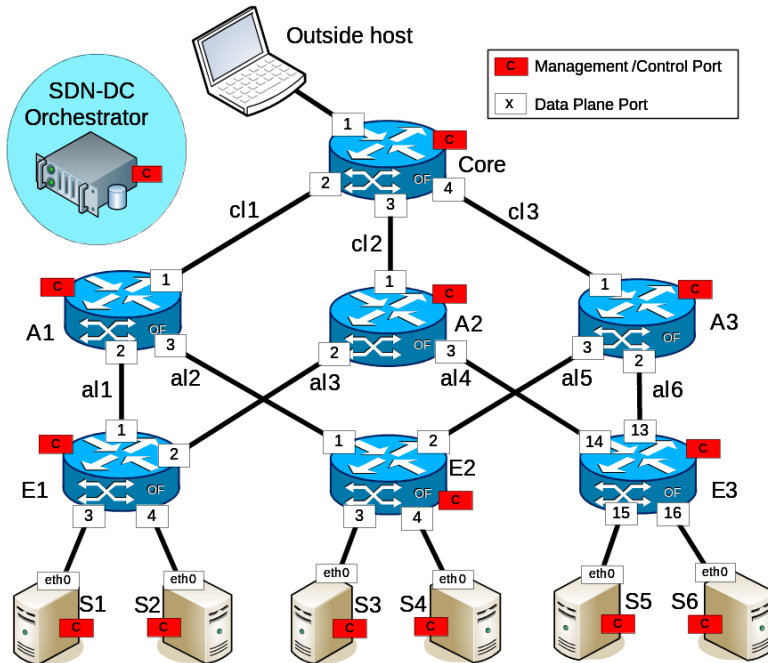


Figure 3.6: Testbed setup

3.4.2 Testbed operation and message flow

This subsection specifies the operation of the SDN orchestrator in the experimental testbed [45]. The sequence of steps followed for requesting and allocating a Virtual Machine.

As we can see in figure 3.7), the following actions occur:

- After signing up, the end user connects to the Web Portal and logs in with his/her credentials (i.e., username and password).
- The end user issues a request for a VM set-up specifying the required VM properties (i.e., CPU capabilities, RAM and Disk Size, network bandwidth, expiration date).
- The SDN-DC orchestrator receives the VM set-up request and executes the Resources Selection and Composition strategies (i.e., Server-Driven or Network Driven coupled with FF or WF policy) based on information status in statistics DC collected in background.
- If the request is accepted (i.e., both a physical server for the VM and a forwarding path from the physical server to a core switch are available), the SDN orchestrator allocates the VM on the selected physical server, and installs the related OF rules in the switches along the selected forwarding path. The OF rules installed in that phase are related to the IP address of the physical server and are only used for testing the reachability of the physical server.

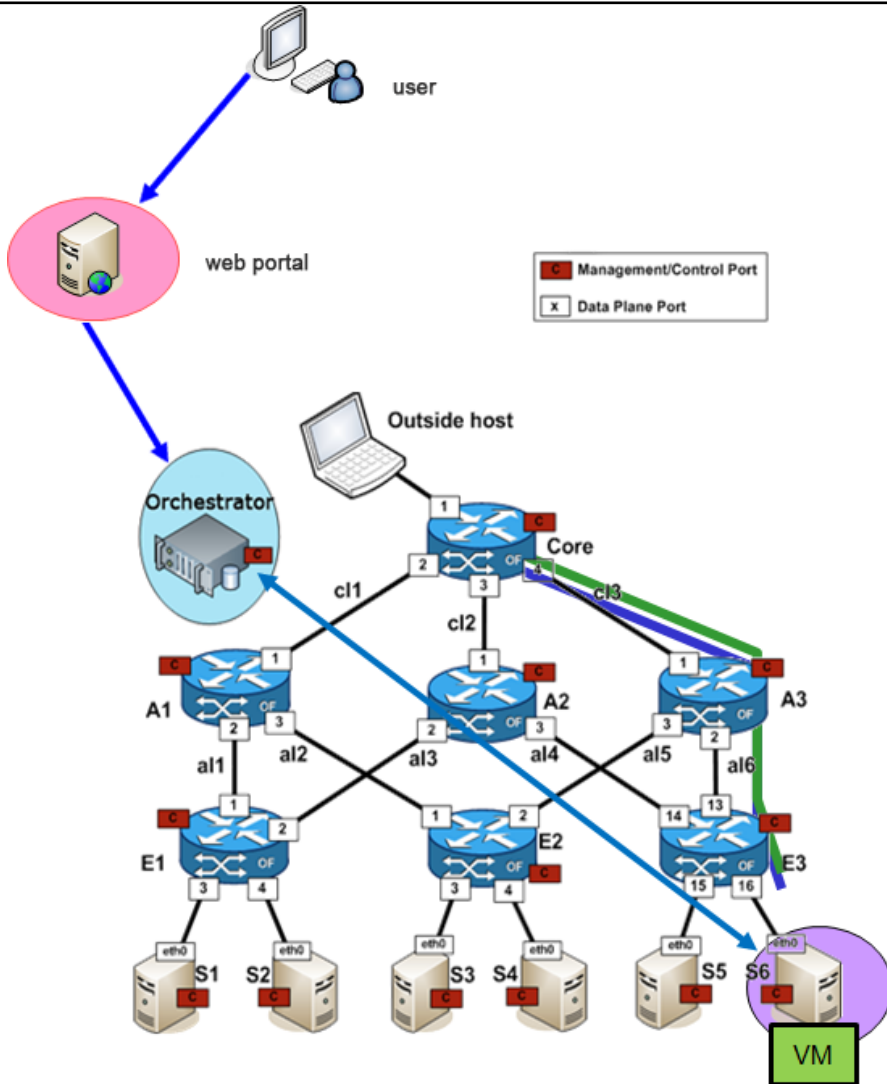
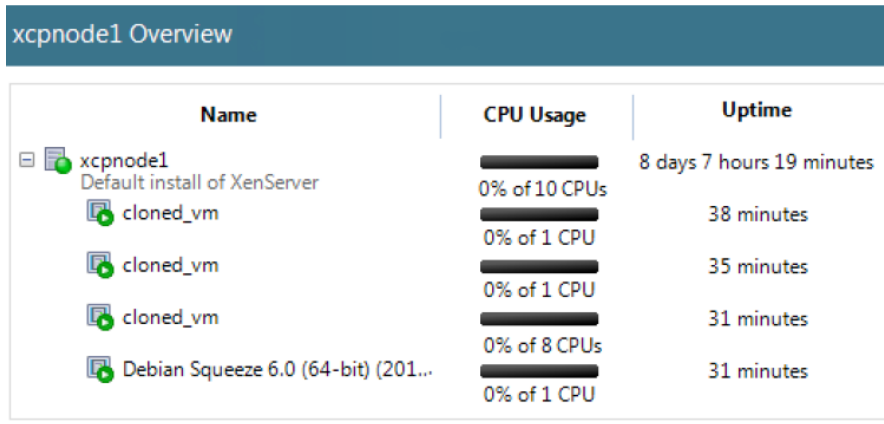


Figure 3.7: SDN orchestrator operation

- At the startup, the new Virtual Machine sends a request for an IP address to the DHCP server.
- The DHCP server assigns a new IP address to the VM.
- Finally, the orchestrator installs OF rules in switches along the VM forwarding path, that enables the reachability of the VM.

Figure 3.8 depict an extract of server utilization after the instantiation of some of the VMs requests. In the example we see that on the server three Virtual Machines have been allocated.

Allocating a new machine, as previously explained, means cloning it from an initial one (Debian Squeeze, in our case). This initial machine is not meant to be used by the user, and does not run any application. It is actually not considered in the counting of used CPU power, because it is not active.



Name	CPU Usage	Uptime
xcpnode1 Default install of XenServer	0% of 10 CPUs	8 days 7 hours 19 minutes
cloned_vm	0% of 1 CPU	38 minutes
cloned_vm	0% of 1 CPU	35 minutes
cloned_vm	0% of 1 CPU	31 minutes
cloned_vm	0% of 1 CPU	31 minutes
Debian Squeeze 6.0 (64-bit) (201...	0% of 8 CPUs	31 minutes
	0% of 1 CPU	

Figure 3.8: Screenshot of the state of Virtual Machine on one of the Xen Servers

In figure 3.9 a portion of the Flow Table in a core switch is shown. We can see the rules that have been instantiated by the SDN controller, leveraging OpenFlow built-in functions, so the switch "learns" where to forward packets (action: on which output port to forward them) depending for example on the source and destination addresses of the packet.

Figure 3.10 shows excerpts from core, aggregation, edge switch, following the path of a flow (in upstream and in downstream) across the hierarchy of switches.

3.4.3 System Validation

In this section we describe initial tests that simply aim at validating our system by showing the behaviour of our algorithms, while dealing with a constant bit rate (CBR) traffic. With reference to figure 4.3, we identify all the available paths as reported in Table 3.2.

We performed four short tests to check the operational functionality of the system in different selection strategies. 20 VM allocation requests were issued with the following requirements: 2 CPU unit, 1 GB disk size, 1 GB RAM and 1Mbps bandwidth. The life time of each VM was greater than the total duration of the test (i.e., we avoid the expiration of the VM during the test). We set the capacity of all links to 15Mbps (i.e., we are in case of oversubscription 1:2). The switches statistics were updated every 35s. After the

EXTRACT FROM THE CORE SWITCH FLOW TABLE

Duration (sec)	Priority	n_pkt	n_bytes	Idle_Timeout, Hard_Timeout	ip, nw_src	ip, nw_dst	action
331	32769	335	24790	0,0	10.0.0.140		output:1
332	32769	335	24790	0,0		10.0.0.140	output:4
523	32769	4	296	0,0	10.0.0.139		output:1
523	32769	4	296	0,0		10.0.0.139	output:4
754	32769	8	584	0,0	10.0.0.138		output:1
754	32769	8	584	0,0		10.0.0.138	output:3
931	32769	4	296	0,0	10.0.0.137		output:1
931	32769	4	296	0,0		10.0.0.137	output:4

Figure 3.9: Extract from the flow table of a core switch

Name	Switches	Links	Served Servers
P1	C-A1-E1	cl1-al1	S1, S2
P2	C-A1-E2	cl1-al2	S3, S4
P3	C-A2-E1	cl2-al3	S1, S2
P4	C-A2-E3	cl2-al4	S5, S6
P5	C-A3-E2	cl3-al5	S3, S4
P6	C-A3-E3	cl3-al6	S5, S6

Table 3.2: Available paths in the topology

allocation, each VM started exchanging CBR traffic at 1 Mbps with the outside host. We set the maximum number of CPUs for each server to 8 (i.e., we supposed that over-provisioning of computational resources is allowed). At the beginning of each test, there is no VM allocated, so all the servers had 8 available CPUs. We dimensioned the number of requests and their parameters to avoid refusal in order to compare the resource utilization using the different orchestration strategies.

In figures 3.11-3.13 we validate our system by showing the behavior of the different resources selection strategies at the end of the experiments.

We present respectively the servers allocation, the path occupancy and the core-link occupation (we report the core link occupation considering that it is the first characterization step of the network resources in network-driven strategies). To understand the behavior of the decision strategies, we should consider jointly the three results.

The ND-BF strategy selects at first step the network resource and then the server to be used for the allocation.

EXTRACT FROM THE CORE SWITCH FLOW TABLE

Duration (sec)	Priority	n_pkt	n_bytes	Idle Timeout, Hard Timeout	ip, nw_src	ip, nw_dst	action
523	32769	4	296	0,0	10.0.0.139		output:1
523	32769	4	296	0,0		10.0.0.139	output:4
331	32769	335	24790	0,0	10.0.0.140		output:1
332	32769	335	24790	0,0		10.0.0.140	output:4

EXTRACT FROM THE AGGREGATION SWITCH (AGGR3) FLOW TABLE

Duration (sec)	Priority	n_pkt	n_bytes	Idle Timeout, Hard Timeout	ip, nw_src	ip, nw_dst	action
523	32769	4	296	0,0	10.0.0.139		output:1
523	32769	4	296	0,0		10.0.0.139	output:2
332	32769	335	24790	0,0	10.0.0.140		output:1
332	32769	335	24790	0,0		10.0.0.140	output:2

EXTRACT FROM THE EDGE SWITCH (EDGE3) FLOW TABLE

Duration (sec)	Priority	n_pkt	n_bytes	Idle Timeout, Hard Timeout	ip, nw_src	ip, nw_dst	action
519	32769	4	296	0,0	10.0.0.139		output:1
519	32769	4	296	0,0		10.0.0.139	output:3
328	32769	335	24790	0,0	10.0.0.140		output:1
328	32769	335	24790	0,0		10.0.0.140	output:3

Figure 3.10: Evolution of blocking probability

While the SD-BF strategy selects the resources in reverse order: the first one is the server selection and then the network resource is considered. We can observe that the first resource chosen is the one that drives the results the most.

For example in figure (in 3.11 we can see that the 20 VMs are allocated in SD-BF using only 5 servers rather than 6 servers as in case of ND-BF.

Considering instead the two worst fit strategies, we can see that the overall behavior is to balance the resources occupation. This feature is very clear analyzing the server and the core links occupations in 3.11 and 3.13 for the two selection strategies SD-WF and ND-WF.

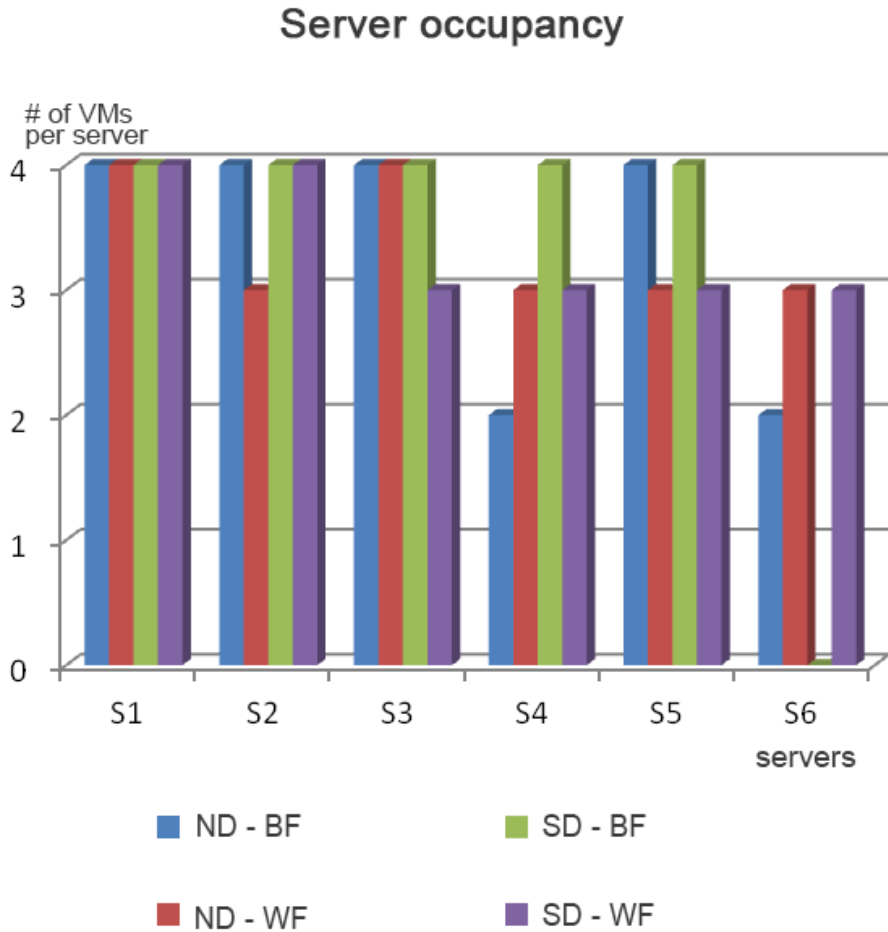


Figure 3.11: Number of VMs per server

In particular 3.13 is very suitable for the comparison of the WF policies with the FF ones: the two WF policies tend to balance the load on all available core links (3 core links used), while the FF ones tend to occupy as much as possible the most used resources (only 2 core links used).

Once we had checked the functionality of the system, and that our algorithms were working, we proceeded to test particular features. For example the blocking probability, that is the probability that a new request will not be accepted.

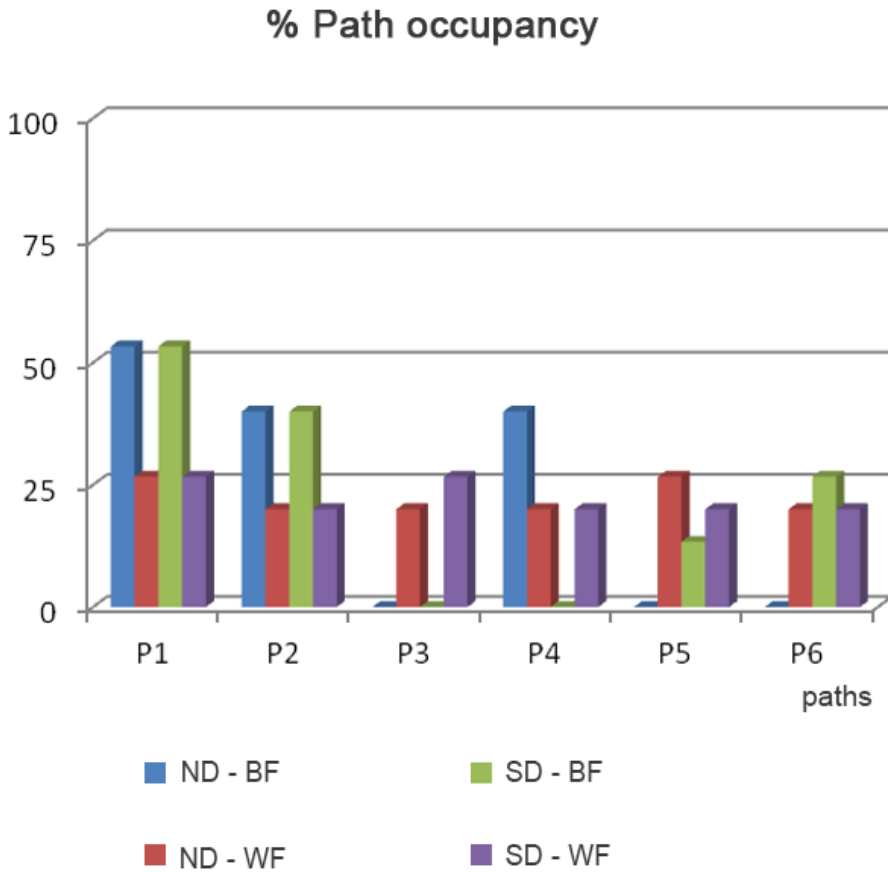


Figure 3.12: Percentage of path occupation

Additionally, we wanted to measure the time it takes to setup a functioning VM for the user who requested it, and understand what factors contribute to the total time. For each test a sequence of VM requests was generated with the following parameters:

- Computational power: uniformly distributed in the interval [2,5] CPU
- Storage capacity: 1 GB
- RAM: 1 GB
- Bandwidth: CBR traffic, uniformly distributed in the interval [2,5] Mbit/s
- Expiration date: after the end of the test.
- Requests inter-arrival time: Poisson distribution with 3000s as mean value.

Network statistics from the switches were updated every 35s. (polling time). The initial computational power of each server was 8 CPU.

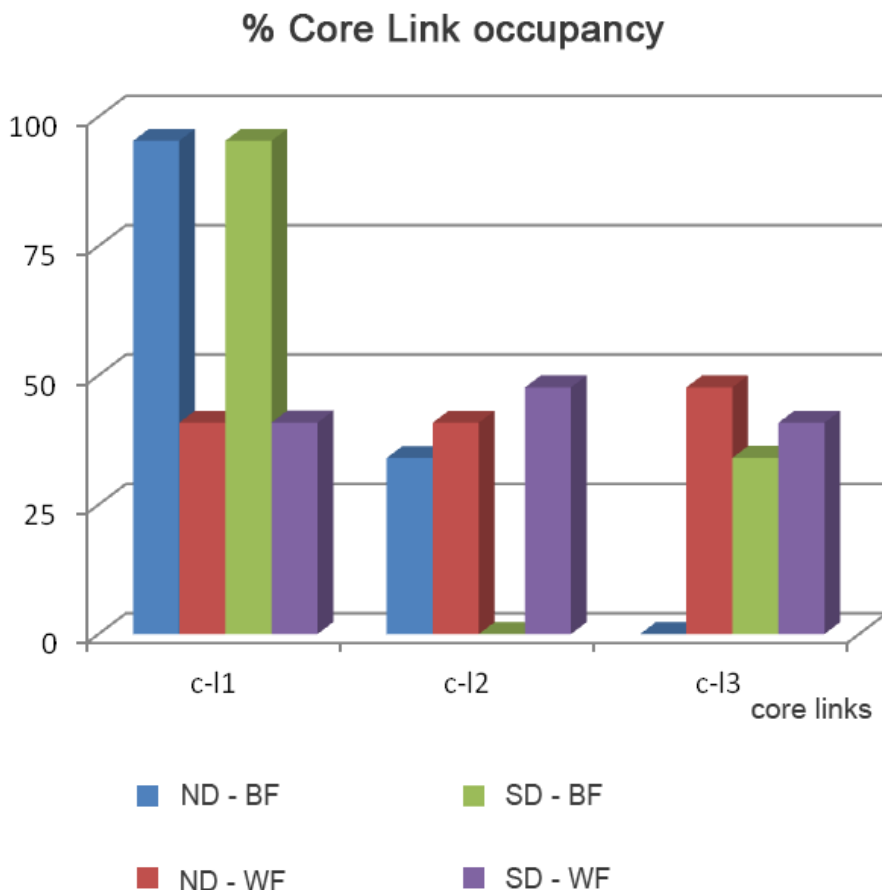


Figure 3.13: Percentage of core link occupation

Figure 3.14 shows the blocking probability. It groups the requests in three chronological blocks: the first third, the second and the third, and shows the probability that the first request that cannot be accommodated falls in one of these. Obviously, it is most common that this happens towards the end, but we see some algorithms perform better than others, as some have lower chances of reaching the block earlier in the simulations.

We also evaluated and characterized the amount of time required for the setup of a VM. We distinguished four main time components: the time for the computation of the allocation algorithm, the time required to clone the “original” VM, the elapsed time waiting for the DHCP request arrival, and the time needed for the setup of the computed flow entries in the switches along the selected path.

We noticed that the DHCP time and the cloning time are responsible for most of the total time necessary to complete the allocation of a new VM, and the duration of such actions mainly depend on the physical server involved, as we show in Table 3.3 and Table 3.4 (confidence intervals). All values are expressed in milliseconds. We can see there are some major differences between different servers, and we can safely say that it is due to hardware differences among the machines.

We see that the contribute from the algorithm computation, as well as the contribute from the setup of forwarding rules, has a negligible impact on the total time.

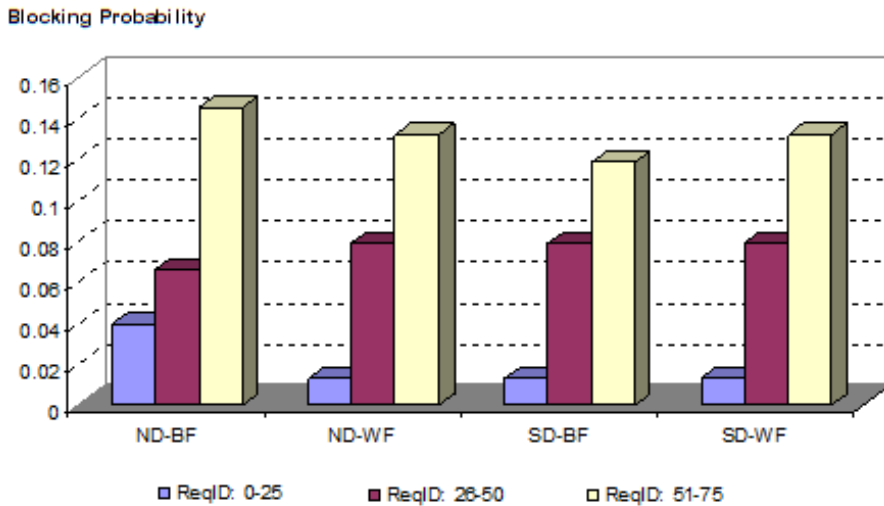


Figure 3.14: Blocking Probability for each RSCA

Server	Algorithm Computation Time (ms)	VM Cloning Time (ms)	IP Address Assignment Time (ms)	VM Setup Time (ms)	Total Time (ms)
S1	12.5	16455.2	29577.2	1.8	45569.7
S2	14.7	17710.2	41177.3	1.9	58905.5
S3	17.9	14573.9	23495.0	1.8	38090.0
S4	16.3	16991.5	26831.7	1.7	43842.5
S5	18.5	16598.9	32199.2	1.9	48819.8
S6	19.7	18293.2	67353.3	1.9	85669.3

Table 3.3: Time for VM setup: per server mean values (ms)

CHAPTER 3. SOFTWARE DEFINED NETWORKING FOR RESOURCE ALLOCATION AND MONITORING

Server	Algorithm Computation Time (ms)	VM Cloning Time (ms)	IP Address Assignment Time (ms)	VM Setup Time (ms)	Total Time (ms)
S1	5.6	94.4	1369.4	0.2	754.8
S2	4.9	1008.8	2548.3	0.2	3309.9
S3	6.9	702.2	1563.4	0.2	2132.9
S4	7.6	6073.0	33231.9	0.6	35299.4
S5	6.3	161.4	1157.6	0.3	1227.4
S6	4.9	2083.2	14913.1	0.3	13590.7

Table 3.4: Time for VM setup: per server CI at 95% (ms)

3.4.4 Evaluation of Traffic Monitoring Strategies

In these tests, we used a traffic with variable rate (VBR), which is typical of Cloud DC applications. In order to have realistic traffic, we injected traces modeled upon the characteristics of real traffic we studied in the project ARPEGGIO (see chapter 2). This allowed us to evaluate the performance of our orchestrator and selection strategies under dynamic conditions.

Tests were repeated with different values for α : 0.2, 0.5, 0.8.

Specific tests were performed to investigate how resource selection and chaining is affected by the configuration settings of the traffic monitoring and load estimation modules (explained in section 3.2).

A core feature made available by those modules, is to take into account past values and calculate average traffic load when making decisions on new VM provisioning requests. This allows the orchestrator to keep track of the kind of traffic the network is experiencing over a longer period of time, and to smooth the effect of very short term situations.

In this case, tests were performed with ND-BF RSCA, that is a policy that considers the network first, and tends to consolidate. We are portraying a network driven approach to better show the effects of these decisions on the network.

We want to show what can happen in different scenarios when a request for a new VM arrives. We start from a situation where link P1 is already loaded by traffic of pre-existing VMs, and at the time indicated with a red line (27 minutes) the request of a particular new VM arrives, requesting a bandwidth of 50 Mbit/s. Note: here we are only showing the network side of the tests.

The blue lines show the traffic load on link P1: light blue is the measurements as we get it from switch statistics (measurement load), while dark blue is the estimation done by our algorithm (estimated load).

Figure 3.15 shows the case with $\alpha = 0.2$: this is the case where history samples are given less weight than the last measured value. In this case the estimations change

almost as rapidly as the traffic on P1. The instant when a request for a new VM arrives (red line), the system estimates an utilization of 75 Mbit/s on P1 (dark blue line), and since the new VM requests 50 Mbit/s, the orchestrator allocates it on a different link (P2, in green), that was unused. But, if the aim was using ND-BF for consolidation, this may have not been the best solution.

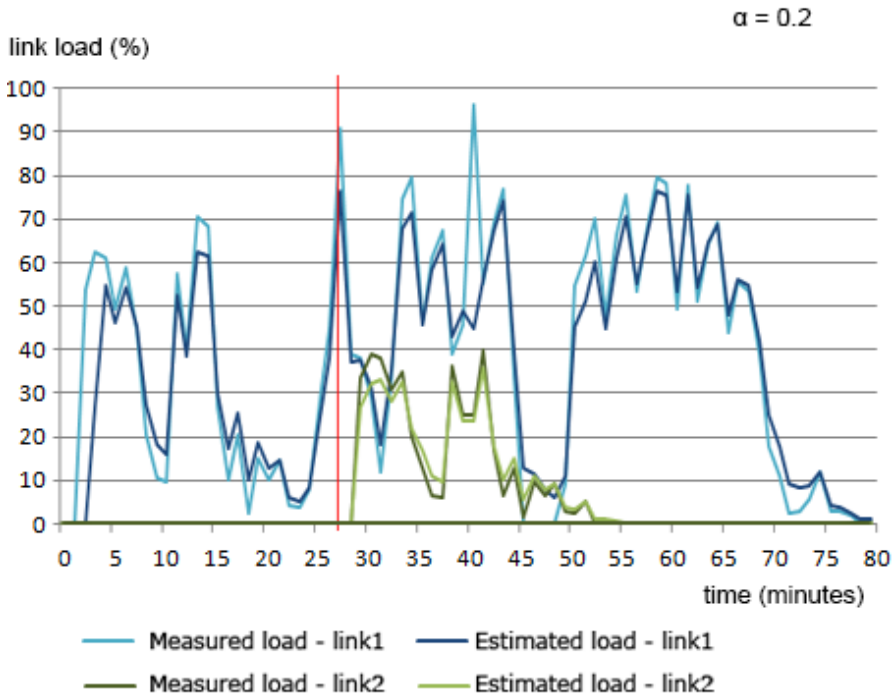


Figure 3.15: Load with history weight 0.2

Let us consider the case with $\alpha = 0.5$ or 0.8 (Figures 3.16 and 3.17): estimations are less biased by sudden spikes, and in both cases the spike of traffic on P1 does not affect the decision, that is to allocate the new VM again on P1, since, on average, it has enough bandwidth left to accommodate it. P2 remains unused, succeeding in the goal of consolidating, that was failed with $\alpha = 0.2$.

We have just shown why we should properly account for past samples, but we want to point out that also accounting too heavily for them would be a mistake, because this can slow the system, preventing it from keeping pace with the changes in the network.

This becomes evident in the next tests. Let us now consider different kinds of traffics.

Figures 3.18, 3.19, 3.20 show again what happens with α equal to 0.2, 0.5, 0.8 respectively.

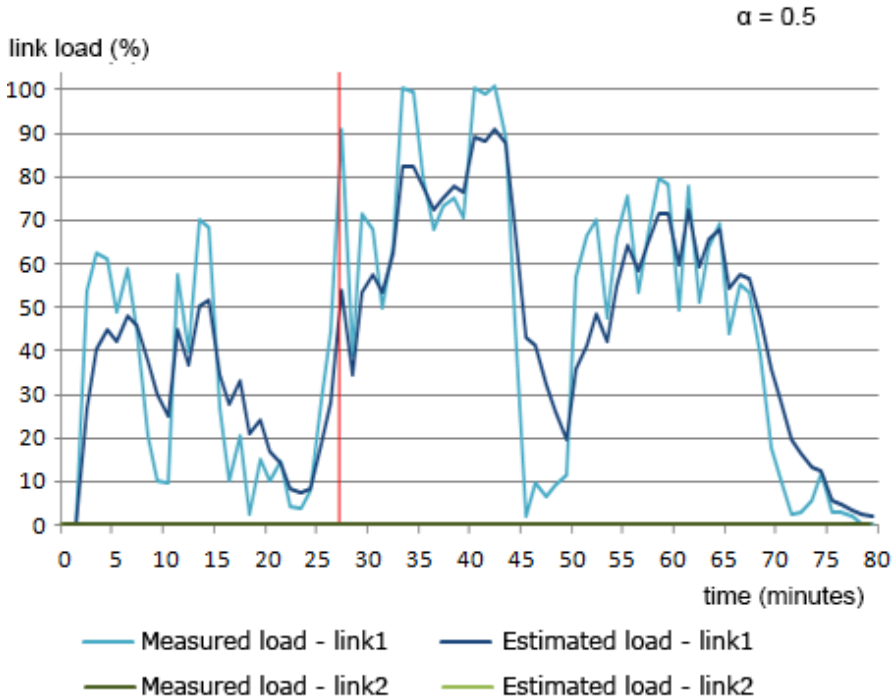


Figure 3.16: Load with history weight 0.5

Let us start with figure 3.18 ($\alpha = 0.2$). Again, there is some traffic on P1, and a new request arrives at the time indicated with a red line.

The orchestrator estimates that P1 cannot accommodate it, and so correctly allocates it on P2. Same happens with $\alpha = 0.5$ (figure 3.19).

Figure 3.20 shows what happens, instead, with $\alpha = 0.8$: the system does not keep up to date enough, and does not realize that P1 does not have enough available bandwidth, so accommodates the new request on P1. This is incorrect: the link is overloaded, and the users may experience a degradation of service.

3.4.5 Aggregate results

Now that we have shown how the value of α affects what happens on a single path, we want to show a wider view, and consider how the different values of alpha affect the achievement of the overall goals of various allocation algorithms when the system is operative.

As previously stated, we can have two main approaches for resource allocation: one that tends to consolidate in order to use as few resources as possible (Best Fit) and one that tends to spread the VMs in a balanced way all over the various resources (Worst

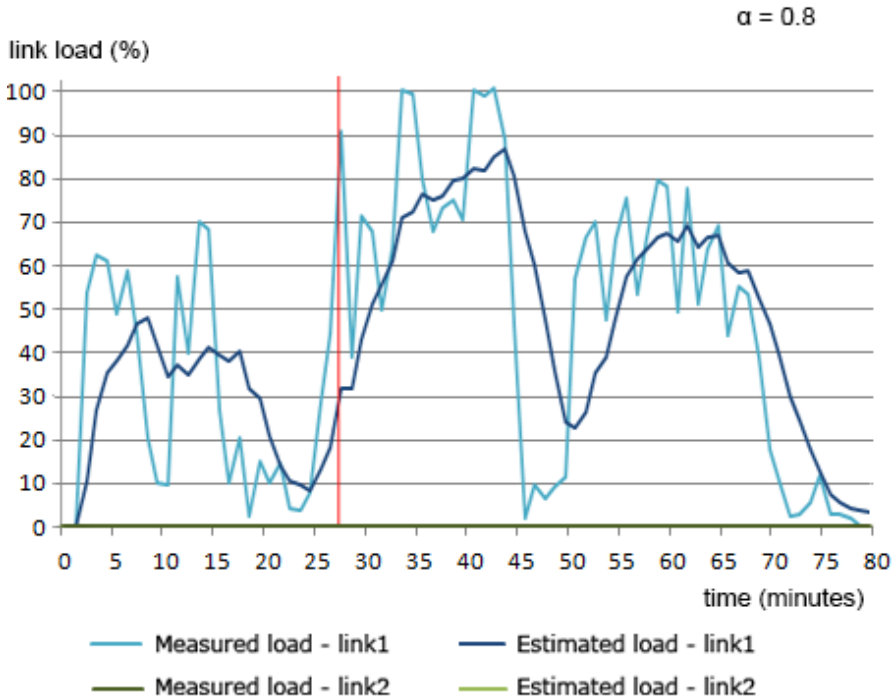


Figure 3.17: Load with history weight 0.8

Fit). We carried out some more tests. Table 3.5 shows the characteristics of the randomly generated test patterns.

Maximum CPU	4
Maximum bandwidth	30 Mbit/s
Average interarrival time	100 s
Average service time	5100 s

Table 3.5: Parameters for randomly generated test patterns

Once set up, a VM starts generating a VBR traffic, as previously described.

Let us start with the Network-Driven Best Fit policy, that considers first network resources, and tends to consolidate their usage.

Figures 3.21 and 3.22 show the graphs for link load for $\alpha = 0.2$ and $\alpha = 0.5$.

On the x axis is the time t in minutes, and on the y axis is the link load in percentage.

Each different colored line represents the load of a different link in time.

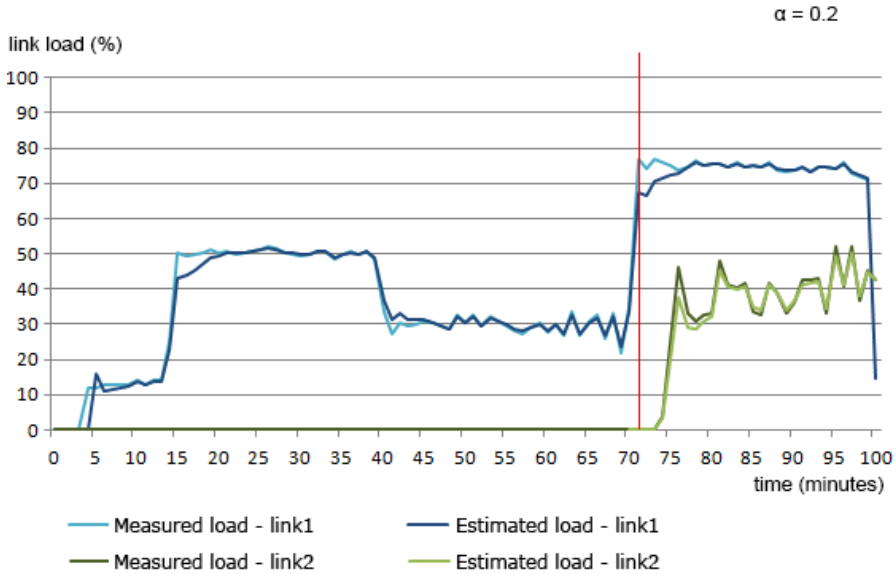


Figure 3.18: Load with history weight 0.2

We decided to analyse all link loads together in order to extract data that would allow us to understand if we were meeting our goals, and which sets of parameters were working best for each goal.

In the case of Best Fit, we want to use some links to the maximum, and leave some completely unused. So parameters to consider will be:

- Number of unused links: the more links left unused, the better the algorithm is
- Maximum link utilization: that is, the maximum percentage throughput recorded among the different links. This is expected to be close to 1, although really reaching 1 may mean that some link is overloaded, which may result in degradation of user experience.

Table 3.6 shows the results for parameters we considered relevant to evaluate the performance of the Best Fit policies.

α	Max link utilization	Unused links
0.2	1.00000	1
0.5	0.90887	2
0.8	0.86983	0

Table 3.6: Results with Best Fit algorithms

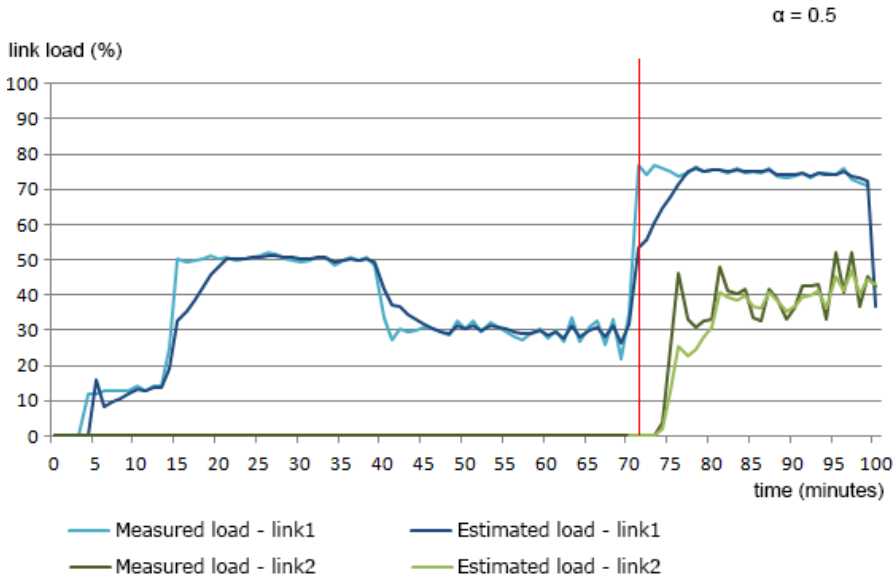


Figure 3.19: Load with history weight 0.5

As we can see, the value of α affects the results, giving us different outcomes.

What we see, again, is that it is best to take into account history (α must not be too small), but we should also find a good trade-off, in order not to slow the system too much, otherwise we risk not being able to keep pace with the evolution of the system.

With $\alpha = 0.2$ some links have become very loaded (reaching 100% of utilization), which means that users may experience a degradation of the service.

An average value for alpha, such as 0.5, achieved the best results, leaving the highest number of links unused (better consolidation) while not reaching the point of overloading any of the links.

Let us now consider some resource selection chaining algorithm using instead Worst Fit approach.

We repeated the same tests, and some relevant results can be seen in table 3.7. Again, figures 3.23 and 3.24 show percentage of link load for $\alpha = 0.2$ and 0.5.

For WF the goals are different, as we aim to spread the VMs and the traffic in a balanced way among resources. So we took into account the maximum link utilization (as again we want to check that links do not get overloaded), but also the average link utilization, and most of all the standard deviation σ .

Mean link utilization is the average traffic load in time.

σ represents the standard deviation from the average load on different links. Let m_k be the average load in time on link k . σ is the standard deviation of $[m_1, \dots, m_N]$, where N is the number of the links.

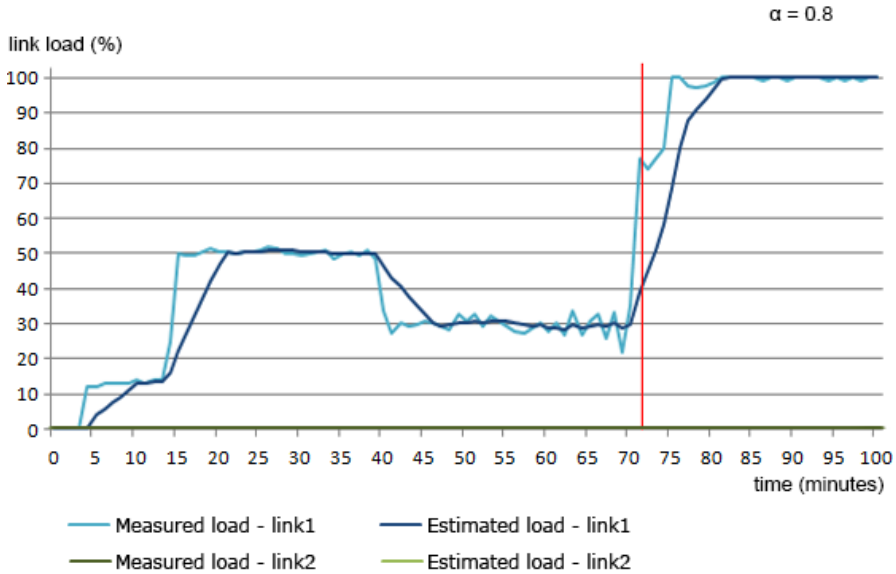


Figure 3.20: Load with history weight 0.8

As we aim at having balanced traffic, we want σ to be small, because it means that there is not a big difference among links, and they are all experiencing the same amount of average traffic load.

α	Max Link Utilization	Mean Link Utilization	σ
0.2	0.96230	0.13444	0.7462
0.5	0.88480	0.13313	0.07268
0.8	0.67553	0.13917	0.07724

Table 3.7: Results with Worst Fit algorithms

In these tests the best results are obtained with $\alpha = 0.2$ and 0.5 (smallest standard deviations), and in particular, even though the differences are quite slight, a value of 0.5 for α gives the best outcome, since we see that the maximum link utilization is lower than for $\alpha = 0.2$, and more important, it achieves the lowest value for σ , which corresponds to a better balance of the load among links.

All these tests confirm that various values of α can be the most appropriate ones, depending on what is the specific goal, although we have seen that an average value for α , such as 0.5 , is a good starting point, as it allows to take into account past measurements enough to not be biased by temporary events, while keeping the system able to properly respond to changes.

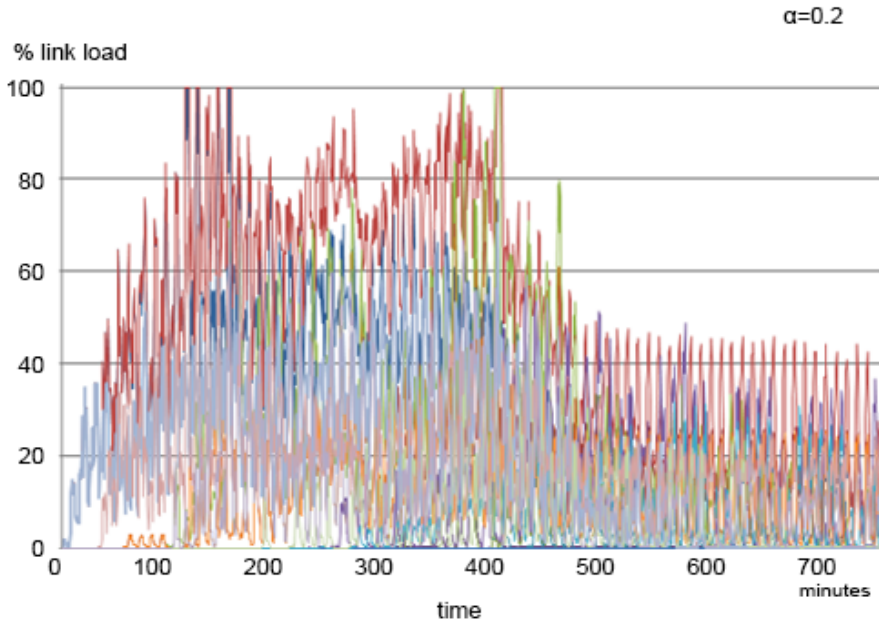


Figure 3.21: Link load for $\alpha=0.2$ (ND-BF)

This parameter can easily be changed and adjusted to fit with different requirements.

3.5 Final remarks

In this project we investigated strategies for resource allocations in a Cloud Data Center environment. Following up a work on studying VDI characteristics and requirements, we designed and tested an SDN-based orchestrator, that allows to coordinate the usage of network and computational resources in a Data Center.

The main goal was to be able to integrate the management of different sets of resources (computational and networking) in order to operate keeping both into account. For this goal, different sets of combined algorithms were envisioned.

A key point of this project was building a scaled down replica of a real system in our premises in the Network Telecommunications lab in the Department of Information Engineering in Pisa, in order to be able to carry real tests, after the sets of simulations.

Another important feature was the implementation of tools that would allow to keep track of the changes in the system, and to respond to these changes selecting the most proper resource allocations from time to time.

An important characteristic of this system is that it can receive as inputs sets of parameters that, as we have shown in several tests, can be used to tune the system according to own specific goals and desired characteristics.

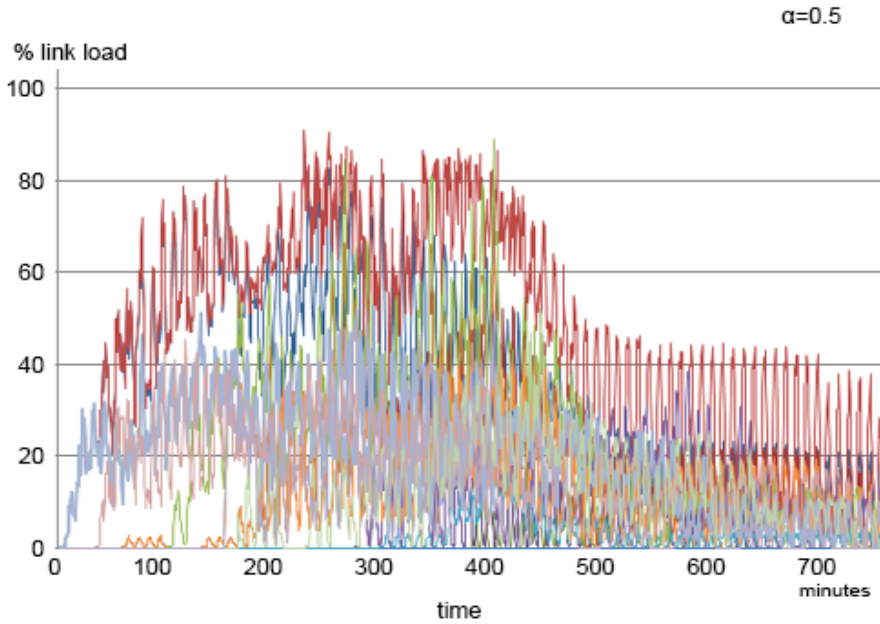


Figure 3.22: Link load for $\alpha=0.2$ (ND-BF)

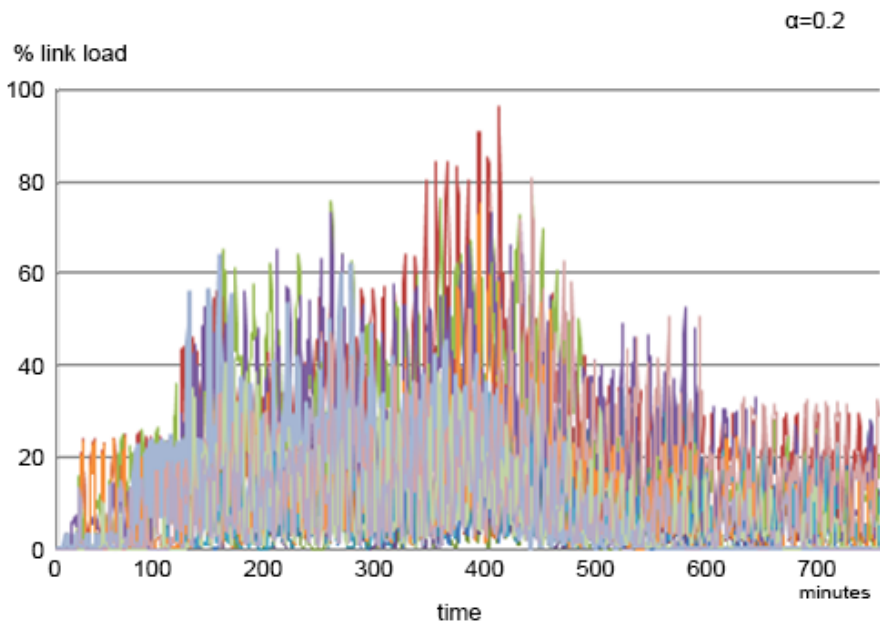


Figure 3.23: Link load for $\alpha=0.2$ (ND-WF)

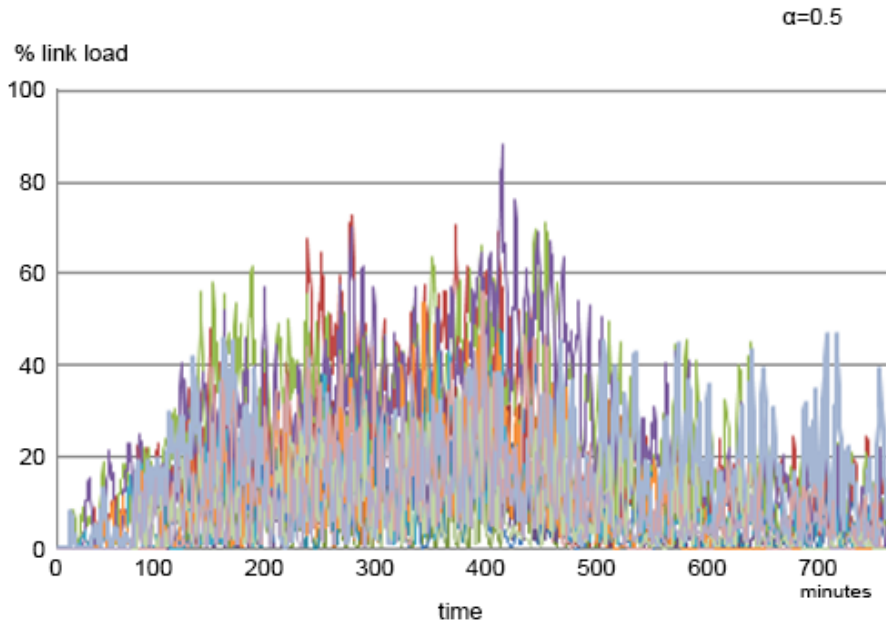


Figure 3.24: Link load for $\alpha=0.5$ (ND-WF)

Software Defined Networking for LTE

4.1 Introduction to LTE

LTE is the most novel technology deployed nowadays for mobile communications.

According to prominent industry such as Cisco [46], global mobile data traffic has been growing steadily by high percentages in the latest years.

The evolution of mobile communications is changing our way to communicate: more and more services are provided for mobile Internet use, new devices have appeared (such as smartphones, netbooks, e-readers, tablets), and this increases the number of users, and of users who actively use these technologies on regular basis, which in return causes even higher growth in mobile demand.

The evolution of mobile network from 3G to 4G has allowed users to achieve higher throughputs: exploiting advanced 4x4 MIMO techniques, we have already reached 300 Mbps in download and 150 Mbps in upload, which is becoming 1Gbps with LTE-A, which also ensures lower latency.

Each advance in performance and bandwidth has been causing surges in mobile subscribers.

From the perspective of a network operator, the more the user expectations increase, the more it becomes fundamentally important to assure high quality, and the most challenging this becomes.

For this purpose, network operators should constantly monitor their own networks, and should proactively detect and solve problems as soon as they appear.

The increasing complexity of computing systems is motivating research on searching for new solutions that aim at simplifying network operations.

One of the issues with mobile and fixed networks nowadays, is they suffer from static implementation, that makes it more difficult for network operators to deploy flexible and innovative network functionalities, which instead are essential in order to provide and test new services, also keeping into account the important aim of reducing time-to-market for products.

Along with the growth of such necessities, we have witnessed the emerging of new technologies that aim to give answers to these problems.

In this context, paradigms such as Software Defined Networking (SDN) [15], for examples, and in general all virtualization techniques, can be a great help, as they give ways to experiment new solutions and architectures, designing a network to work according to customizable rules, that allow to fulfill all these requirements.

This is not only interesting for research, but it can also be a very interesting and viable solution for the industry of communication equipment [47].

In fact, in order for operators to know where to make necessary changes, what to improve, they will need first of all to have good models of the network, of what happens in it. And to achieve that, they need to deploy systems for efficient network monitoring.

The observation of this fact, led to considering that, for very complex networks, such as those that network operators may have to deal with, this will imply the necessity to have a great amount of probes, of points of measurements in the network.

These probes may in some case implement simply basic capturing and processing primitives, but for managing of complex networks, there may be a need of special probes that are able to perform more advanced functions, to exchange the collected data, to aggregate them, to process them in some ways, or to forward them in appropriate ways.

These measurement probes themselves may be considered to be forming an additional network on their own, and this network would have to be designed, managed, configured, changed in a customized way, and it might need to be flexible, as goals and necessities might have to change or be updated, depending on the desired results.

In order to manage such a particular yet delicate network, a solution based on Software Defined Networking would be ideal: the data exchanged among the various probes can be seen as general "user plane", while an ad-hoc control plane, controlled in software, can be deployed to manage the network of probes.

Moreover, more virtualized solutions can be combined. Another promising technology, for example, is NFV (Network Function Virtualization [48]), that is sometimes used in combination with SDN, and that allows to create specific network functions on a general purpose hardware.

In this case, for example, the deployed probes could be actually virtual probes, instantiated via software commands only when needed.

In this chapter, we show the potentiality of using SDN and virtual probes for the purpose of deploying a centralized monitoring system for LTE networks.

After the theoretical modeling, a set of demonstrations have been carried on a testbed that has been setup to simulate a network of LTE probes.

4.2 SDN for Traffic Monitoring

Software Defined Networking allows to have an abstraction of the network. This makes it possible to have different approaches to it, to have more control on it, configure it, and make decisions on it using software tools.

SDN has a centralized approach (since instead of leveraging on distributed routing, there is a central controller that is in charge of making decisions about routing). The

controller will make rules for all the switches in the network, writing a new entry for every flow in their flow tables.

This centralization and abstraction allow researchers to slice and configure the network according to their needs, making it possible to develop new protocols, policies and tests.

Having a centralized approach means that there is an SDN controller that takes decision while having a global view of the whole network, and the way these decisions are taken can be programmed by the one managing the network.

The network is divided in two parallel planes: a forwarding or user plane, where the actual traffic exchanged among users go, and a control plane, that is used for communications between the controller and the devices of the network.

We have already shown how this kind of solution can be useful for some specific environments such as data-center, and for applications such as elastic resource allocation of Virtual Machines [49]. In this chapter we show the work that has been carried to demonstrate the use of SDN-based approach also for LTE monitoring scenarios, and how it can be deployed for the creation of a network of probes to monitor traffic.

Figure 4.1 shows an example of distributed monitoring: there is a number of probes disseminated across the network, and these probes need to communicate in an efficient and secure way with each others, thus forming an additional network themselves.

This new network will need to be constantly reconfigured, changed, controlled in order to get the measurement data we need for different goals at different time. We may want to configure this network among probes with very specific characteristics, in order to be able to synchronize them according to our needs and correlate data captured from different network connections.

Some possibilities of SDN-based approaches for LTE networks have also been discussed in [22].

In collaboration with Telecom Italia LABORatory, we have studied some of the issues that arise in LTE monitoring, and in which configurations these probes can be used [50].

In mobile networks, there is a need for concurrent monitoring of more interfaces at the same time. Probes must therefore be synchronized, and the packets they provide (containing the monitoring data), have to be in the right sequence, in order to verify that each LTE procedure (e.g. Handover Procedure, Attach Procedure, etc. [51]) is performed correctly.

One more scenario that could greatly benefit from the use of SDN solutions, is the one where data collected in different points (for example data coming from different eNodeB's), need to be redirected to a same dedicated server, to collect statistics, to store them, to aggregate them or to compare and process them.

Due to its flexibility, and to the infinite combination of customizable functions that it can offer, we think that deploying the SDN approach for such a system of probes will be useful.

We can decide to manage the network according to the specific measurements we want to achieve, to create abstractions of the network, to segmentate it dynamically, to

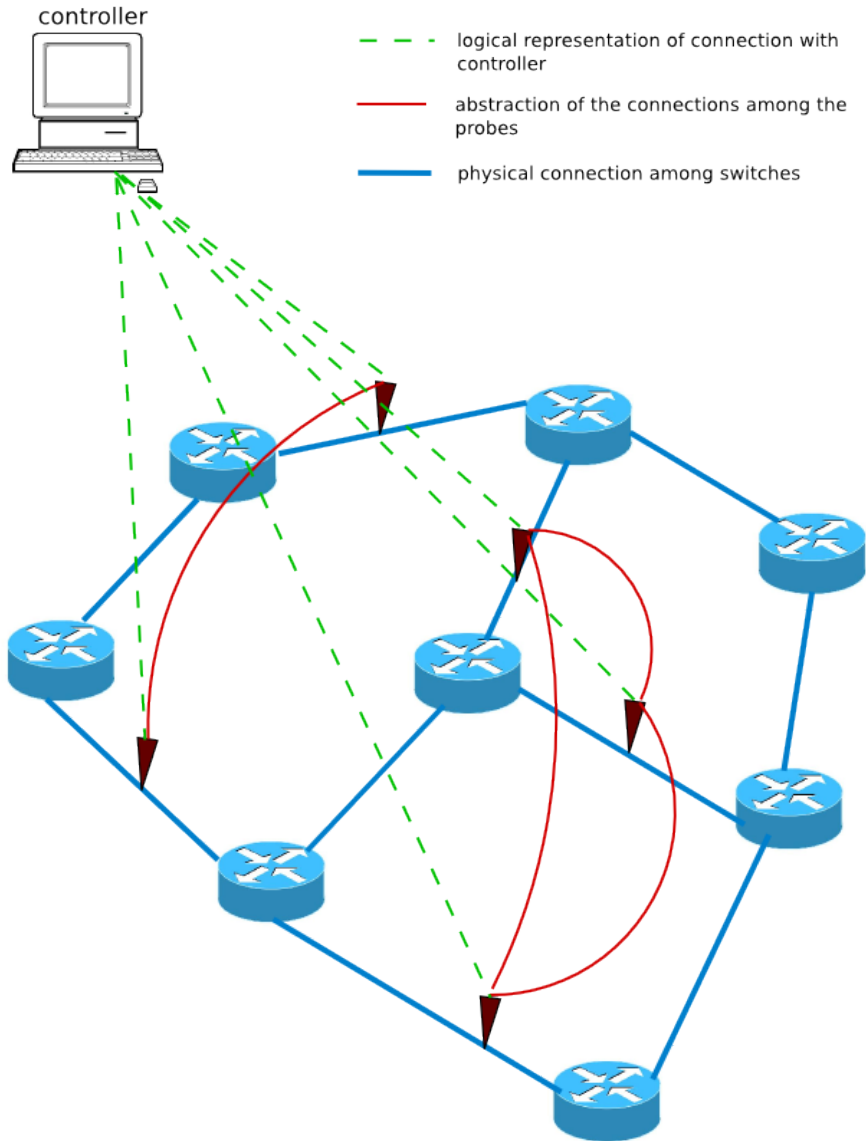


Figure 4.1: Distributed monitoring

install rules to redirect the traffic or specific parts of the traffic in a way that best suits our goals.

Solutions have been designed that leverage possibilities offered by virtualization, in order to create architectures that can respond to these necessities.

For example, we can think of a solution slicing a network in two or more parts that are isolated from each others, and to set up rules for routing such that probes belonging to a same set can exchange packets with each others, but two probes belonging to different sets will not even see each other (for example for higher security, or just to avoid unneeded communications). We can also set up rules such that these different slices of network, even though isolated from each others, are all able to communicate with a common server that will be collecting all the data.

Also, the various slices might need to be reconfigured from time to time, and SDN gives the possibility to do that easily with software tools.

Let us give a more formal formulation for these functions.

Network segmentation

Network segmentation (or segregation) means dividing a network in slices, so that it is possible to communicate among machines in a same slice (or group, or set), while machines belonging to different slices cannot communicate with each others.

This is the case of different sets of probes monitoring different chunks of the network, or performing different kinds of measurements. Or cases where for privacy or security reasons the different sets need to be isolated from each others. And we may need to have the chance to reconfigure these segments, for example to perform different kinds of measurements.

We plan to achieve this goal without using the classical techniques for subnetting, also because the separation we want to achieve is functional, and is independent from the IP addresses of the devices.

Traffic redirect

Another rule we want to implement is the redirecting of some specific traffics: even though we have a sliced network, with different sets of probes isolated from each others, we might have the need to redirect measurement (traffic) from all the probes to a same specific host, that will collect all the traffic of a specific kind (for example, we may want to redirect some specific packets to a machine that will calculate statistics on those data). Or there could be some machines that are allowed to send traffic across different slices to all probes, for example for configuration, setup, synchronization.

After investigating on the subject and stating the goals, a system that gives a simple demonstration of this kind of SDN-based solutions has been implemented, with the purpose of showing for proof-of-concept.

Of course there could be completely different approaches to solve these problems, but they would involve instantiating additional elements such as DHCP server, DNS, or routers that can implement Source or Destination NAT or PAT. We aim at demonstrating how it is possible to implement these functions leveraging only SDN, applying minor modifications in the Python code of the POX controller, without adding other elements to the network.

4.3 Testbed

We designed and implemented a small OpenFlow-based local network, managed with a POX controller, in order to give an example of the functions to implement.

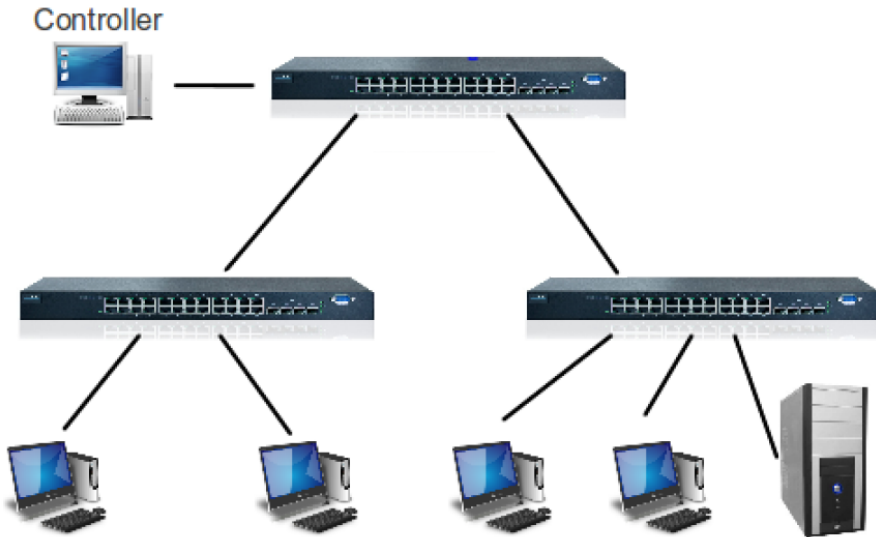


Figure 4.2: Topology

In figure 4.2 we show the topology of our network. It is a simple tree topology, a network consisting of three switches and five end-users.

For this topology, we are going to deploy Virtual Machines (acting as end-users, i.e.: probes), and Open vSwitch, acting as OF switches.

We could have done this in a simpler way using emulators such as mininet [52][53], but we chose to use a local network of VMs instead, because it allows to create a network that more resembles a real one, since the hosts will be real and functioning machines, not just simulated items.

Figure 4.3 shows the topology as it has really been realized with physical hosts.

The three physical hosts, are arranged in a simple tree topology. Host2 and Host3 use Ubuntu 12.04 (kernel 3.3) as operating system, and are equipped with 4 GB of RAM. Both of them use VirtualBox [54] as Hypervisor, that is used to instantiate the Virtual Machines on the two hosts (two VMs on Host2, representing to probes, and three on Host3, representing two more probes and one server).

Each VM is configured with Xubuntu 10.04 and 512 MB of RAM, that is the minimum amount required.

Host1 uses Ubuntu 10.04 (kernel 2.6) as operating system and is equipped with 2GB of RAM.

Open vSwitch is installed on all three hosts, and behaves as an OpenFlow switch. Figure 4.3 offers a more precise idea of the testbed.

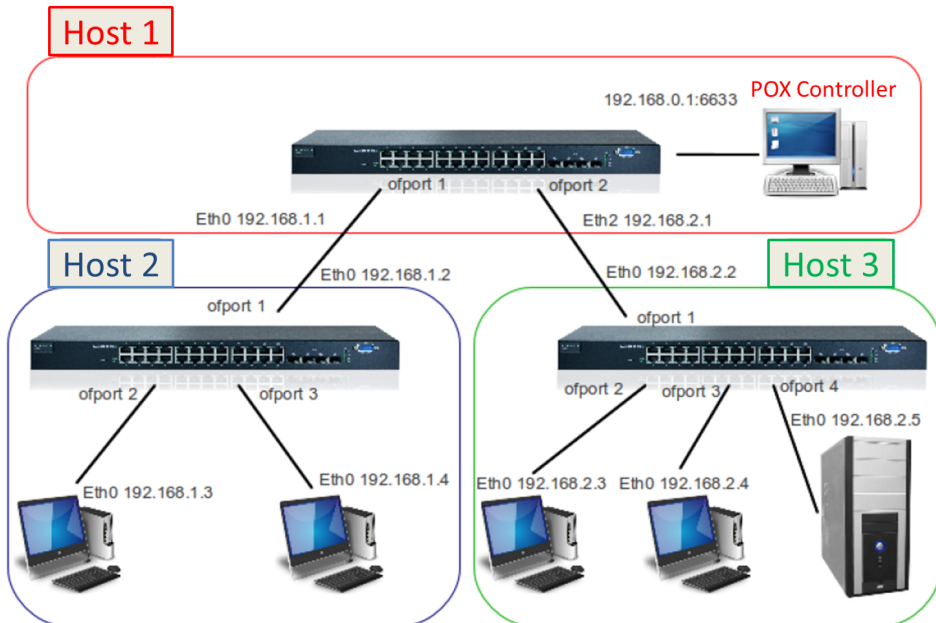


Figure 4.3: Testbed

4.4 POX Controller Configuration

POX controller was downloaded from official git repository [55] and installed on Host1. We chose to modify *l3_learning.py* module to implement network segmentation and traffic redirection.

Network segmentation

For network segmentation, we decided to bind IP addresses this way:

- Group 1: 192.168.1.3 and 192.168.2.3
- Group 2: 192.168.1.4 and 192.168.2.4

as shown in figure 4.4.

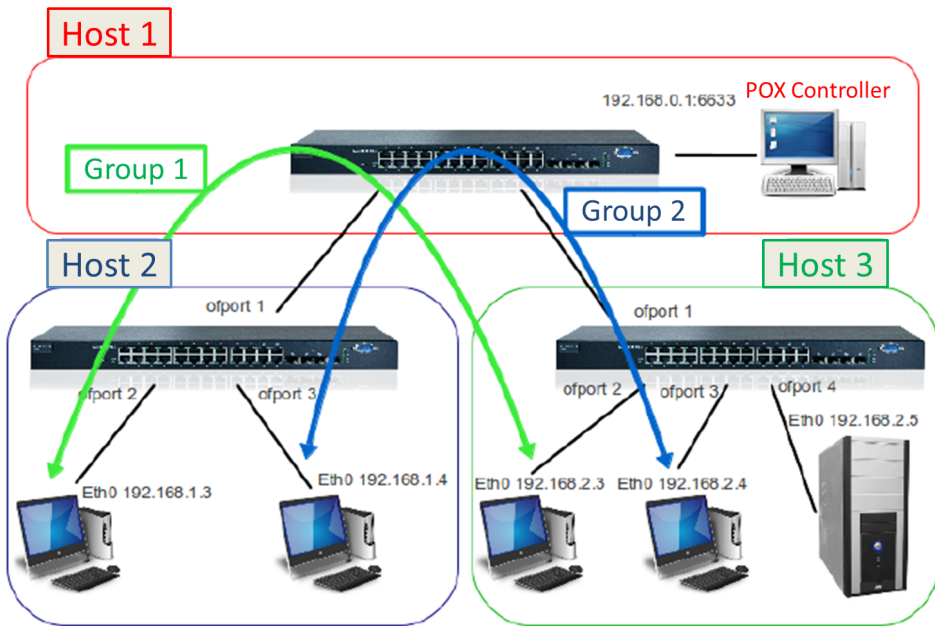


Figure 4.4: Network Segmentation

To achieve this, the POX controller was re-programmed so that it would add appropriate rules to the Flow tables of the OF switches.

First we defined a function `_handle_ConnectionUp`, that is an event handler, triggered by an event listener. In our case, the event is the connection between the controller and the switch. As soon as such connection is detected, the controller instructs the switch on how to manage IP packets, basing on source and destination address. This is an extract of python code as example.

```
#### Source address 192.168.1.3
event.connection.send( of.ofp_flow_mod(
    action=of.ofp_action_output(port=of.OFPP_NONE),
    priority=65535,
    match=of.ofp_match(
        dl_type=0x800,
```

```

nw_src="192.168.1.3",
nw_dst="192.168.1.4" )))

event.connection.send( of.ofp_flow_mod(
    action=of.ofp_action_output(port=of.OFPP_NONE),
    priority=65535,
    match=of.ofp_match(
        dl_type=0x800,
        nw_src="192.168.1.3",
        nw_dst="192.168.2.4" )))

```

match statement will detect IP packets (*dl_type=0x800*) with source address 192.168.1.3 and destination address 192.168.1.4 or 192.168.2.4, that is one of the non-permitted connections.

If there arrive a packet that matches this rule, at first the switch will forward it to the POX controller, since it does not have rules for it yet. The POX controller receives it, matches it to its rules, and does what *action statement* specifies: enforces rules that forward the packet to port *OFPP_NONE*, that is equivalent to dropping the packet.

Traffic Redirect

We also want to be able to redirect some specific kind of traffic to a specific host. As an example, we are setting a function that will redirect all HTTP traffic: it could be any other kind of rule.

We instantiated one of the VMs to work as an HTTP server, so that it will generate HTTP traffic. All the traffic with TCP port equal to 80 or 8080 (that is: HTTP traffic) is redirected to this virtual server.

The goal is to obtain the behavior depicted in figure 4.5.

Unlike what happened for network segmentation, now the controller may need to send different instructions to each Open vSwitch, since for each of them the server to reach will probably be on a different port, so the Flow Table of each server must receive its own entries.

For example: if outbound HTTP traffic is generated by one guest on Host2, it must be forwarded to *ofport 1*, to be received by *ofport 1* of the virtual switch on Host1. From here, the packet will be sent from *ofport 2* to *ofport 1* of Host3, and will reach the server through *ofport 4*. The traffic generated by HTTP server will be handled and forwarded according to the rules laid down in the routing module executed by POX. An extract of python code is shown below:

```

### Outbound Traffic
if event.dpid ==67384867511: ###Dpid of PC1 vSwitch
    event.connection.send( of.ofp_flow_mod(
        action=of.ofp_action_output(port=2),

```

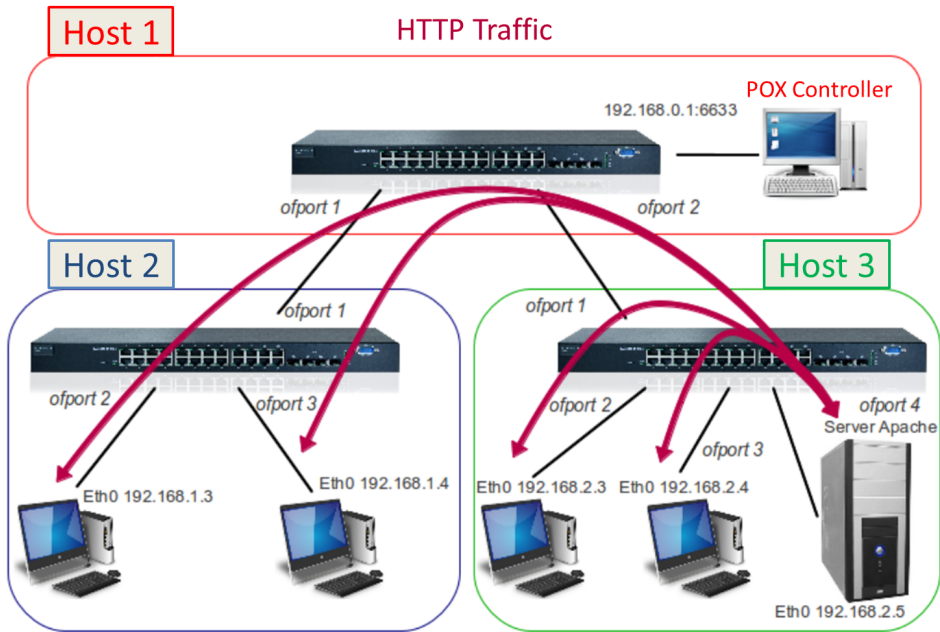


Figure 4.5: Traffic Redirect

```
priority=1,
match=of.ofp_match(
dl_type=0x800,
nw_proto=6,
tp_dst=80 )))
```

4.5 Experimental results

In this section we are going to show the results obtained with our testbed.

Network Segmentation

The modified version of the POX code will segmentate the network as shown in figure 4.4: the network will be divided in two groups that cannot communicate with each other.

Verifying this function is a simple task: we only need to show that machines belonging to the same groups are able to reach each others with a ping command, while machines belonging to different groups cannot do that.

The following figures show the output of the command *ping* on different machines.

Figure 4.6 shows that from host with IP 192.168.1.3 we can correctly reach the host with IP 192.168.2.3 (as expected, since they belong to a same slice).

```
root@Guest1:/home/guest1# ping 192.168.2.3
PING 192.168.2.3 (192.168.2.3) 56(84) bytes of data.
64 bytes from 192.168.2.3: icmp_seq=1 ttl=64 time=57.0 ms
64 bytes from 192.168.2.3: icmp_seq=2 ttl=64 time=0.939 ms
64 bytes from 192.168.2.3: icmp_seq=3 ttl=64 time=7.08 ms
^C
--- 192.168.2.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2009ms
rtt min/avg/max/mdev = 0.939/21.673/57.085/25.105 ms
root@Guest1:/home/guest1# █
```

Figure 4.6: Reachability test from IP 192.168.1.3 to IP 192.168.2.3

Figure 4.7 shows instead how from host with IP 192.168.2.4 it is impossible to reach the address 192.168.2.3: even though they are really on a same physical host, but rules have been set so that they cannot communicate with each others..

```
guest2@Guest2:~$ su
Password:
root@Guest2:/home/guest2# ping 192.168.2.3
PING 192.168.2.3 (192.168.2.3) 56(84) bytes of data.
^C
--- 192.168.2.3 ping statistics ---
7 packets transmitted, 0 received, 100% packet loss, time 6001ms
root@Guest2:/home/guest2# █
```

Figure 4.7: Test for isolation between IP 192.168.2.3 and IP 192.168.2.4

Traffic Redirect

We want to make sure that the HTTP traffic will be redirected on the *ofport* associated with the HTTP server, independently of other parameters, such as the real destination IP of the flow.

In order to verify what is happening, we captured traffic on the server, and figure 4.8 shows the results.

The IP address of the server is 192.168.2.5, and we expect to find packets that do not have this as destination address, but have been redirected here due to being HTTP.

In fact, we find packets that have 8.8.8.8 as destination IP, and are found here because they have been redirected here.

The image shows a Wireshark interface with a filter set to 'tcp'. The packet list pane displays several TCP SYN packets. The first packet (No. 1) is from 192.168.2.3 to 8.8.8.8. Subsequent packets (Nos. 2, 3, 4, 10, 11) are also from 192.168.2.3 to 8.8.8.8. The packet details pane for the selected packet (No. 1) shows: Ethernet II, Src: CadmusCo_08:57:04 (08:00:27:08:57:04), Dst: aa:11:00:ae:22:32 (aa:11:00:ae:22:32); Internet Protocol Version 4, Src: 192.168.2.3 (192.168.2.3), Dst: 8.8.8.8 (8.8.8.8); Transmission Control Protocol, Src Port: 33500 (33500), Dst Port: http (80), Seq: 0, Len: 0.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.2.3	8.8.8.8	TCP	74	33500 > http [SYN] Seq=0 Win=0 Len=0
2	0.000150	192.168.2.3	8.8.8.8	TCP	74	33500 > http [SYN] Seq=0 Win=0 Len=0
3	2.999382	192.168.2.3	8.8.8.8	TCP	74	33500 > http [SYN] Seq=0 Win=0 Len=0
4	2.999671	192.168.2.3	8.8.8.8	TCP	74	33500 > http [SYN] Seq=0 Win=0 Len=0
10	8.998912	192.168.2.3	8.8.8.8	TCP	74	33500 > http [SYN] Seq=0 Win=0 Len=0
11	8.999189	192.168.2.3	8.8.8.8	TCP	74	33500 > http [SYN] Seq=0 Win=0 Len=0

Figure 4.8: Redirecting of HTTP traffic

4.6 Practical scenarios

We have given examples of possible functions that can be interesting and that can be easily implemented by leveraging SDN. We have developed a testbed to give concrete examples of these functions and to test the architecture.

Software Defined Networking allows for new ways to approach network architecture, and it offers researchers a valuable tool to implement and test new solutions.

Concerning *network segmentation*, we can use these results to isolate two or more virtual probes belonging to different sets. For example, probes that monitor different links (e.g. S1-MME and S1-U interface in LTE network) could need to be synchronized in order to correlate traffic properly. This is the typical scenario that we observe in TILAB (Telecom Italia LABoratory): we aim to replace physical probes with commodity or programmable hardware, to deploy on-the-need and on-the-fly virtual probes capable of capturing traffic and returning some relevant results in the proper way.

Concerning *traffic redirect*, instead of forwarding HTTP traffic, we can configure each Open vSwitch to route some specific kind of traffic to a specific server. From a network operator perspective, it could be interesting and useful to be able to select specific data that need to be analyzed by a centralized point. For example, when a problem appears

in the LTE network (e.g. increasing number of Attach Reject in mobile network) it is extremely important to circumscribe the area where the problem is, analyze the traffic and solve the problem as quick as possible. In these cases, we may choose to focus particularly on the data from the probes that are in that area, and perform further analysis on the data they are collecting.

Hardware acceleration

All through this PhD some work with hardware was carried [56], and in the third year of this path there was the chance to do a research stay for some months the Computer Laboratory of the University of Cambridge, working in the NetOS group.

The work there focused on hardware developing, working with the high-performance FPGA-based board for networking, NetFPGA, and in particular with the latest release, SUME.

5.1 NetFPGA SUME

NetFPGA SUME [57], portayed in figure 5.1 is the newest, most advanced board of the NetFPGA family [58]. This board was realized thanks to a collaboration between University of Cambridge, Digilent and University of Stamford. It is an open-source project.

The main features of the board include:

- Four SFP+ interface (4 RocketIO GTH transceivers) supporting 10Gbps
- PCI-E Gen3 x8 (8Gbps/lane)
- QTH Connector (8 RocketIO GTH transceivers)
- Two SATA-III ports
- One HPC FMC Connector (10 RocketIO GTH transceivers)
- Three x36 72Mbits QDR II SRAM (CY7C25652KV18-500BZC)
- Two 4GB DDR3 SODIMM (MT8KTF51264Hz-1G9E1)
- MicroUSB Connector for JTAG programming and debugging (shared with UART interface)
- Two 512Mbits Micron StrataFlash (PC28F512G18A)
- Xilinx CPLD XC2C512 for FPGA configuration
- User LEDs and Push Buttons

Since the NetFPGA projects have now been going on for many years, there are many open-source projects [59] already developed on these boards by a worldwide community of researchers and companies.

During the research stay in Cambridge, a contribute has been given on the testing and release of the new FPGA-based network board NetFPGA SUME, by building Python scripts for testing and providing feedback on functionality.

But the main contribution was working on the porting of an Open Source Network Tester project (OSNT) [60] on the new, better performing board.

5.2 Open Source Network Tester

OSNT was conceived given the necessity to have better, efficient and more flexible solution for network monitoring and testing.

Nowadays, in fact, networks have an increasingly important role, thus it is fundamental to ensure their reliability, and to monitor their development and performances.

Notwithstanding, most commercial solutions are close, inflexible, expensive, do not adapt to the necessity of designing and monitoring new architectures and protocols, and easily go out of date respect to the new technologies emerging all the time. It is easy to see that especially for research, these solutions are not the best tools.

OSNT instead is an open source platform, which means it collects the contributes of many, and is built with the goal of providing a tool for prototyping, so allows all the needed flexibility for the users, especially if they are researching and experimenting with new network solution. All this beside the benefits of a lower cost.

It is a capture system and traffic generator, designed as a portable, scalable architecture, but currently optimized for NetFPGA boards. A previous version was already implemented on NetFPGA 10G, but the new features of NetFPGA SUME will allow a growth in performance and efficiency.

One of the advantages of OSNT is that it allows to test not only single elements of the network, but also wider portions. Also, it can work both as generator and capture system, both modes working with high hardware precisions, allowing to test any traffic and working condition and analysing the results with high resolution.

The main modes of operation are four:

- *OSNT Traffic generator*: a single board is used as generator, and will also receive the traffic from the element under test (which can be a piece of network equipment, or a portion of the network)
- *OSNT Traffic monitoring*: a single board is used for monitoring: it can capture packets, filter them if necessary, and send them to a host for further analysis. The forwarding to the host easily becomes the bottleneck of the system when operating at full line rate, thus there are some techniques of packet cutting, parsing, hashing that increase the throughput.
- *OSNT Hybrid system*: it is possible to combine the two previous modes, using them jointly in a single board, that on one side generates real or synthetic traces with high time precisions, and on the other side traffic is captured with high precision again, allowing to characterize the element under test with high resolution

- *OSNT Scalable system*: while the previous three modes envisioned using a single board, more boards can be deployed in the network and synchronized, allowing to test and monitor wider portions

The main contribution of the research stay was the porting of some of the fundamental hardware modules on the new board, specifically modules for packet cutting, interpacket delay and rate limiter.

These modules are located in the generation architecture and in the monitoring architecture.

5.2.1 Generation

Traffic generator, as shown in figure 5.2, consists of micro-engines, each of them capable of supporting multiple protocols at various levels (network level, transport level, application level, etc). The support for the most common protocols is already in place, while the modularity of the architecture allows to easily add functions for additional protocols to be used.

Moreover, multiple flows, even with different characteristics, can be generated in parallel, independently.

Each micro-engine consists of various modules:

- *traffic model*: this module contains informations about the traffic to generate (e.g.: packet length, etc)
- *flow table*: this module includes header templates, used to generate packets. The various fields can be set according to necessities
- *data pattern*: this module creates the payload of the packets, that can be random or with given characteristics.

All these modules are used to generate synthetic traces. But it is also possible to replay captured or computer-generated traces, leveraging on the micro-engine *pcap replay*.

In the previous implementation of OSNT it was only possible to replay stored traces, limiting de facto the possibilities, as it was only possible to store traces in the order of some megabytes. In this new implementation, the more advanced solutions for communicating with a host allow to receive traces from a host and replay them at full line rate.

After the line of micro-engines (that can and should be used in parallel to create multiple flows), there is a per port arbiter that distributes the traffics on the output ports, where we find additional model that shape the traffic: a *delay module* and a *rate limiter*. Both modules work with parameters that are set by registers. These registers can be read and written in real time from a host.

Right before being transmitted, the packets are assigned a high precision timestamp (precision around 6.25ns), as previously explained

5.2.2 Monitoring

The architecture for traffic monitoring is shown in figure 5.3. This allows to capture traffic on the interfaces, and perform actions such as filtering, collecting statistics, etc.

When a new packet arrives, after assigning a timestamp, the packet is parsed, that is, information of interest is searched in the fields of the headers. This information typically allows to perform a preliminary filtering, allowing to consider only the flows we are interested in. Typically, further analysis will be carried by a software system, to which these flows are forwarded.

If the throughput is high, the connection between OSNT and the host that receives the traces can become the bottleneck of the system. OSNT provides solutions to keep the throughput at the levels reached in hardware.

These tools are mostly based on cutting the packets after the header, and performing hash functions on the rest of the body, in order to send a "reduced" version of the packets to the system, but without losing the important information.

5.2.3 Final remarks

All the functions presented, make it quite clear how such a solution can be a great resource not only for the goals of research, but also for enterprise environment and for use in industry.

The characteristics of LTE traffics, for example, make OSNT a great candidate for the needs of the networks.

The capture functions, in fact, allow to filter traffic, selecting specific flows to be further analysed and monitored. These flows can be chosen according to many policies: it can be a tuple (i.e.: identifying a flow by the source and/or destination IPs, ports, etc), or specific protocols, or packets tagged in customized way.

Thanks to the module *packet_cutter*, it is possible to forward to a system even just some parts of the filtered packets (e.g.: only the header), in order to maintain high throughputs.

Moreover, thanks to modules like *rate_limiter*, *interpacket_delay*, it is possible to generate many kinds of traffic, set throughput, delays, interarrival times.

But above all, it is a useful feature that the two modalities, generation and capture, can be used jointly, on a same board in different pipelines, so to put under test a network or a part, achieving great precisions both in generation and in capturing of packets.

Moreover, we must not forget that OSNT is meant to be scalable, and the system can be distributed on different network boards around the network, all coordinated and synchronized. Given the high degree of flexibility and customization, this allows to use novel approaches to test wide portion of networks, generating patterns that can recreate any kind of traffic and network conditions.

The new implementation of OSNT, on the new NetFPGA SUME board, which includes SFP+ interfaces, makes this solution even more attractive for LTE networks, especially for monitoring traffics on the eNodeB interface. We can in fact connect such board on the S1

interface (figure 5.4) and use OSNT to collect data on the traffic or aggregate flows, or even more to generate or replay appropriate traces with all desired parameters.

For example, only a minor change to adapt the parsing module is enough to have an ad hoc solution for GTP, in order to collect statistics on user traffic. Or, in the same way, adapting the parameters for headers in traffic generation, it is possible to spread appropriate traffic patterns in the network.

The infinite possibility to choose delays, throughput limitations, jitter, or even simulate packet losses, opens many opportunities.

All these approaches can be applied for managing S1-AP protocol as well, thus collecting statistics on traffic on the control plane.

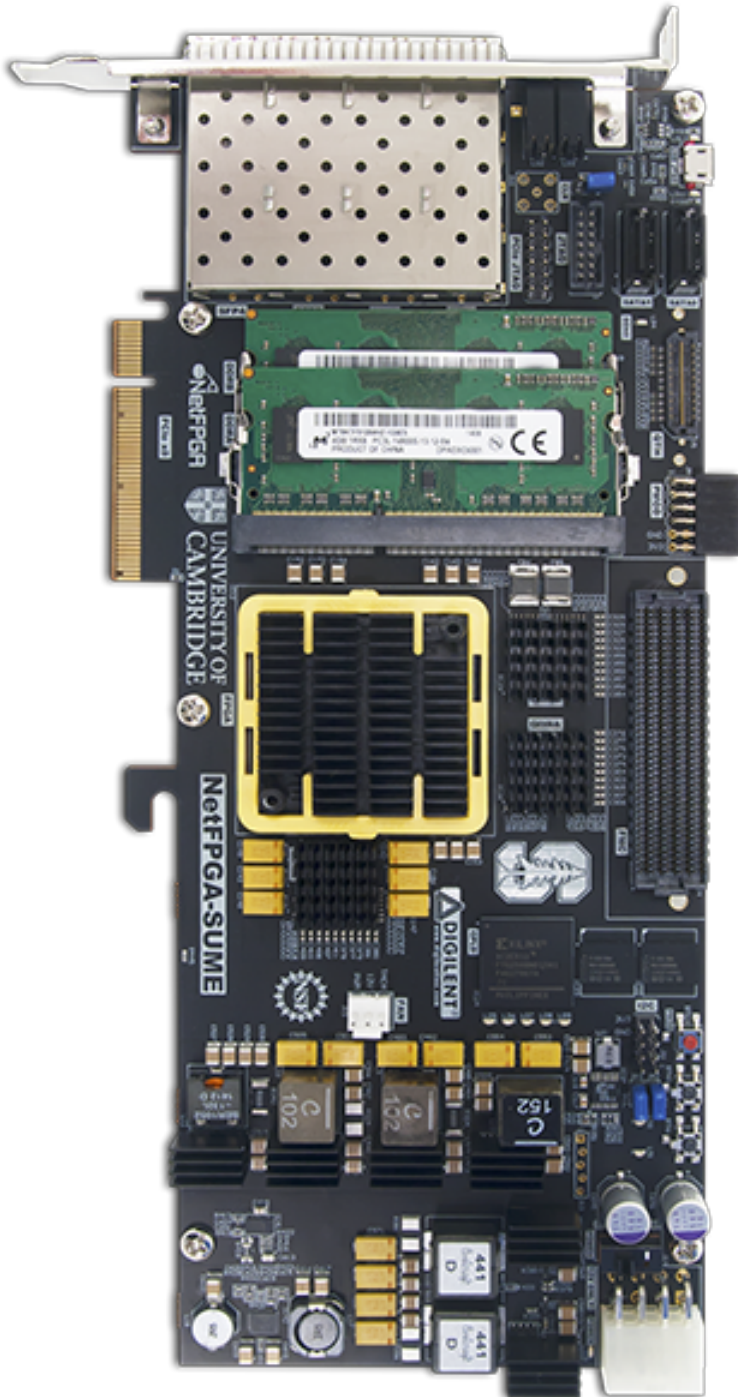


Figure 5.1: NetFPGA SUME

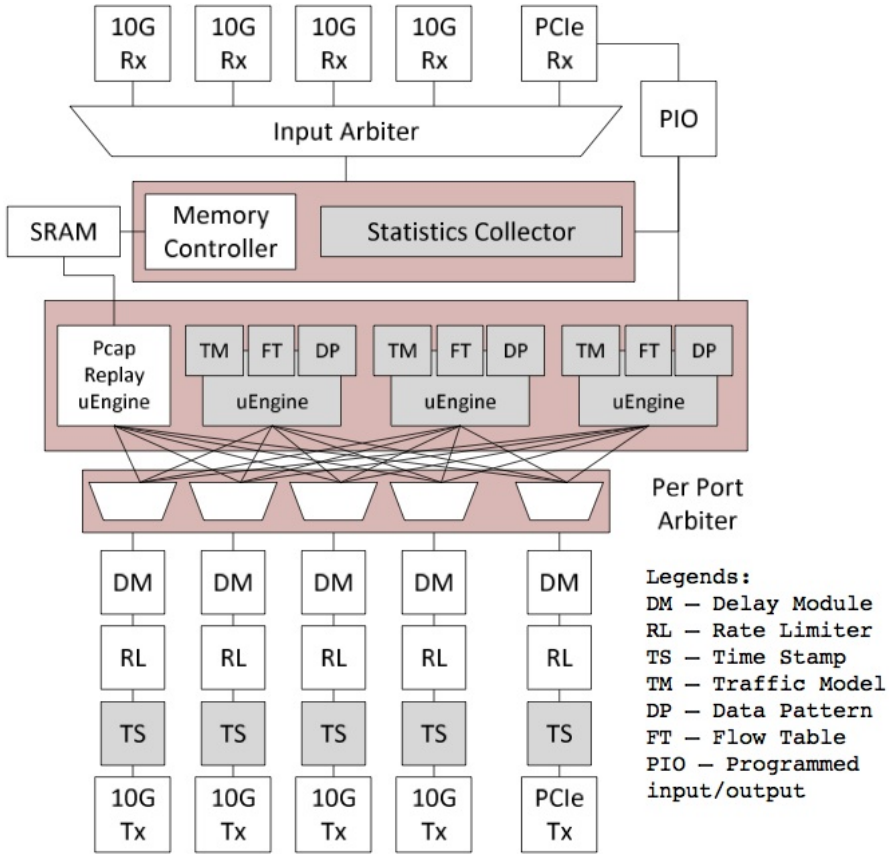


Figure 5.2: OSNT: architecture for generation

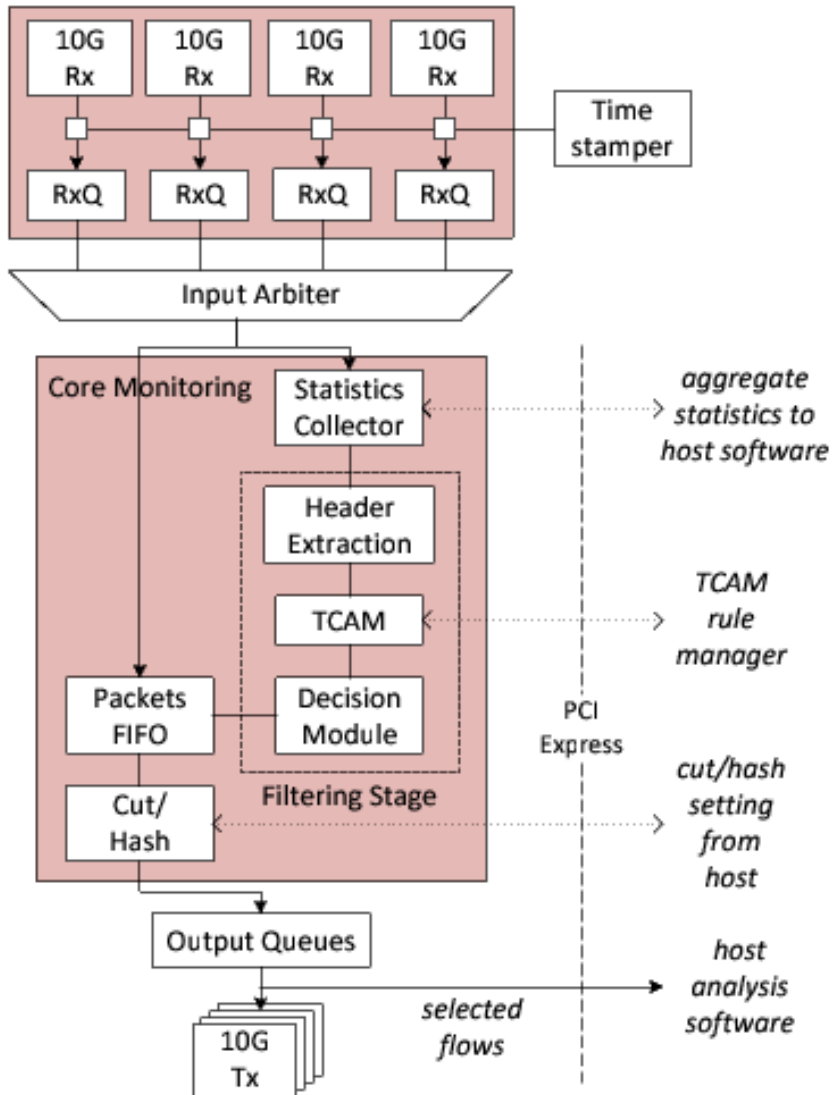


Figure 5.3: OSNT: architecture for monitoring

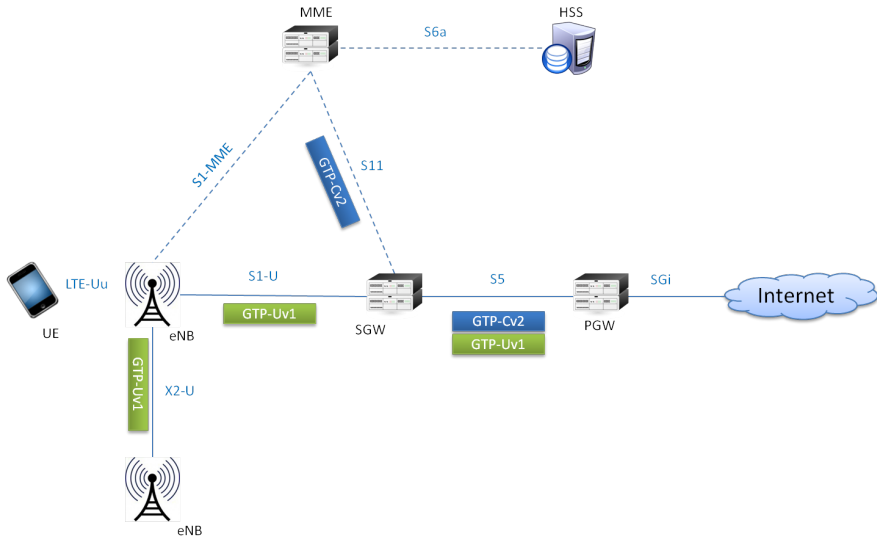


Figure 5.4: Traffic for eNodeB

Conclusions

The work in this thesis started from considering technologies and approaches that can contribute to solve open issues in the world of communications.

In particular, we focused on virtualization technologies, given their expansion and more and more common deployment.

Studying the characteristics of virtualization approaches, we were convinced to examine in depth their resources, the state of the art, and from there develop new ideas and design new solutions.

The work in this thesis focused on virtualization technologies, and how they can be deployed in order to provide concrete solutions, or leveraged to provide novel approaches.

Thanks to an interesting collaboration in the context of the project ARPEGGIO, we were able to work with a real VDI (Virtual Desktop Infrastructure) system, in order to collect real data on this technology.

The knowledge developed in this project, was a great addition when studying, designing, testing SDN-based solutions in Cloud Data Center environments, especially for understanding what kind of traffics are found in practice in these systems.

This allowed to tune our models when designing algorithms, and to create realistic traffic patterns when testing the system. In the spirit of obtaining practical results, a real testbed was built to create a downscaled version of a working Data Center, where to carry test.

Solutions based on SDN have been proposed and designed also for other environments. Thanks to a collaboration with TILAB, we had the possibility to get to testing activity on deployed LTE systems, and work on ideas to solve some issues that were emerging, especially concerning the lack of possibility for customization in measurement platforms.

These solutions, aimed at solving specific problems, have been tested in an ad-hoc testbed, built to offer a simulation of the envisioned system.

This work aimed at pointing out the many possibilities offered by virtualization techniques, at analyzing use cases when they can be deployed, and offering concrete solutions.

References

1. VMware, <http://www.vmware.com/>
2. XEN Cloud Platform, <http://www.xenproject.org/>
3. M. Al-Fares, A. Loukissas, A. Vahdat, "A scalable, commodity DC network architecture", Proc. of ACM SIGCOMM 2008, August 17–22, 2008, Seattle, Washington, USA.
4. A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: a scalable and flexible DC network," in Proc. ACM SIGCOMM 2009 conference on Data communication, ser. SIGCOMM '09, 2009, pp. 51–62.
5. R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: a scalable fault-tolerant layer 2 DC network fabric" in Proc. ACM SIGCOMM 2009 conference on Data communication, ser. SIGCOMM '09, 2009, pp. 39-50.
6. C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, S. Lu "DCell: A Scalable and Fault Tolerant Network Structure for DCs", in Proc. of the ACM SIGCOMM 2008 August 17–22, 2008, Seattle, Washington, USA.
7. K. Chen, C. Hu, X. Zhang, K. Zheng, Y. Chen, A. V. Vasilakos, "Survey on routing in DCs: insights and future directions," IEEE Network, vol. 25, no. 4, pp. 6–10, Jul. 2011.
8. M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridha, "DC TCP (DCTCP)", in Proc. of ACM SIGCOMM 2010, Aug. 30 – Sept. 3, 2010, New Delhi, India.
9. T. Benson, A. Anand, A. Akella, M. Zhang, "The case for fine-grained traffic engineering in DCs", Proc. of USENIX NSDI Workshop for Research on Enterprise Networks (WREN), San Jose, CA, Apr 2010.
10. M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat. "Hedera: Dynamic flow scheduling for DC networks", in Proc. of NSDI, San Jose, CA, April 2010. USENIX.
11. D. Verchere, "Cloud computing over telecom network," in Optical Fiber Communication Conf. (OFC), Los Angeles, CA, Mar. 2011
12. Y. Zhang, A.J. Su, G. Jiang, "Evaluating the impact of data center network architectures on application performance in virtualized environments," Quality of Service (IWQoS), 2010 18th International Workshop on, vol., no., pp.1-5, 16-18 June 2010.
13. B. Martini, M. Gharbaoui, P. Castoldi, "Cross-Functional Resource Orchestration in Optical Telco Clouds", 17th International Conference on Transparent Optical Networks (ICTON) 2015.
14. A. Lara, A. Kolasani, B. Ramamurthy, "Network Innovation using OpenFlow: A Survey", IEEE Communications Surveys & Tutorials, vol. 16, Issue: 1, 2014.
15. White paper ONF, "Software-defined networking: The new norm for networks" <https://www.opennetworking.org/images/stories/downloads/sdnresources/white-papers/wp-sdn-newnorm.pdf>," 2012

References

16. R.Wallner, R.Cannistra, "An SDN Approach: Quality of Service using Big Switch's Floodlight Open-source Controller", Proceedings of the Asia-Pacific Advanced Network 2013 Vol.35, p.14-19
17. K.Jeong, J.Kim, Y.T.Kim, "QoS-aware Network Operating System for Software Defined Networking with Generalized OpenFlows", 2012 IEEE/IFIP 4th Workshop on Management of the Future Internet (ManFI)
18. M.F.Bari, S.R.Chowdhury, R.Ahmed, R.Boutaba, "PolicyCop: An Autonomic QoS Policy Enforcement Framework for Software Defined Networks", Future Networks and Services (SDN4FNS), 2013
19. M. Davy, and et al, "A Case for Expanding OpenFlow/SDN Deployments On University Campuses," GENI Workshop Whitepaper, Jul, 2011.
20. T. Benson, A. Anand, A. Akella, M. Zhang "MicroTE: Fine grained Traffic Engineering for Data Centers", CoNEXT 2011, Tokyo, Japan.
21. R. Pries, M. Jarschel, S. Goll, "On the Usability of OpenFlow in Data Center Environments", Workshop on Clouds, Networks and Data Centers collocated with the IEEE International Conference on Communications (ICC 2012), Ottawa, Canada, June 2012.
22. L.Donatini, G.Foddiss, R.G.Garroppo, S.Giordano, G.Procissi, S.Roma, S.Topazzi, "Advances in LTE Network Monitoring: a Step Towards an SDN Solution", MELECON 2014
23. C.Callegari, L.Donatini, S.Giordano and M.Pagano "Improving Stability of PCA-based Network Anomaly Detection by means of Kernel-PCA" – International Journal of Computational Science and Engineering
24. D.Adami, L.Donatini, S.Giordano, M. Pagano, "Enabling QoS Routing for Traffic Engineering in Software Defined Networks" – IEEE ICC 2015
25. D.Adami, P.Castoldi, L.Donatini, M.Gharbaoui, S. Giordano, B.Martini, "Design and Evaluation of SDN-based Orchestration System for Cloud Data Centers" – IEEE ICC 2016
26. A.Ishimori, F.Farias, E.Cerqueira, A.Abelem, "Control of Multiple Packet Schedulers for Improving QoS on OpenFlow/SDN Networking", 2013 Second European Workshop on Software Defined Network
27. I.Bueno, J.I.Aznar, E.Escalona, J.Ferrer, J.A.García-Espín, "An OpenNaaS based SDN Framework for Dynamic QoS control", In Future Networks and Services (SDN4FNS), 2013 IEEE SDN for (pp. 1-7). IEEE.
28. Open Networking Foundation <https://www.opennetworking.org>
29. <https://www.vmware.com/it/products/horizon-view>
30. Introduction to VMware View Manager - VMware
31. VMware Capacity Planner, <http://www.vmware.com/products/capacity-planner/>
32. IBM Workload Deployer Home Page <http://www-01.ibm.com/software/webservers/cloudburst/>
33. S. Metha and A. NEogi, "Recon: a tool to recommend dynamic server consolidation in multi-cluster data centers," NOMS 2008.
34. J. Shahabuddin, A. Chrungoo, V. Gupta, S. Juneja, S. Kapoor, and A. Kumar, "Stream-packing: Resource allocation in web server farms with a QoS guarantee", HiPC, 2001.
35. Y. Ajiro and A. Tanaka, "Improving packing algorithms for server consolidation", in Proceedings of the International Conference for the Computer Measurement Group (CMG), 2007.
36. V. Jacobson, "Congestion avoidance and control", Proceedings of ACM SIGCOMM '88, pages 314–329, August 1988
37. S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The Nature of Datacenter Traffic: Measurements & Analysis", Proc. IFIP/IEEE International Symposium on Integrated Network Management (IM), 2009.
38. "SDN Analytics for Elephant Flow marking", Alcatel-Lucent Enterprise Application Note.
39. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, "Openflow: enabling innovation in campus networks", SIGCOMM Comput. Commun. Rev., vol. 38, pp. 69–74, March 2008.

40. B.Martini, D.Adami, A.Sgambelluri, M.Gharbaoui, L.Donatini, S.Giordano and P.Castoldi, "An SDN Orchestrator for Resources Chaining in Cloud Data Centers" -IEEE EuCNC 2014
41. A.Sgambelluri, D.Adami, L.Donatini, M.Gharbaoui, B.Martini, S.Giordano, P.Castoldi "An SDN orchestrator for Cloud Data Center networks: an experimental testbed" - IEEE NOMS 2014
42. D.Adami, C.Callegari, P.Castoldi, L.Donatini, M.Gharbaoui, S.Giordano, B.Martini, A.Sgambelluri, "Cloud and Network Service Orchestration in Software Defined Data Centers" – SPECTS 2015
43. <http://www.fp7-ofelia.eu/>
44. Open vSwitch <http://git.openvswitch.org>
45. Linux HTB <http://luxik.cdi.cz/devik/qos/htb>
46. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013–2018
47. A.Manzalini, R.Saracco, C.Buyukkoc, P.Chemouil, S.Kukliński, A.Gladisch, M.Fukui, W.Shen, E.Dekel, D.Soldani, M.Ulema, W.Cerroni, F.Callegati, G.Schembra, V.Riccobene, C.Mas Machuca, A.Galis, J.Mueller, "Software Defined Networks for Future Networks and Service", White Paper based on IEEE Workshop SDN4FNS 2013
48. White paper on "Network Functions Virtualisation"
http://portal.etsi.org/NFV/NFV_White_Paper.pdf
49. D.Adami, P.Castoldi, A.Del Chiaro, L.Donatini, M.Gharbaoui, S.Giordano, B.Martini, A.Sgambelluri, "An OpenFlow Controller for Cloud Data Centers: Experimental Setup and Validation", IEEE ICC 2014 - Communication QoS, Reliability and Modeling Symposium
50. D.Adami, L.Donatini, G.Foddis, S.Giordano, S.Roma, S.Topazzi, "Design and development of management functions for distributed monitoring based on SDN-based network" - IEEE EMTC 2014
51. White Paper: Protocol Signaling Procedures in LTE
52. Mininet: <http://mininet.org/>
53. Mininet <http://github.com/mininet/mininet/wiki/>
54. <https://www.virtualbox.org/>
55. <http://www.github.com/noxrepo/pox>
56. G.Antichi, L.Donatini, R.G.Garroppo, S.Giordano, A.W.Moore, "An Open-Source Hardware Approach for High Performance Low-Cost QoS Monitoring of VoIP Traffic" - Mathematical and Engineering Methods in Computer Science, LNCS volume 8934, pp 1-15
57. N. Zilberman, Y. Audzevich, G.A. Covington, A.W. Moore, "NetFPGA SUME: Toward 100 Gbps as Research Commodity", IEEE Micro, vol.34, no.5, pp.32-41, September-October 2014
58. NetFPGA: <http://netfpga.org/>
59. NetFPGA projects: <https://github.com/NetFPGA/NetFPGA-public/wiki/Projects>
60. G. Antichi, M. Shahbaz, Y. Geng, N. Zilberman, A. Covington, M. Bruyere, N. Feamster, N. McKeown, B. Felderman, M. Blott, A.W. Moore, P. Owezarski, "OSNT: Open Source Network Tester", IEEE Network Magazine, Special issue on Open Source for Networking: Tools and Applications, 2014

Acknowledgments

First and foremost, I want to thank my family: mom and dad, for being there, for all the support, the encouragement. For all the love, for making me feel safe, for making me really feel I'm never alone. And thank you for my brother Alessandro, the bestest brother and friend anyone could wish for. Thank you for all the times you listen to me, for being someone I can always turn to, for being always on my side.

Thank you to my grandparents, wherever you are, for all the times you played with me, for all the times you prayed for me. For the precious time you spent with me and all that you taught me.

For the work in this thesis, the biggest thank you goes to my supervisors: Prof. Stefano Giordano and Ing. Davide Adami, for guidance and tutoring through my PhD.

Huge thank you to all my colleagues in Pisa for making these years memorable. Christian, thank you and sorry for all our conversations. Gregorio for all the advices. Rosario for the additional guidance provided. Simone for sharing these PhD years. And thank you to all people I've met in these years working here (Michele, Luca, Nicola, ...), from all I have learned something.

Thank you to Telecom: to Simone, to Gianluca, to everybody in TILAB. And to all the people I had the chance to meet in Torino: even if my stay was quite brief, it had a huge impact on me, it is something I will forever cherish.

Thank you to Prof. Andrew Moore for having me as visitor in Cambridge: having the chance to work in such environment was an honour. It was great to be part of that team. Thank you Gianni for... where do I even start? For everything and more. I feel grateful for all the wonderful people I met in the stunning city by the Cam. For the friendships, the lessons, the joy this experience brought me: the months I spent there will always have a most special place in my heart.

Thank you Poiane for years of friendship, of laughs, of support. Of adventures that are providing a lifelong reserve of anecdotes to tell... or to never tell.

Huge thanks to all my friends near and far, irl and online: thank you, from the bottom of my heart, for each and every hug, for each and every text, each and every coffee,

each and every call, each and every chat: for all these little things that are not little at all, whether they seem or not, I truly thank you all from the bottom of my heart.

Thank you Poets Of The Fall for keeping my peace, for forcing me to remember about colours when all looked black and grey.

Thank you Imagination Technologies for believing in me and for giving me this chance.

To everybody mentioned here, and to those not mentioned as well: thank you for walking some of this path with me.