



UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Facoltà di Ingegneria

Dottorato di Ricerca in Ingegneria Informatica ed Automatica

XXV Ciclo

Dipartimento di Informatica e Sistemistica

THE PATH TOWARDS AN INTERNET'S SON:  
SERVICE ORIENTED NETWORKING

ROBERTO BIFULCO

Ph.D. Thesis

TUTOR

Prof. Roberto Canonico

COORDINATOR

Prof. Francesco Garofalo

March 2013

# Abstract

The work described in this Ph.D. thesis deals with the evolution of the current Internet architecture towards a communication model suited for dealing with services rather than with nodes and processes. The work introduces some practical use cases to describe the current “as a service” orientation of many network applications, and the required new degrees of flexibility in the resources management and deployment. Several solutions to this aim are designed, implemented and evaluated when integrated in the legacy network infrastructure. Furthermore, the thesis explores the concepts of network virtualization and software defined networking, both in practical and theoretical aspects, applying them to the concrete design of a networking architecture that enables the split of network identifier and locator concepts. The proposed architecture is a first step towards an implementation of a service oriented architecture on top of the current Internet. At the same time, the thesis provides an example on how network protocols can be used in a new way without changing their dynamics, or using a term familiar to software engineer, how they can be “overloaded”, to create new functions provided that the network control plane is correctly designed to handle them.

# Contents

<b>Introduction</b>	<b>1</b>
<b>1 Everything as a Service</b>	<b>8</b>
1.1 Grid as a Service . . . . .	9
1.1.1 Context and motivations . . . . .	9
1.1.2 Grid Computing reference architecture . . . . .	11
1.1.3 Related works . . . . .	12
1.1.4 GaaS: Grid as a Service . . . . .	14
1.1.5 The SCoPE case study . . . . .	17
1.1.6 Outcomes and Future Work . . . . .	17
1.2 Testbed as a Service . . . . .	18
1.2.1 Context and motivations . . . . .	18
1.2.2 OMF architecture . . . . .	19
1.2.3 NEPTUNE . . . . .	20
1.2.4 Automatic deployment of a OMF virtual testbed . . . . .	23
<b>2 Providing flexible infrastructures</b>	<b>25</b>
2.1 Virtual links . . . . .	25
2.1.1 Context and motivation . . . . .	25
2.1.2 Virtualization in Network Emulation Systems . . . . .	27
2.1.3 Experimental Evaluation . . . . .	32
2.1.4 Conclusions . . . . .	37
2.2 Fast provisioning of virtual nodes . . . . .	38
2.3 Resource allocation problem . . . . .	40
<b>3 Introducing mobility in virtual infrastructures</b>	<b>45</b>
3.1 Local mobility . . . . .	46
3.1.1 Context and motivation . . . . .	46
3.1.2 Datacenter networking . . . . .	48
3.1.3 Virtualization . . . . .	49
3.1.4 Our solution . . . . .	50

---

3.1.5	Evaluation . . . . .	56
3.1.6	Related work . . . . .	59
3.1.7	Conclusion . . . . .	60
3.2	Global mobility . . . . .	60
3.2.1	Context and motivation . . . . .	60
3.2.2	NEPTUNE-IaaS . . . . .	61
3.2.3	The Service Switching Paradigm . . . . .	64
3.2.4	Related Work and Conclusions . . . . .	68
<b>4</b>	<b>Issues in managing flexible resources</b>	<b>70</b>
4.1	Security Issues . . . . .	70
4.1.1	Context and motivation . . . . .	70
4.1.2	Cloud computing . . . . .	72
4.1.3	The Eucalyptus Cloud Computing System . . . . .	73
4.1.4	Detecting Attacks in a Cloud Computing System . . . . .	75
4.1.5	Experimental evaluation . . . . .	79
4.1.6	Related works and Conclusions . . . . .	83
4.2	Resources access management . . . . .	84
4.2.1	Context and motivation . . . . .	84
4.2.2	PlanetLab: architecture, usage model and resource management . . . . .	88
4.2.3	OMF: architecture, usage model and resource management . . . . .	91
4.2.4	PlanetLab and OMF integration . . . . .	100
4.2.5	Experimental setup . . . . .	107
4.2.6	Experiments . . . . .	108
4.2.7	Related work . . . . .	113
4.2.8	Conclusions . . . . .	115
<b>5</b>	<b>Programmable networks</b>	<b>116</b>
5.1	SDN's control plane scalability . . . . .	117
5.1.1	Introduction . . . . .	117
5.1.2	OpenFlow . . . . .	119
5.1.3	OpenFlow Scalability . . . . .	121
5.1.4	Control logic dimensions . . . . .	125
5.1.5	Scalability solutions . . . . .	126
5.1.6	Conclusions . . . . .	130
5.2	Network programming . . . . .	131
5.2.1	Introduction . . . . .	131
5.2.2	Related work . . . . .	132
5.2.3	Rule-based programming . . . . .	133

---

5.2.4	Interactions definition . . . . .	134
5.2.5	Interactions detection . . . . .	138
5.2.6	Use cases . . . . .	141
5.2.7	Conclusions and future work . . . . .	144
<b>6</b>	<b>Towards service oriented networks</b>	<b>145</b>
6.1	Follow-Me Cloud . . . . .	146
6.1.1	Introduction . . . . .	146
6.1.2	Related work . . . . .	148
6.1.3	End-Points Mobility . . . . .	149
6.1.4	Scalability . . . . .	151
6.1.5	Distributed Controller . . . . .	153
6.1.6	Evaluation . . . . .	157
6.1.7	Conclusions and future work . . . . .	159
6.2	Implementing FMC . . . . .	160
6.2.1	Introduction . . . . .	160
6.2.2	Follow-Me Cloud . . . . .	161
6.2.3	Controller design . . . . .	162
6.2.4	Discussion . . . . .	169
6.2.5	Related Work . . . . .	171
6.2.6	Conclusions . . . . .	173
6.3	Handover handling . . . . .	173
6.3.1	Introduction . . . . .	173
6.3.2	Related Work . . . . .	173
6.3.3	Follow-Me Cloud . . . . .	175
6.3.4	Handover analysis . . . . .	177
6.3.5	Discussion and Conclusions . . . . .	179
	<b>Conclusions</b>	<b>182</b>

# Introduction

*“Ignoranti quem portum petat nullus suus ventus est”*[133]. The aim of technology is to help the human life in everyday activities, providing the basis for the next enhancement in the life-style and enabling unforeseen possibilities. In the iterative process of building technologies to support ideas, gathered from the features provided by new technologies, it usually happens that a technology is used in way it was not designed for, to temporary fulfill the emerging needs while waiting for the deployment of a new technology, tailored for the purpose. The technological evolution is, hence, an inherently iterative process in which a technology builds on the outcome of a previously developed technology, sometimes precisely using the existing technology, sometimes forcing the existing technology in working in an unoptimized way to provide new unforeseen features. While this process is usually highly desirable to foster fast technological evolution, it can be, on the other hand, a limitation as well. In some cases, the link among technologies is so tight that the change of a well-established and wide-spread technology would cost enough to discourage the change of such technology toward a more promising one. Understanding the point in which a technology turns from an enabling factor to a limiting one is usually a hard task: the legacy technology is still providing the required features, while the new one, although promising, has no chance to be proven as effective without affording the risks and costs of a major change. A great example in this sense is the Internet, that is a collection of well established technologies on top of which several other technologies are relying on. Considering the impressive boost the Internet gave to many fields, providing an high number of improvements and new possibil-

ities in a short time-span, such as the Web, video, audio streaming services, converged telephony, data and video services, just to cite some of them, it sounds somewhat surprisingly that all these innovative applications are supported by a substantially unchanged set of core technologies. The core of the Internet protocol suite, that is constituted by the protocols of the network and transport layers, was designed in the early 1980s: the Internet Protocol (IP)[119] and the Transmission Control Protocol (TCP)[120] were both defined as standards in September 1981 and they are still used mainly unchanged. Despite the clear success of the original design, many of the actual applications were just unforeseen in the original architecture, which, ignoring them, is unable to provide proper technological support. Hence, the ability to support new applications was provided mainly by the addition of new protocols that on one hand solve the limitations of the architecture, while on the other hand they introduce some complexity and/or force the design to work in an unexpected way. An example of such an approach is the Mobile IP (MIP) protocol[111], which was designed to introduce the support for mobile end-points in an IP network, and which requires the use of inefficient “triangular routing” to provide mobility overcoming the inherited limitations of IP. Content Delivery Networks[109] are another example of a technological workaround used to provide a service, which the current Internet architecture is not able to support, fulfilling all the requirements in terms of cost and efficiency: to provide users with the requested content, a set of content replicas is distributed in several locations, and the location that serves a particular user requesting that content is decided using a strategy based, e.g., on the requesting user location, using protocols that were born with other purposes, like DNS, to gather indirectly the required user’s location information.

With the evolution of the network applications, asking for more and more complex features, over the years, a strong interest growth in developing a new Internet architecture able to support emerging applications. There are several proposals for network architectures based on new paradigms and abstractions, such as Content-Centric Networking [65] or Service-Centric Net-

working [26] [98]. Nevertheless, the scientific and industrial communities involved in the Internet enhancement had to recognize that supporting the evolution in the Internet is becoming more challenging over time, because of the difficulty in combining the need to keep a critical infrastructure reliable, as the Internet became in past decades, with the need of making such infrastructure able to support new requirements in an efficient way. This problem, also known as the Internet “ossification” problem, is mainly related to the ability of supporting changing requirements instead of a set of already clear requirements mandated by applications. The current Internet, despite its flexibility, is considered unable to efficiently support evolution, hence, the scientific community recently proposed a *pluralist* approach to the network architecture[11]. In this approach, the network is able to support a plurality of architectures in a concurrent manner. The main motivation for the *pluralist* theory comes from the observation that the Internet is already supporting an high number of heterogeneous applications, with a broad set of requirements. It’s rather intuitive that a telephony application like a Voice-over-IP application has different requirements from a file-sharing one. Given this heterogeneity of requirements it’s hard to design a one-size-fits-all architecture. One of the aim of the pluralist approach is to avoid that a new ossification affects the new Internet architecture. Hence, an architecture that is able to support flexibility and extensibility is actually considered the best solution for the future Internet. The outcome of these research efforts is that the network view is changing, since the network itself is seen as something that can be flexibly configured to support different architectures[47], provided that some basic interface are well-defined[73], to allow interoperability in a world composed by several heterogeneous networks. Unsurprisingly, a strong interest in this direction was firstly shown by the community involved into network testbed designs, whose purpose is to enable the share of a network testbed among several experiments, enabling at the same time each experiment to provide its, potentially novel, own network architecture and protocols.



To this aim, the effort of the scientific community in defining new Internet architectures splits among *substrate* architectures and *application oriented* architectures, where the former ones are used to host the latter ones.

A first step into the direction of supporting several network architectures over the same substrate has been network virtualization. Network virtualization architectures [35] provide a mean to virtualize network links and nodes, by enabling the creation of virtual links and nodes on top of the physical ones. Virtualized components can be used to create virtual networks, that can implement their own architectures and protocol stacks. The main difference with previous network slicing approaches, such as the ones based on light paths provisioning, is the dynamic creation and management of the virtual networks, and the higher level of abstraction, e.g., a virtual link can be actually mapped onto a physical path.

Together with the efforts in defining network virtualization infrastructures, a new research area for providing network flexibility has been recently explored: Software Defined Networking (SDN)[104]. SDN is a new approach that tries to bring software applications in a world actually dominated by hardware applications. To this aim, SDN provides an architectural view in which the control and data planes of the network are separated, in order to make their evolution simpler and decoupled from each other, moreover, this way the network behaviour can be defined through software applications that are easier to extend and evolve, which, being decoupled by the limitations of hardware devices, can provide powerful control logics to be hosted on (clusters of) general purpose servers.

While in network virtualization a substrate network is able to host several independent virtual networks, in SDN an open interface provides the ability to support custom control logics on network devices. In both cases, the most important provided feature is the ability to roll-back changes in a fast and simple way, overcoming the limitation in supporting evolution of the current internet: the ability to roll-back changes ensures limited impact on costs of the new technology, since it is always possible to re-establish the

proven legacy technologies. The combination of the two approaches has been quite natural considering the context and the purposes in which they were developed: network virtualization is one of the methods applicable by means of SDN to provide the co-existence of different network architectures over the same substrate. Network virtualization is hence a feature, or from a perspective an application, that runs on top of a network whose management is performed through an SDN approach. Unsurprisingly, one of the way of implementing network virtualization in a SDN network corresponds in enhancing the abstraction level of the network, by running the control logics for virtualized networks on top of a low-level control logic that takes care of sharing the substrate network among the virtual networks[134].

The network as shaped by both network virtualization and in particular SDN, is actually not providing a set of specific features anymore, but it is able to adapt to several purposes its operations. Like in the past happened with special purpose hardware that evolved into nowadays general purpose computers, the network is moving towards a general purpose network: the same network can be reprogrammed to enhance/change the provided services.

Given the new tools and methods we can use to provide network services, some questions still need an answer to validate the suitability of the approach to non trivial applications, e.g., can we evolve from the current network to an SDN one transparently? How can we handle already deployed technology and protocols? In this work we try to answer these questions and others, with the aim of understanding advantages, issues, problems in several fields, when applying network virtualization and SDN approaches in a traditional network, in order to change the way in which the the network operates to provide a networking paradigm closer to the semantic of services rather than to the one of hosts as it used to happen in Internet.

This thesis summarizes the outcomes of several works related to the aforementioned topics, targeting at several specific research problems that need to be solved in order to provide a service oriented network architecture on top of a traditional TCP/IP network enhanced with SDN devices. The thesis is

organized as follows:

- **Chapter 1** - Describes trends in offering resources and applications as a service, analyzing two practical examples: a system for providing GRID infrastructure and a system for providing Network Testbeds. The aim of the chapter is to introduce the different approach in accessing resources when the provisioning systems uses a “service model”.
- **Chapter 2** - In this chapter the problem of making traditional network infrastructure flexible by means of virtualization is described, and solutions for both nodes and links virtualization are presented. The advantages of contextualizing the virtualization solution to specific applications are explored, as well as problems and algorithms to optimize the management of virtualized resources and their allocation on the physical infrastructure.
- **Chapter 3** - The decoupling from the physical infrastructure provided by virtualization enables the mobility of virtual resources. Since the operations are still performed relying on the traditional TCP/IP network architecture, solutions to enable seamless and transparent mobility both in local and geographical boundaries are required. This chapter presents solutions for the two cases, relying on the TCP/IP network architecture and provides a reference comparison point for the next chapters, in which similar features are provided by means of an SDN devices enhanced network.
- **Chapter 4** - This chapter analyzes and provides solution to the problems of managing access to flexible resources and monitoring them for security purposes. The studies are conducted in already in-production environment such as a distributed network testbed and a cloud computing system.
- **Chapter 5** - When the network uses the SDN paradigm, new issues arises. In this chapter we provide an extensive study on the scalability

issues and viable solutions for the SDN control plane design. Moreover, exploiting the software orientation of the network control logic, we introduce an algorithm for analyzing/debug and optimize the operations in an SDN network.

- **Chapter 6** - In the last chapter we show how a traditional network enhanced with SDN devices can provide a service-oriented networking paradigm, overloading the meaning of the protocol headers fields used in the legacy TCP/IP network. The designed architecture is analyzed both in terms of scalability and performance, the analysis results, the design decisions and the implementation solutions are included in this chapter.

# Chapter 1

## Everything as a Service

In the beginning, the Internet was built to solve the problem of establishing a communication channel between two hosts, or, more precisely, between two processes distributed over the network. After years of evolution, a common Internet user is currently rarely accessing a given host or process: the user is actually asking for a service. The service can be the view of a video stream, the provisioning of a given web page, the access to the bank account management, and so on. In 2006 Amazon started a service called Elastic Compute Cloud[6] in which even an IT infrastructure was provided as a service. To identify this trend of providing everything as a service, the research community uses the umbrella term *Cloud Computing*. Cloud Computing assumes different names depending on the provided resources. When provided resources are computing nodes and storage, Cloud Computing is called Infrastructure as a Service (IaaS). Other resources types include application frameworks (Platform as a Service, PaaS in short) and specific software applications (Software as a Service, SaaS). The different embodiments of Cloud Computing target different customers as well. For instance, while IaaS and PaaS target service providers, developers and engineers, SaaS is targeted to a broader audience, since the offered services are mainly specialized applications, built to address one purpose. In this chapter we describe the work done in providing in a “as a service” manner a Grid Environment and a Network Testbed, describing both the usage model and the underlying sys-

tems design and issues. Even if the presented case studies refer to particular fields, the theoretical and technological finding can be extended to more general applications, in particular in the fields of IaaS and PaaS provisioning systems. This chapter shows, presenting two practical use cases, the possibility of providing even complex infrastructures, such as GRID and network testbed ones, “as a service”, abstracting the way users access the given infrastructure. At the same time, the network that connects users to these infrastructures is still based on a process-based model, even if such a model is completely different from the one used by the presented systems, highlighting the gap in the current network architecture in properly supporting systems based. for instance, on service abstractions.

## 1.1 Grid as a Service

### 1.1.1 Context and motivations

On-demand computing is a model in which computing resources are made available to users as needed. It could be considered a valid solution for people who need a huge amount of resources, to reduce the *Total Time to Solution*, and cannot bear the costs of systems. In particular, these costs grow up when the needed resources are provided by specialized systems, e.g., HPC ones.

The scientific community developed the Grid Computing paradigm to enable the sharing of huge amount of resources through a well-defined distributed infrastructure model, in order to solve large scale problems in a collaborative manner. The Grid Computing resources aggregation model is rather “static”: a group of organizations set up several Grid management services and computing resources in a layered structure that separates the management responsibilities (and corresponding management services) among the organizations involved in the Grid.

Users belonging to the organizations forming the Grid can retrieve information on resources (e.g., their number, status, configuration, etc.) and

access them, but can neither change the topology of the grid (e.g., by increasing the number of resources) or manage resources configuration and composition. It would be desirable to have a more “*elastic*” infrastructure in which users can ask for resources on-demand, to suit their needs in terms of resources type and configuration (i.e. compilers, scientific libraries, problem solving environments, etc.).

With the advent of new applications and the pervasiveness of IT into everyday activities, also the industrial and private sectors have developed a need for fast access to high demanding IT infrastructures at low costs. The industry answer to these needs has been the Cloud Computing model.

According to the official NIST definition, “*Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources ... that can be rapidly provisioned and released with minimal management effort or service provider interaction*” [90].

Given the flexibility in resources management through the Cloud Computing paradigm, it seems a promising approach to provide flexible Grid Computing infrastructures through the combination of the Grid and Cloud paradigms.

Some interest has been already shown in this direction [91]. So far, the proposed approaches may be labeled as either “Grid over Cloud” or “Cloud over Grid”, since the composition of the two paradigms may be performed through either the exploitation of IaaS-provided resources to build Grid infrastructures or through the use of Grid-provided computing resources to create IaaS clouds.

In this section we describe our experience in designing and implementing a solution that creates more flexible Grid infrastructures, by exploiting IaaS-provided resources, in a novel way resembling the PaaS paradigm. We call our solution Grid as a Service (GaaS).

### 1.1.2 Grid Computing reference architecture

We assume as Grid reference architecture the one implemented by the middleware gLite-EMI, developed in the context of EGI (European Grid Infrastructure). The gLite-EMI middleware provides a Grid infrastructure that is accessible to community members organized into Virtual Organizations (VO). A VO is defined in [48], as *“a set of individuals and/or institutions defined by such sharing rules...”*. *“VOs vary tremendously in their purpose, scope, size, duration, structure, community and sociology”*. In particular, people from a scientific community, sharing the same “experiment”/applications, can constitute a VO.

VO “managers” make available all the software needed to run the applications of interest of the community on computing resources (the applicative level of the middleware).

The Grid infrastructure is a distributed infrastructure whose management is centralized, while the computation functions are distributed among several sites. The infrastructure provides users with high level services for scheduling and running computational jobs, accessing and moving data, and obtaining information on the infrastructure itself. Services are embedded into a consistent security framework [76]. Those provided are services for authentication/authorization (e.g. VOMS - Virtual Organization Management System), resources allocation and discovery (e.g. LB/WMS - Logging & Booking and Wokload Management System), infrastructure Information System (IS). Computing resources (WNs - Worker Nodes) are provided by means of CE (Computing Element) that is an endpoint with a set of queues handled by an LRMS (Local Resource Management System). User can access these services from a User Interface (UI).

Management services (UI, VOMS, WMS) are instantiated only once and shared among all the sites, while computing-related services (IS, CE) are replicated in each site. A graphical representation of a minimal gLite-based Grid infrastructure is presented in figure 1.1.

To use the Grid infrastructure, the user has to (1) authenticate himself



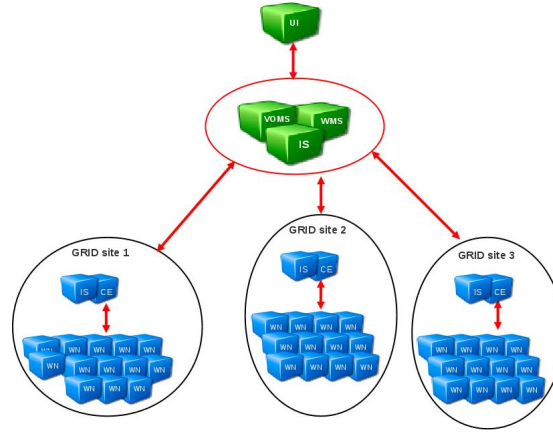


Figure 1.1: A gLite-based infrastructure with some central services (UI, VOMS, WMS, IS) and three grid sites (with a CE and several WN).

on the infrastructure; (2) define a *job* in terms of resources requirements and tasks to be performed; (3) submit the job to the infrastructure by selecting the appropriate resource queue; (4) monitor the job status; (5) retrieve the job execution results. The *resource queue* is an abstraction provided by the Grid architecture to either group resources based on their properties and to share such resources among several users.

The typical Grid usage model described so far does not allow the user in customizing the grid environment. Users cannot change the Grid infrastructure that runs their experiment, in particular, a user cannot create a new Grid site nor add an existing site for his VO. Also the Grid sites are static and cannot be customized by users, hence, it is not possible to add new worker nodes to a site to extend its capabilities, and it is also not allowed to organize resources into customized queues to shape them in accordance to the computation needs. Finally, even the configuration of the worker nodes cannot be changed.

### 1.1.3 Related works

Related works about the Cloud and Grid integration can be classified either as “Cloud-over-Grid” or “Grid-over-Cloud” approaches.

An example implementation of the “*Cloud-over-Grid*” is presented in

[143], where CLEVER, a cloud management system, is used to provide an IaaS system over Grid. The solution requires the installation of both a specialized CLEVER's management software and a virtual machine monitor (e.g. VirtualBox) into Grid worker nodes. When the CLEVER cloud requires more resources, new worker nodes can be assigned to it, to dynamically extend the resources available to the cloud.

WNoDeS [131] applies a "*Cloud-over-Grid*" approach as well. WNoDeS (Worker Nodes on Demand Service), developed by the Italian National Institute for Nuclear Physics (INFN), is a solution to virtualize computing resources and to make them available through local, Grid or Cloud interfaces. The Grid infrastructure is exploited through the use of a "*special*" gLite job: the "*power on*". Users define "Power on" jobs selecting tailored virtual machine images to be launched on computational resources managed by the CE.

In [34] is described an example of "Grid-over-Cloud", that transparently provides dynamically-instantiated VM-based worker nodes, in an EGEE production grid.

StratusLab [80] is applying a "Grid-over-Cloud" approach as well. The StratusLab project aims at developing a complete, open-source cloud distribution that can be deployed in production in both academic and industrial environments. StratusLab provides Grid services using StratusLabs IaaS system resources. The provided Grid infrastructure can exploit the dynamic nature of the cloud, provisioning resources as needed and running user-level (and community level) services using pre-packaged appliances, selected by users and made available by a "*marketplace*".

In [15] we presented the design and implementation of an on demand computing service, which is able to obtain a right trade-off among management cost reduction, environmental sustainability and user satisfaction. In particular, the work described an experience in designing and implementing a flexible infrastructure, built on the basis of local or remote cloud resources, with the aim of saving energy to reduce the overall operational cost and to

improve environmental sustainability.

Recently, also commercial Cloud Providers are trying to explore the HPC market, by providing resources for, e.g., scientific computations. Most notably, Amazon, is offering “cluster compute” instances, through its Elastic Compute Cloud service, whose resources are tailored for HPC, i.e., they are provided with huge amounts of RAM, processing power, and are deployed on a 10 Gigabit Ethernet network with low delay[8].

To ease the execution of specific workloads, some tools provide automatic configuration of Amazon resources. An example in such sense is CloudFlu[39], that allows the easy execution of OpenFOAM[103] jobs on an automatically configured cluster of Amazon EC2 HPC resources.

#### 1.1.4 GaaS: Grid as a Service

In 2012, the European Middleware Initiative (EMI, <http://www.eu-emi.eu>) has published a report in which they describe four possible integration scenarios of virtualized infrastructures in the Grid computing architecture [91]. In this paper, we present *Grid-as-a-Service* (GaaS), a service model designed according to the *Dynamic Grid Services* scenario described in that report:

*[Dynamic Grid Services] utilizes the cloud infrastructure to provision grid services using IaaS/PaaS/SaaS models. The grid services, or suitable subsets of the current grid services, can therefore be instantiated on demand ... by deploying and configuring the services on base virtual machines according to specific user community requirements and then disposed of when not needed anymore.*

The GaaS model combines the advantage of providing users with an usage model that is familiar to the traditional Grid, with the possibility of flexible management of computational resources in a IaaS-like manner. Hence, our model can be classified as a Platform-as-a-Service for extending Grid

environments with elastic (e.g., virtual) resources. By using GaaS, “privileged” grid users, e.g. the VO administrator, can define new Grid Sites, add computational resources to existing Grid Sites and modify the resources aggregation scheme, e.g., site queues. In particular, GaaS provides privileged users with the following functions:

1. WNs management (fig. 1.2.a): definition, addition and deletion of WN to be used by the Grid infrastructure;
2. Queues management (fig. 1.2.b): dynamic management of resources in queues, and queues policies configuration;
3. Sites management (fig. 1.2.c): creation and management of new Grid Sites;

GaaS flexibility provides several advantages to traditional Grid infrastructures, e.g., WNs can be customized with software tailored to a given set of users, as well as queues can be configured to fulfill a specific computation needs. Moreover, GaaS support the creation of complete Grid sites in order to, e.g., enable a community that has to share resources for the life time of a project, to avoid the burden of configuring from scratch all the required services and resources. Computational resources can be both virtual and physical. The use of virtual resources is not denied to HPC users but, if virtual resources are chosen, users are notified by IS, about the possible performance limitations.

Even if our approach is based on a Grid-over-Cloud model similar to the StratusLab one, in GaaS resources are made available through their configuration in Grid abstractions, e.g., queues. Hence, provided resources can be reconfigured or differently aggregated on the basis of users needs (in a way resembling the PaaS paradigm). Moreover, GaaS enables the provisioning of several high level functions: from queues creation/reconfiguration to the instantiation and configuration of whole grid sites.

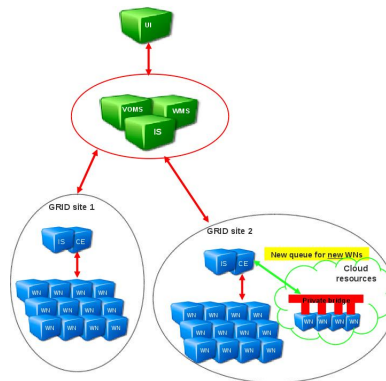
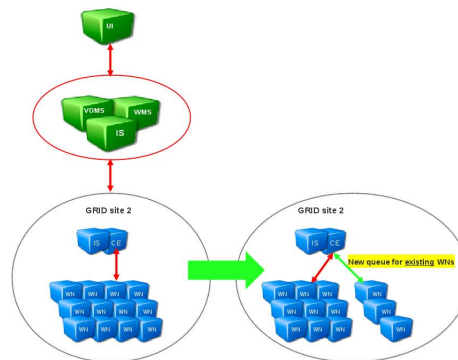
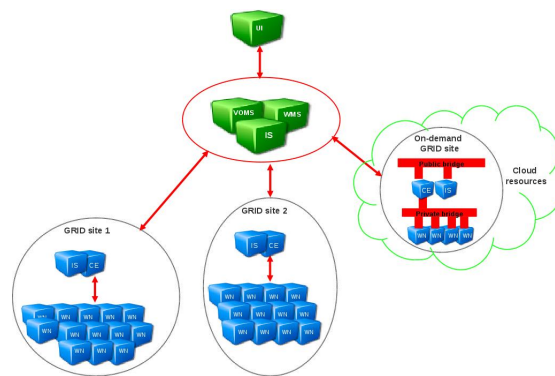
(a) *WNs management*(b) *Queues management*(c) *Site management*

Figure 1.2: The GaaS solution.

### 1.1.5 The SCoPE case study

For the validation of the proposed model we implemented a prototype and integrated it into the context of the S.Co.P.E. Datacenter at University of Naples Federico II, a self contained grid infrastructure that offers storage and computational resources and all the high level core services for infrastructure management (VOMS, WMS, IS, etc.). Moreover S.Co.P.E. resources are integrated also into national IGI and international EGI relevant distributed computational infrastructures and used from people belonging to different scientific reserach fields and to VOs from very rilevant international experiments (e.g. LHC, ATLAS, Super-B, etc.). Thus S.Co.P.E. is a suitable context to validate effectively our approach by means of a prototype.

Our prototype is based on the gLite-EMI [76] Grid middleware and on the OpenNebula [12] cloud management system. The modularity of OpenNebula allows for fast introduction of new features to the management system, hence, it allows the easy integration of the Cloud-provided resources into the Grid infrastructure. A subset of the S.Co.P.E. resources are assigned to the OpenNebula managed resources pool. Such resources host an hypervisor, currently Xen [14], to create virtual machines (VM), that are then used as dynamically provided resources for the Grid infrastructure. VMs are used to both create Grid's WNs and management services such as CE, IS, etc.

The main efforts in the prototype development were (i) the definition of templates for gLite-EMI services configuration, and (ii) the enabling of their fast provisioning. A detailed description and an evaluation of the designed solution is presented in chapter 2.

### 1.1.6 Outcomes and Future Work

In this section we presented GaaS, a PaaS model for Grid Computing systems, that lets VO administrartors to dynamically customize the grid environment they are offering to VO's unprivileged members. VO administrators can define new Grid Sites, add computational resources to Grid Sites and modify the resources aggregation scheme (queues). A prototype of GaaS

model has been implemented and deployed in a real-world Grid Datacenter.

Even if the presented work is a successful proof-of-concept, many issues still have to be solved. In particular there is a need to assess the applicability of virtualized resources in HPC contexts, the payed overhead, and the possibility to extend the model to a mix of virtualized and physical resources according to the users needs.

Moreover, an evaluation of the impact on management operations and costs of GaaS approach is needed as well, in order to integrate a smart management of resources with the aim of providing, e.g., energy savings.

## 1.2 Testbed as a Service

### 1.2.1 Context and motivations

In the last few years, network emulation has gained interest in the community of network researchers, being considered an important technique to evaluate the effectiveness of new protocols and applications in heterogeneous, controllable and realistic network scenarios. Today's most complex network emulation systems are cluster-based. These systems are made of a large number of hardware components arranged in a common facility that can be remotely accessed by users through a web interface. In a typical cluster-based network emulation system, users submit to the system an experiment request. An experiment request contains a "virtual" network description to be reproduced with the available cluster resources.

Most of existing emulation systems concentrate on the provisioning of resources but lack of procedures which would automatize the execution of experiments and the collection of results. These features are some of the main characteristics of OMF, a testbed *cOntrol and Management Framework* originally developed for the ORBIT testbed. Leveraging OMF, these important capabilities were added to the NEPTUNE network emulation system. The outcome is a system that allows the fast and automatic creation of "virtual OMF testbeds" on-demand. As in the case of GaaS (cfr. Â§1.1.4),

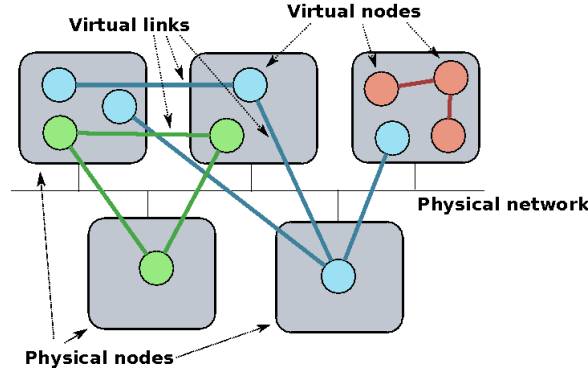


Figure 1.3: Virtual network topology on a cluster of PCs.

the combination of OMF and NEPTUNE provides a PaaS-like paradigm for network testbeds.

### 1.2.2 OMF architecture

OMF is a platform supporting the management and the automatic execution of experiments on a networking testbed. Currently OMF is being developed by NICTA [123].

OMF supports all the phases of an experiment lifecycle, from the provisioning of resources to the collection of experimental data. The most important component is the *Experiment Controller* (EC), which is also the interface to the user. It accepts as input an *experiment description* and takes care of orchestrating the testbed resources in order to accomplish the required experiment steps. It interacts with the *AggregateManager*, the entity responsible of the resources of the testbed as a whole, and provides some basic services to the EC, such as checking the status of a node, rebooting a node, etc. The EC also interacts with the *Resource Controllers* (RCs) installed on the testbed nodes, that are responsible of performing local configuration steps and of controlling the applications, e.g. a traffic generator.



### 1.2.3 NEPTUNE

NEPTUNE [52] is an open-source cluster-based network emulation system developed at University of Napoli “Federico II” that can be used to assess new networking technologies and protocols (e.g. new QoS routing protocols and Traffic Engineering schemes in MPLS-based networks), as well as new distributed applications and architectures (e.g. multimedia peer-to-peer applications). NEPTUNE provides researchers with the ability of interactively designing multiple virtual network topologies, which are then deployed onto a cluster of real machines and used as if they were dedicated physical testbeds. NEPTUNE was designed with two goals in mind: manageability and portability. Manageability, because a requirement was that NEPTUNE could have been easily deployed and managed by system administrators. Portability, since NEPTUNE is not linked to specific hardware solutions, but it can be installed on general purpose machines and its features can be conveniently extended by software developers. In NEPTUNE, an experiment is a collection of virtual nodes deployed on a subset of a cluster’s physical nodes, each running a virtualization layer, and properly configured in order to reproduce a user-defined virtual network topology. To achieve higher degrees of scalability, complex systems are reproduced by allocating multiple virtual network nodes onto each of the cluster’s real nodes (*node multiplexing*). Likewise, multiple virtual links are multiplexed onto the same shared physical link by associating each virtual link endpoint to a different virtual NIC (*link multiplexing*). Multiple fully isolated experiments can be run by NEPTUNE at the the same time, while providing users with the illusion of having allocated a dedicated infrastructure (virtual cluster). A role-based authentication system allows flexible definition of roles and actions allowed to each role. Roles and permissions are stored in XML files for simple editing, to allow system administrators modify policies even at run-time.

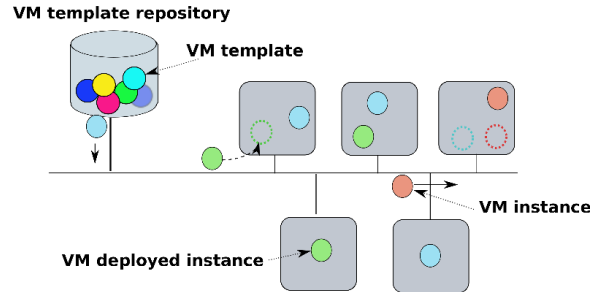


Figure 1.4: VM instantiation in NEPTUNE.

## NEPTUNE Architecture

NEPTUNE’s physical architecture (Figure 1.5) is composed of three parts: i) a set of worker nodes providing computational resources used to reproduce emulated networks, ii) a centralized repository providing storage space to worker nodes and iii) a front-end node, NeptuneManager, hosting system management software. All these physical components are connected by a switched LAN, carrying what we call “control traffic”. Worker nodes are also connected by a second high-performance LAN, carrying traffic generated by users’ experiments. All users (both normal users and administrators) access the system through the NeptuneManager web interface. All system functions are exposed by this interface, so users can set up and execute their emulation experiments by means of a user-friendly AJAX web-interface.

For testing purposes, NEPTUNE runs on a cluster of 28 HP ProLiant DL380 servers, each equipped with two Intel Pentium IV Xeon 2.8 GHz CPUs, 5 GB of PC-2100 RAM, one 100 Mbps Ethernet NIC, one Gigabit Ethernet NIC. Each node is equipped with a 34.6 GB SCSI disk. A 700GB centralized disk array is also available to the whole cluster. The cluster nodes are connected each other through a set of 100/1000 Ethernet switches.

## Usage Model

An experiment life-cycle begins with the definition of a virtual network topology. Once the topology is defined, an experiment can be allocated onto the cluster’s physical nodes. A running experiment can be either suspended

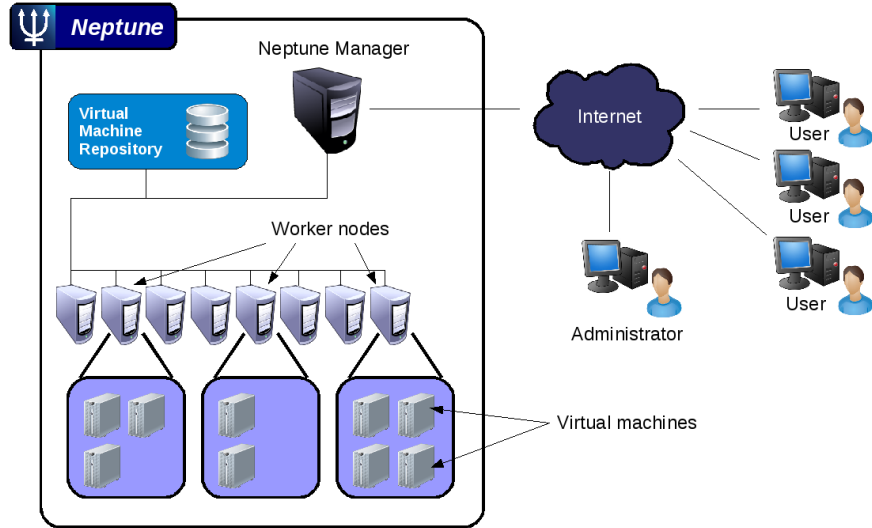


Figure 1.5: NEPTUNE architecture.

for future reallocation or definitively terminated. Allocation of experiments onto the cluster is made under control of system administrators, who need to explicitly accept users requests. Once accepted, experiment's topology allocation process starts. Such allocation process is automatic, involving tasks like virtual nodes mapping on cluster's physical nodes and IP addresses assignments.

To define virtual topologies, users can both write a topology description in a custom XML format or use an interactive graphic tool embedded into the web user interface. It is also possible to select pre-defined topologies for fast experiment definition, modify and in case save them as new topology templates.

To define virtual nodes software configuration, users can access via the Neptune web interface a "Virtual Nodes Template Images Repository" and select a VM template for each of the emulated nodes. VM templates, which enclose OS filesystem and in case other software, can be modified and saved as new templates for reuse.

Furthermore, users can control an experiment status and can execute actions to terminate that experiment or save it. Our current implementation

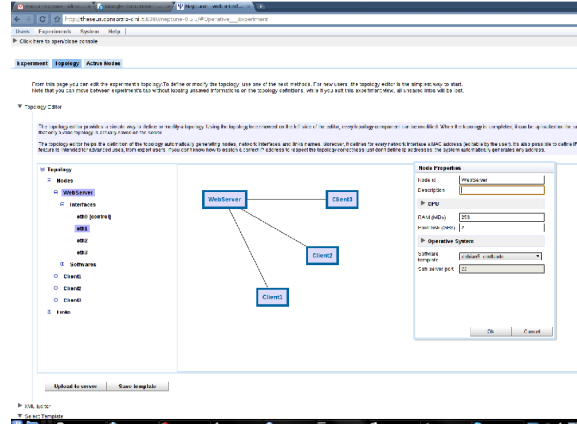


Figure 1.6: Neptune Manager screenshot.

of experiment status saving only creates copies of virtual nodes file systems, as saving the whole status of a running experiment is a distributed snapshot problem [28] which is actually out of the scope of our system. Commands and tools to manage and monitor virtual nodes and links are provided too. Finally, remote access is made available to each of the experiment nodes through a VPN tunnel.

#### 1.2.4 Automatic deployment of a OMF virtual testbed

To allow the fast and automatic execution of experiments on user-defined emulated virtual networks, we decided to extend our NEPTUNE platform by giving to it the ability to instantiate OMF-ready virtual testbeds.

The integration of the two platforms comprised two steps:

1. Configuration of two VM templates: i) one preconfigured with OMF EC and AG instances and ii) another one preconfigured with a OMF RC.
2. Creation of a set of scripts for automatically configuring OMF components using NEPTUNE's virtual topology description.

To automatically deploy an OMF virtual testbed, a user describes the testbed's topology he/she needs to perform experiments, by means of the

NEPTUNE GUI. The defined virtual testbed must use virtual nodes templates that include OMF components. A typical virtual testbed setup could be composed of a single virtual node hosting OMF EC and OMF AG, and several virtual nodes hosting OMF RCs. Once the virtual topology has been defined in NEPTUNE, our scripts automatically configure the OMF components according to the provided virtual topology. Once this last step is completed, the virtual testbed is ready to be used. User can then access the nodes through secure ssh connection or supply an experiment description to the OMF EC, which will execute the experiment. At the end of the experiment execution, the experimental results will be available on a MySQL database.

## Chapter 2

# Providing flexible infrastructures

In the previous chapter we introduced the recent trend in providing everything applying an “as a service” model, as well as the findings in adapting some specific applications to such a model. Nevertheless, even if the user access model follows an “as a service” approach, as already pointed out, the underlying infrastructure, in most of the cases, is still a traditional TCP/IP network. Introducing flexibility in the traditional network infrastructure becomes a stringent requirement to support the highly dynamic service provisioning model, ensuring at the same time a small operational cost. In this chapter we will focus on issues we faced while designing both GaaS and NEPTUNE (both introduced in the previous chapter). In particular, we will explain how a common network substrate can be virtualized to provide virtual links, how to exploit the knowledge of the provided application to optimize the infrastructure provisioning time, and, finally, how to automatically optimize the resources usage in the substrate infrastructure.

### 2.1 Virtual links

#### 2.1.1 Context and motivation

Network emulation is an experimental methodology that is widely adopted to test innovative protocols and distributed applications in realistic and con-

trollable scenarios. Unlike simulation, which reproduces a system's behavior by modeling all the interacting components of the system, emulation allows researchers to test a *real* implementation of a system component, by making it interact in real-time with other real world or modeled components of the system [27]. In the specific case of networked systems, emulation consists in reproducing a “virtual” network setup on top of a collection of physical devices. In particular, one of the issues of network emulation is the ability of reproducing the behavior of different communication links (such as geographic point-to-point links, shared LANs, wireless LANs, and so on) on top of a general purpose facility. One of the first emulation tools was *dummynet* [128], a software system developed by Luigi Rizzo as an extension of the FreeBSD kernel. Dummynet makes a FreeBSD system able to shape and delay the traffic flowing through it. Due to the ease of deployment, dummynet is often used in small scale testbeds to emulate the behavior of congested wide area networks for testing of protocols and applications. A modified version of dummynet has also been recently deployed in PlanetLab Europe [118], where FreeBSD “Dummynet Boxes” have been deployed in front of a subset of PlanetLab Europe nodes. Dummynet Box can be dynamically configured, so that individual users (slivers) can independently and concurrently set up the characteristics of an emulated link for their experiments [81]. Modern network emulation systems are able to reproduce in a virtual environment the behavior of complex network topologies, and let these virtual networks interact with real applications under test. The architectures of these emulation systems are extremely different, ranging from *centralized implementations*, reproducing the emulated network within a single computer, to *distributed emulation facilities*, usually relying on clusters of PCs interconnected by programmable networking devices. We call this latter kind of systems “cluster-based network emulation systems”. Since realistic evaluation scenarios often require thousands of nodes, scalability is a key requirement for network emulation systems. Different solutions have been proposed in the literature to scale-up the maximum size of emulated networks. Grau

et. al classify them into parallelization, abstraction, node virtualization, and time virtualization [53]. Node virtualization has been used in several cluster-based network emulation systems, such as University of Utah’s EmuLab [62] and University of Stuttgart’s NET [60] [84]. NEPTUNE (see chapter 1) is a cluster based network emulation system that makes use of Xen for node virtualization [41],[40]. Besides node multiplexing, a network emulation system needs proper techniques to emulate the behavior of different point-to-point links on top of a shared networking infrastructure. This latter problem is usually referred to as *link multiplexing*. In this section we present and experimentally evaluate the “One Link per Virtual Interface” technique (OLVI in short) we use in NEPTUNE as the basis for link multiplexing. OLVI combines Xen bridging with the emulation features provided by the NetEm extension of the Linux kernel [59] to multiplex several point-to-point communication links, each of which with its own bandwidth and delay, on top of a single high-performance LAN.

### 2.1.2 Virtualization in Network Emulation Systems

A typical network emulation system is composed of a set of physical resources (links, LAN switches, PCs) that are used to reproduce an emulated networking environment. Several strategies can be adopted to map the emulated scenarios on top of the available physical resources. While conservative allocation policies may easily lead to underutilization, better resource utilization might be achieved if they could be decomposed in many “virtual resources”, each appearing as a separate physical resource. Such a technique is usually referred to as *resource multiplexing*. In the context of network emulation we are interested in two particular forms of resource multiplexing: *node multiplexing* and *link multiplexing* [31]. Node multiplexing is the problem of emulating more than a network node on the same physical node, while link multiplexing focuses on emulating multiple point-to-point connections on top of one or more shared links.

Virtualization technologies are a widely used solution for *resource mul-*



*tiplering* problems. In general terms, virtualization is a technique in which a software layer multiplexes lower-level resources for the benefit of higher level software programs and systems. Virtualization can be applied to either single physical resources of a computing system (e.g. a single device) or to a complete computing system. When applied in this latter sense, (a.k.a. *Platform Virtualization*), it allows the coexistence of multiple “Virtual Machines” in the same computing host. Platform virtualization is implemented by means of an additional software layer, called Virtual Machine Monitor (VMM) (or *hypervisor*), that acts as an intermediary between the system hardware resources and the Operating System. There are many approaches to platform virtualization: *Full Virtualization* implements in software a full virtual replica of the emulated system’s hardware, so that the operating system and user applications may run on the virtual hardware exactly as they would in the original system. *Paravirtualization*, instead, makes available a software interface to virtual machines that is similar but not identical to the underlying hardware in order to improve scalability and performance over full virtualization, at the cost of requiring the guest operating system to be explicitly ported for the para-API. Finally, *Operating system-level virtualization* further improves scalability allowing a physical server to run multiple isolated operating system instances sharing the same kernel with little overhead, but at the cost of a reduced flexibility.

While node multiplexing is inherently a problem of platform virtualization, link multiplexing is a more specific problem that can be solved in different ways, at different layers of the communications stack. Several network emulation systems have been designed (or re-designed) in order to use virtualization techniques for efficient resource multiplexing. University of Utah’s Emulab, as first example, implements node multiplexing by means of a modified version of FreeBSD Jail. ModelNet [144] implements node multiplexing by means of so called Virtual Nodes (VNs) which is just a process level isolation approach, while link multiplexing is implemented by combining IP aliasing, a socket interposition library and centralized Core Nodes running

dummysnet. University of Stuttgart’s NET implements link multiplexing by combining the use of VLANs and a virtual device driver, NETShaper, which allows to dynamically configure bandwidth, delay and loss rate [61]. Besides network emulation systems, virtualization techniques have also been used for resource multiplexing in large scale distributed testbeds. The VINI project has created a virtual network infrastructure allowing experimental evaluation of protocols and services under real traffic loads, in controllable network conditions [16]. VINI uses two container based virtualization technologies for node multiplexing, VServer and NetNS, in addition to Ethernet EGRE tunneling [19] for layer 2 encapsulation.

The use of virtualization techniques is also at the basis of NEPTUNE. NEPTUNE relies on Xen [14] for node multiplexing, which implements paravirtualization by means of an hypervisor and several domains, running on top of that hypervisor. The hypervisor controls guest domains access to the physical machine’s hardware resources, while for the sake of reliability and efficiency, device drivers are kept in an isolated “driver domain” (Domain 0, or dom0) with special privileges. Domain0 is created at boot time and, through it, users may create and terminate other unprivileged domains (domUs), control CPU scheduling parameters and resource allocation policies.

### **Node multiplexing in NEPTUNE**

Node multiplexing is implemented in NEPTUNE by means of Xen, using virtual machines as network nodes. Our current implementation relies on libvirt virtualization APIs [125], making it feasible supporting different virtualization technologies in the future. Mapping of virtual nodes onto the cluster physical nodes is described by an allocation map which can be generated either manually by a system administrator or automatically, by means of a software module implementing a Lin-Kernighan derived optimization algorithm [126]. When a virtual network is to be deployed on the physical cluster, NeptuneManager distributes Virtual Machine template instances to the physical cluster nodes. This distribution process is composed of two

phases for each virtual node: 1) raw copy of the virtual machine image file containing VM template, and 2) VM creation on the target virtual machine monitor. During this last phase, virtual hardware resources are provided to the virtual node according to node definition provided by the experiment topology description. Use of Xen for node multiplexing provides a totally virtualized environment to test applications. Xen isolates virtual machines from each other and guarantees them the availability of resources assigned at VM creation time. Because of the total isolation between VMs, it is possible to run custom operating systems on each VM and, hence, custom network stacks. This allows us to correctly emulate different network devices, e.g. routers, within a single physical host.

### **Link Multiplexing in NEPTUNE**

In several network emulation systems, link multiplexing is performed by means of Virtual LANs (VLANs). Such a solution is implemented by properly configuring the Ethernet switches and does not require any configuration and processing in the cluster nodes. This makes, however, the system configuration software extremely dependent on the characteristics of the network switches. For the above reason, we decided not to use VLANs in NEPTUNE and we adopted a network device independent solution for link multiplexing that we call “One Link per Virtual Interface” (OLVI in short). The OLVI technique is implemented by exploiting the network virtualization mechanisms implemented in Xen. Every time a new virtual machine is instantiated, Xen creates a new pair of “connected virtual ethernet interfaces”, with one end of each pair within the virtual machine and the other end within the virtual machine monitor. Virtualised network interfaces have their own ethernet MAC addresses, whose values can be assigned at virtual machine creation time [151][92]. When using OLVI technique, each point-to-point link is identified by its end points, which are virtual NICs in virtual nodes. Since a unique MAC address is assigned to each virtual NIC, virtual links are uniquely identifiable within NEPTUNE. To completely implement link em-

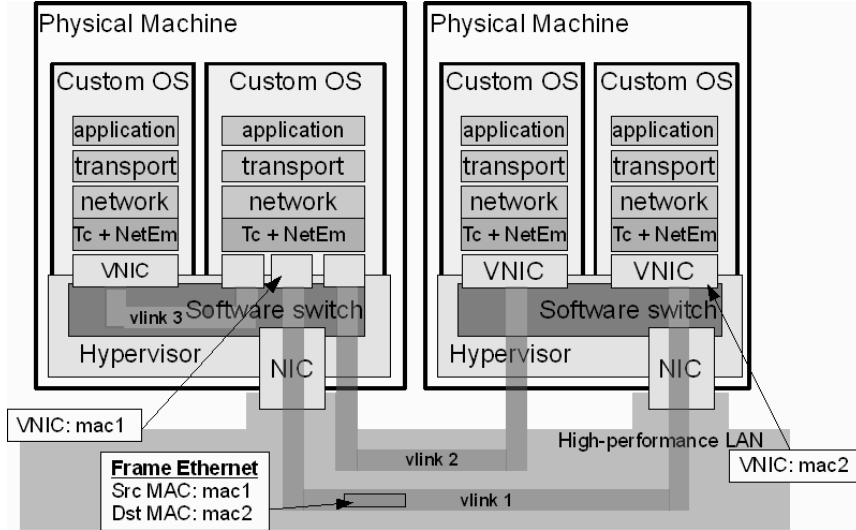


Figure 2.1: Link multiplexing in NEPTUNE.

ulation, virtual links need to be configured according to specific user-defined properties. Such properties are assigned to emulated links through the use of a queuing discipline and a traffic shaper that are associated to both ends of an emulated link. Queuing discipline are enforced through the traffic control module of the Linux kernel, while traffic shaping is done through NetEm, another Linux kernel module, provided by default from kernel2.6 distributions, that gives the possibility to emulate delay, loss rate, re-ordering and duplication over a link.

A major problem when dealing with the creation of virtual links is the need to assign IP addresses to both ends of virtual links, according to a general IP addressing scheme. In NEPTUNE users can manually define IP addresses for a link's end-point, but such a task is tedious, error prone and not viable when dealing with big topologies. For these reasons NEPTUNE also provides an algorithm that automatically assigns subnets to links and IP addresses to their end-points. Furthermore, since several experiments can be running on the same shared infrastructure, the algorithm also ensures non overlapping of address spaces used by different experiments. This latter requirement is also enforced when experiments use manual allocation of IP

addresses.

### 2.1.3 Experimental Evaluation

The need to create repeatable experimental scenarios to test network applications and protocols is the main reason for the adoption of simulation and emulation techniques. Node and link virtualization in network emulation testbeds provide users with the possibility of trading off the amount of physical resources to be allocated to a given experiment for emulation accuracy. First target of our experimental evaluation is to identify limits of proposed multiplexing techniques and in particular the maximum network throughput a NEPTUNE's virtual node can achieve. Here we discuss experimental results demonstrating the maximum throughput obtainable when one VM is running on top of the hypervisor: these performance values are an upper bound for the case of multiple concurrent VMs running on top of the same physical hardware. To assess performance levels of NEPTUNE, first of all we need to find a reference performance value. To this end, we set up a preliminary experimental scenario (Setup #0), in which two identical machines are connected by a point-to-point 1Gbps Ethernet cable. Both nodes are HP ProLiant DL380 servers, each equipped with two Intel Pentium IV Xeon 2.8 GHz CPUs, 5 GB of PC-2100 RAM, one 100 Mbps Ethernet NIC and one Gigabit Ethernet NIC. The adopted CPUs support the Hyper-Threading Intel technology. Both hosts run native GNU/Linux (CentOS5.3) with a 2.6.18 Linux kernel.

In such a scenario, we run the D-ITG suite [25] to generate network traffic and measure effective throughput in terms of generated/received packets per second (pkt/s). We used D-ITG to generate UDP constant bit rate traffic, at different rates, with packets size of 1042 bytes “on wire”.

Similar experiments were repeated among the same pair of physical machines, but in a different experimental scenario (Setup #1 shown in Figure 2.2): one of the two hosts runs a Xen Virtual Machine Monitor (Xen version 3.1.2) with a 2.6.18 Linux kernel plus a single domU virtual machine.

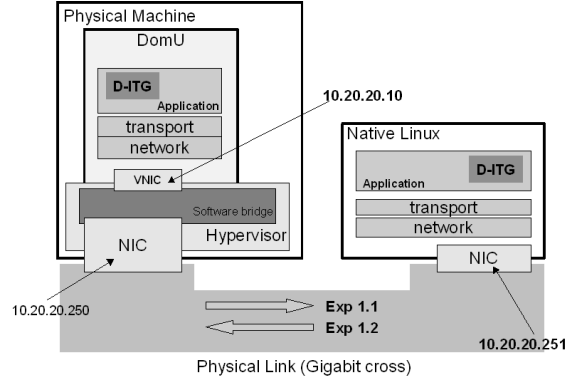


Figure 2.2: Experimental setup #1.

Two different experiments were run in this configuration: Experiment 1.1, where domU node acted as CBR traffic generator, and Experiment 1.2, where domU acted as receiver. Due to the use of Hyper-Threading, our systems appear having four “logical” CPUs, numbered as CPU0, CPU1, CPU2 and CPU3. The CPU enumeration order used by Xen is: hyperthreads, cores, sockets. On our system, the four CPUs are then mapped as follows:

- cpu 0 : socket 0, [core 0], hyperthread 0;
- cpu 1 : socket 0, [core 0], hyperthread 1;
- cpu 2 : socket 1, [core 0], hyperthread 0;
- cpu 3 : socket 1, [core 0], hyperthread 1.

We configured Xen by constraining domUs to use no more than one CPU at a time.

Experiment run in Setup #0 provides us with the maximum number of packets per second that our Linux-based hosts are able to receive. Since experiments run in Setup #1 show a lower throughput, we are confident that this performance penalty is caused by the use of Xen.

Figure 2.3 shows achieved throughput for Experiment 1.1, compared to throughput measured in Setup #0. This graph demonstrated that domU performance is about 75% of the native GNU/Linux host, providing ourself with

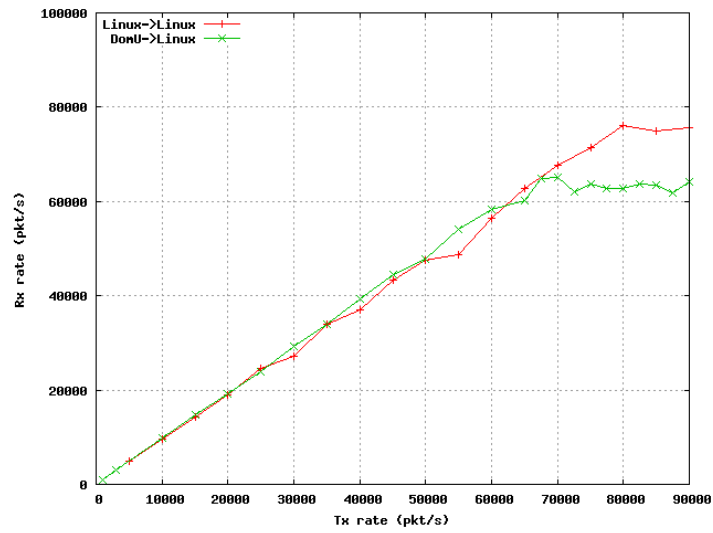


Figure 2.3: DomU transmission performance.

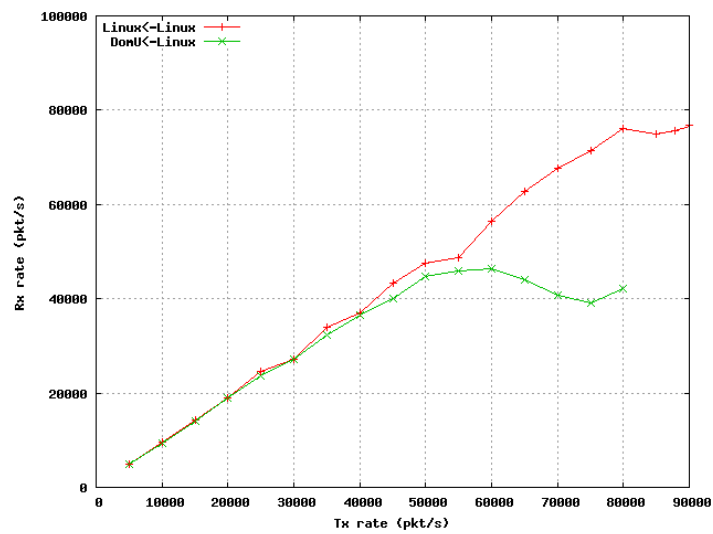


Figure 2.4: DomU reception performance.

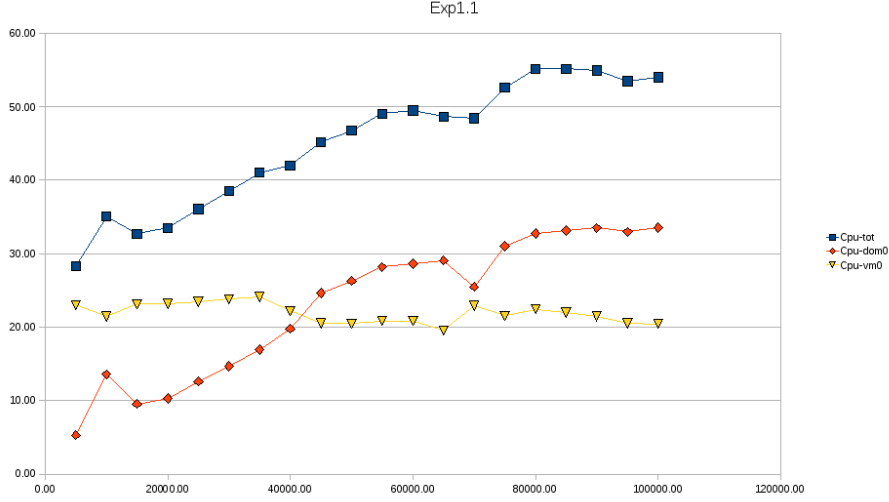


Figure 2.5: DomU transmission performance.

an upper bound for transmission capabilities of a NEPTUNE virtual node. Figure 2.4 shows achieved throughput for Experiment 1.2, again compared to throughput measured in Setup #0. This graph shows that, as receiver, domU performance is 65% of native GNU/Linux.

Figures 2.5 and 2.6 show the average CPU load measured during Experiments 1.1 and 1.2, as reported by the *virt-top* monitoring tool. The overall CPU capacity refers to the whole set of four logical CPUs, hence, due to the configuration of domUs, each domU may consume as much as 25% of the overall CPU capacity. Figure 2.5 shows that the traffic generating domU consumes as much CPU as possible even at low packet rates, while dom0 increases the CPU utilization as the packet rate increases. Figure 2.6 shows that a Xen domU acting as a packet receiver linearly increases its CPU utilization with the packet arrival rate up to a given threshold. Since incoming packets are first processed by dom0, the domU CPU load reflects the dom0 curve up to a threshold value (50 kpacket/s in Figure 2.6). For packet rates above this latter threshold, domU saturates its CPU utilization curve.

We also carried out an experimental evaluation of the impact of NetEm on performance of the emulation layer. These experiments were performed



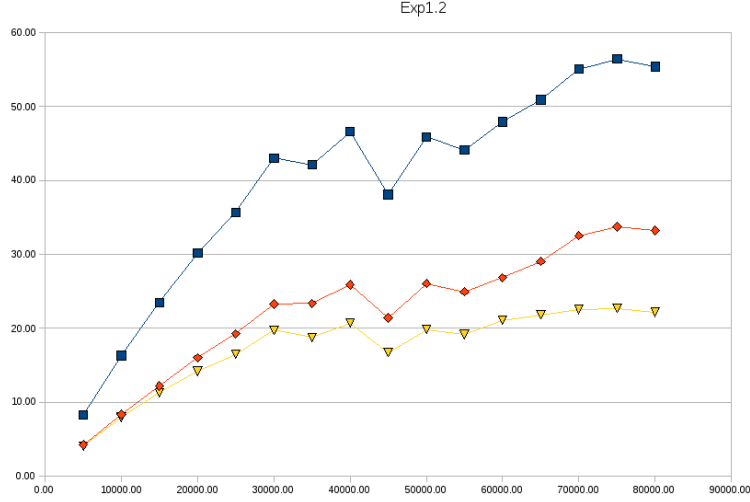


Figure 2.6: DomU reception performance.

using the same pair of physical machines as for previous experiments, but engaging a different scenario (Setup #2 shown in Figure 2.7). Since our tests were unidirectional, we only configured NetEm on the transmitting side.

Again, we performed two experiments. In both of them, we configured the emulation layer to emulate a virtual link of limited bandwidth, by means of a token bucket filter (TBF). Our goal was to evaluate if the combination of bandwidth limitation mechanisms and virtualization layer (Xen) produced any unexpected effects on the throughput and jitter experienced by packets carried by the emulated link. Figure 2.8 shows traffic received by the native GNU/Linux host, for several TBF limits applied at the sender side, when the rate of generated traffic was progressively increased. This graph demonstrates that traffic rate has been correctly shaped, delivering the expected bandwidth.

In Figure 2.9 we compare jitter values experienced on link when using two different configurations: in the first one, traffic sender run on native GNU/Linux (no virtualization layer here), while in the second one a Virtual Machine worked as sender (Figure 2.5). In both cases we generated 25 Mbps of UDP constant bit rate traffic over a virtual link tailored at 20 Mbps by

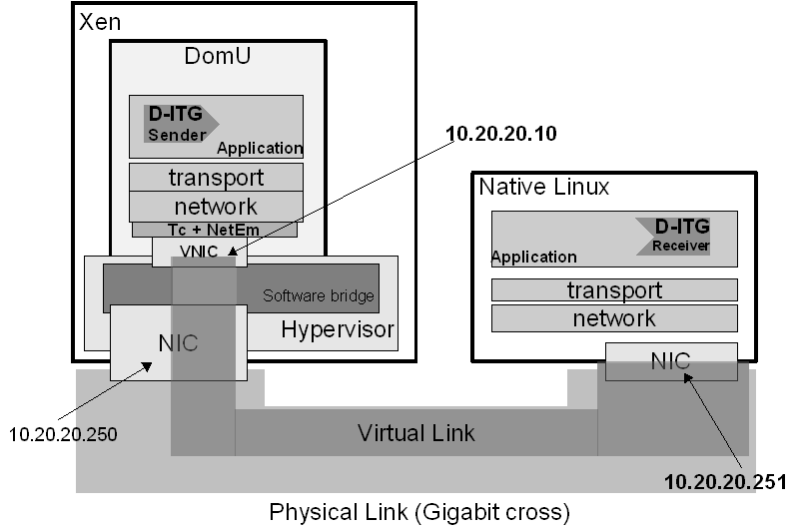


Figure 2.7: Experimental setup #2.

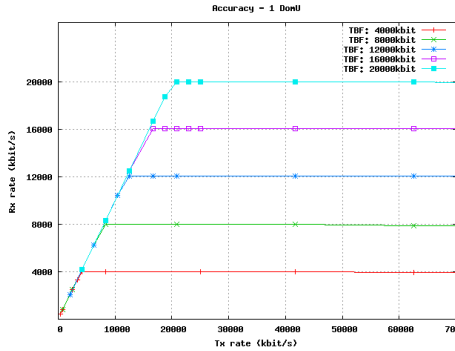


Figure 2.8: Token Bucket bw limitation.

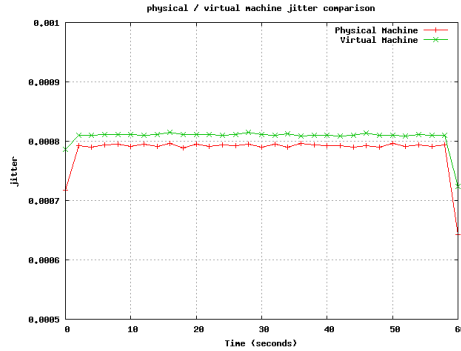


Figure 2.9: Jitter.

applying a TBF shaper. Results show that the virtualization layer has no significant effects on the jitter, that maintained similar properties.

## 2.1.4 Conclusions

Several papers presented performance evaluation analysis of Xen networking. Our work is not specifically aimed at evaluating Xen performance. Nonetheless, since Virtual Machine Monitor layer is a major component of our network emulator architecture, experiments we presented in this section are strongly related to those presented in other works. In [43], packet forward-

ing performance of Xen’s dom0 and domUs are evaluated and compared to native GNU/Linux. Results show that dom0, in the absence of concurrent domUs, has comparable performance (within 5%) of native GNU/Linux. On the other hand, a remarkable performance drop is experienced by domUs, especially in the case of multiple concurrently active domUs running on top of the VMM. This paper also compares performance of Xen’s bridging and routing configurations in terms of packet forwarding within domUs. This last work is of less interest for our purposes, since we assumed the use of bridging configuration for Xen in the NEPTUNE architecture. Other papers ([93][33]) have evaluated overhead caused by the VMM layer, for previous versions of Xen (v2.x). In [93] a detailed profiling of Xen shows that the execution of I/O operations in a domU has a higher instructions count with respect to both native GNU/Linux and dom0, which explains throughput degradation. Here, authors also examine how performance can be negatively affected by the use of a general virtual NIC driver that causes domUs to execute operations like TCP segmentation in software, ignoring physical NIC TCO (TCP segmentation offload) capabilities, which are instead used by native GNU/Linux and dom0. Our experiments show similar results to those presented in previous papers. The relevance of results we provide is mainly to identify an upper bound to the capabilities of our emulation system, in order to guarantee correctness of the emulation.

## 2.2 Fast provisioning of virtual nodes

In many IaaS management system, VM templates are usually stored in a “template repository”. A new VM is created copying the selected template to the running location of the VM. The duration of this process is the main factor in the resources deployment time. Since the copy process involves the storage infrastructures that are hosting the *template repository* and the newly created VMs disks, assuming that a minimal VM template is several hundreds of megabytes big, the process, for each VM creation, takes a time in the order of dozens of seconds.

During the design and development of the GaaS system (see chapter 1), to optimize the infrastructural resources used during the provisioning process, and to reduce the overall provisioning time, we took into account the peculiarities of the GaaS system. In particular, we made the following observations:

1. all the VMs are prescribed to host the same operating system, which is imposed by the gLite-EMI middleware;
2. all the VMs hosting the Grid services (e.g., WN, CE, etc.) can be produced by customizing the configuration of a single VM template;

We designed our VM disk provisioning system in order to provide fast VM creation and avoid as much data copy as possible. Our solution is based on the GNU/Linux's Logical Volume Manager (LVM). LVM allows the creation of logical volumes (LV) and the creation of snapshots starting from a reference LV. Snapshots can be read and written, since their creation is performed through the use of a "*delta meta-data*", that contains all the differences with the original LV. This approach makes the creation of a snapshot really fast (a few milliseconds) since it involves no copy of data. Once the snapshot is created, following observation 2, a configuration script is executed to customize the virtual resource according to its functional destination. Since both read and write actions involve an a read/update of the *delta meta-data*, the operating performance of the snapshot could be compromised in particular conditions. In our case, we are mainly interested into reading performance, and, moreover, we assume that the majority of reads happen in the bootstrapping process of a VM (e.g., for the loading of the required applications).

In figure 2.10 is presented a performance comparison of read and write operations on 64 KBs data blocks. These tests were executed on an HP DL380 Proliant server equipped with two Intel Pentium IV Xeon 2.8 GHz CPUs, 5 GB of PC-2100 RAM. The server was running a Debian Linux with the OS kernel configured to use only 1 GB of RAM. We compare the results obtained by operating on both a raw partition (labeled as "normal") and on a

LVM snapshot (labeled as “snapshot”). For various amounts of read/written data (ranging from 512 bytes up to 8 GBytes), we compare the throughput achieved for write (left graphs) and read (right graphs) operations. Figure 2.10 shows the results of six series of experiments. On the left hand of the figure, graphs a), c) and e) present the throughput obtained by write operations performed in the following cases: graph a) refers to operations performed on freshly mounted disks, with no OS caching effects, graph c) refers to sequential write operations performed several times on the same data, in order to maximize the OS caching effects, graph e) refers to random write operations. On the right hand of the figure, graphs b), d) and f) present the results for similar experiments involving read operations.

From the graphs it can be easily observed the effect of caches on performance, when data size is bigger than the available filesystem cache (our system had about 512 MBs of cache space). What is of particular interest for us, is that the performance drop when snapshots are used is marginal, since the *delta meta-data* is likely stored in the system cache anyway, hence, we can fast provision resources to the grid, without paying any sensible penalty on disk performance. Moreover, performance drops are visible when the written/read data size is bigger than the filesystem cache, in particular for write operations. Since snapshots are just used to create VMs’ OS bootable disks, i.e., disks that contains the OS and the applications code but that are not meant to be used as data storage, they are mainly involved in read operations, and the dimension of read data is likely to be smaller or comparable with the dimension of the filesystem cache. Hence, we expect little or no performance drop in the VM operations.

## 2.3 Resource allocation problem

One of the key steps in the Virtual Infrastructure deployment process is the mapping of Virtual Machines onto the physical resources of the target data-center. This problem is known in literature as the *network testbed mapping problem*[126]. Due to its complexity, the challenge is to find a *good* solu-

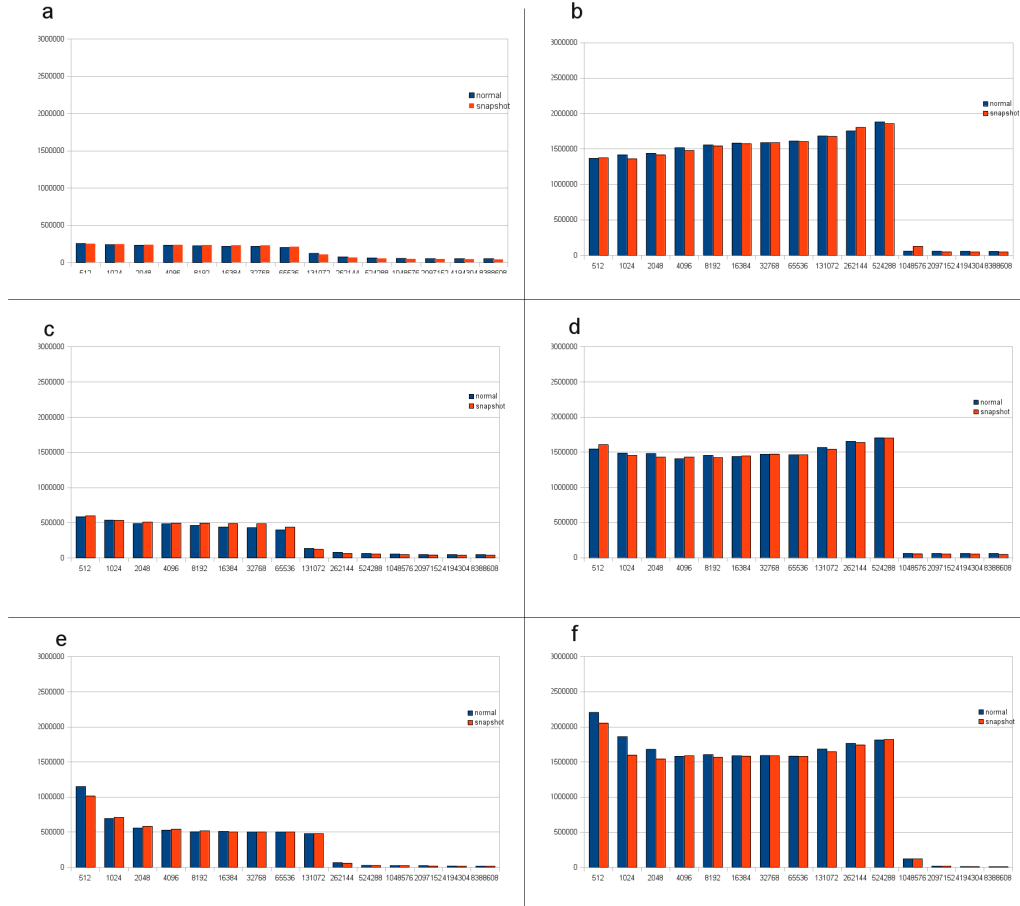


Figure 2.10: Storage write (left column) and read (right column) performance. The y axis shows the read and write throughput (bytes/sec), while the x axis shows the amount of data read/written. Graphs a, b refer to sequential operations with no OS caching effects. Graphs c, d refer to sequential repeated operations to maximize OS caching effects. Graphs e, f refer to random operations.

tion in *acceptable* computational times. Our approach to manage complexity consists in splitting the mapping problem in two sub-problems: *topology partitioning* and a *partition mapping*.

Several graph partitioning algorithms have been proposed in the literature. An algorithm that provides good results with reasonable times of calculation is the Lin-Kernighan (LK) heuristic algorithm [70]. Theoretical complexity of LK is  $O(n^2 \log n)$ . We implemented this algorithm in JAVA to evaluate its applicability to cluster environments and to assess its per-

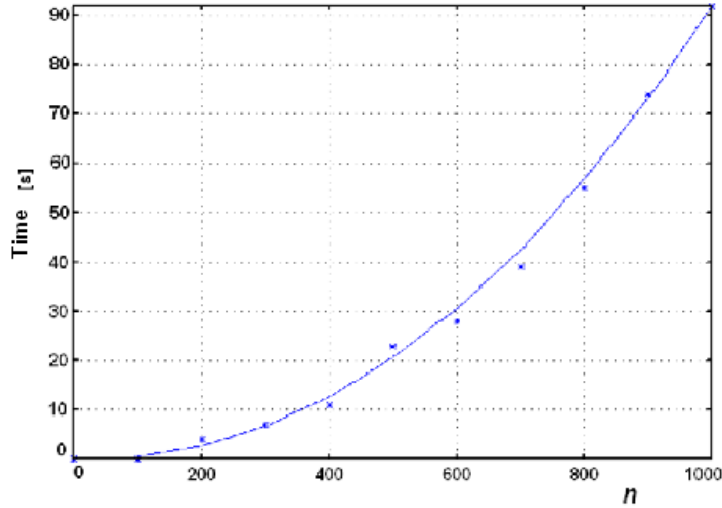


Figure 2.11: LK best mapping solution times

formance. A first test was performed to estimate the solver execution time while varying number of nodes in the graph. Size of the matrix was varied between 100x100 and 1000x1000 with steps of 100. The graph was been partitioned into subsets of cardinality equal to 5 while non-zero elements incidence for considered matrix were 2%. Computational times represented in Figure 2.11 were calculated by using a system equipped with 2 GB of RAM and an Intel CPU T2250 running at 1.73 GHz.

Our algorithm implementation requires that once found a minimum cost solution, the procedure is restarted with a new initial solution. After running 5 iterations the algorithm stops and returns the minimum cost solution. This test highlights the relationship between the iteration  $i^*$  at which the optimal solution is found with the size and density (arcs/nodes ratio) of the matrix.

	arc/nodes=4	arc/nodes=6	arc/nodes=8
Matrix 20x20	100run	100run	100run
Matrix 100x100	100run	100run	100run
Matrix 400x400	100run	100run	100run

Table 2.1: Tests organization

Virtual links and physical links bandwidths have been respectively fixed

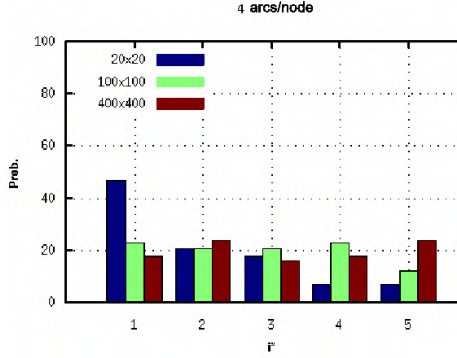


Figure 2.12: Four-arcs/node case

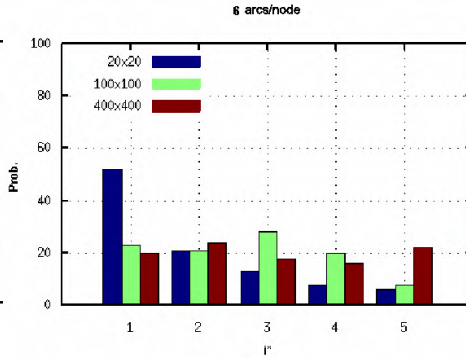


Figure 2.13: Six-arcs/node case

at 10 and 100. Matrices are generated randomly and before subjecting a matrix to the solver, it is verified that each node has at least one connection and that the sum of the costs associated to all the outgoing arcs from one same node does not exceed 90% of the physical connections bandwidth. Tests organization is shown in Table 2.1, while tests results are shown in Table 2.2.

	i*=1	i*=2	i*=3	i*=4	i*=5
Matrix 20x20 arc/nodes=4	47	21	18	7	7
Matrix 20x20 arc/nodes=6	52	21	13	8	6
Matrix 20x20 arc/nodes=8	43	27	14	6	10
Matrix 100x100 arc/nodes=4	23	21	21	23	12
Matrix 100x100 arc/nodes=6	23	21	28	20	8
Matrix 100x100 arc/nodes=8	18	16	26	18	22
Matrix 400x400 arc/nodes=4	18	24	16	18	24
Matrix 400x400 arc/nodes=6	20	24	18	16	22
Matrix 400x400 arc/nodes=8	22	28	14	12	24

Table 2.2: Tests results

Results for the case 4 arcs/node and 6 arcs/node are further shown in Figure 2.12 and in Figure 2.13.

This test demonstrates that for matrices of small size (20x20), our solver returns in almost 50% of the cases the least-cost solution at the first iteration. When the matrix increases in size (100x100 and 400x400), the probability of finding good solutions at the first iteration is lower. In these cases, better re-



sults could be obtained by running more iterations, but the rapid increase of computational times does not encourage this approach. The Lin-Kernighan algorithm does not guarantee that it is always possible to find an admissible solution, so it could happen that the found solution does not meet the admissibility constraints. However, in our tests, the solver always returned an acceptable solution.

## Chapter 3

# Introducing mobility in virtual infrastructures

The adoption of flexible infrastructures provides a complete set of new features to IT and networks. The introduction of virtualized infrastructures allows the decoupling of services from their physical location. Unfortunately, one of the assumption on which the current Internet is built is that the network nodes do not change their points of attachment to the network. This is an already well-known limitation of TCP/IP networks, that has been explored in past years, mainly to provide the ability to change network to mobile nodes, such as the ones equipped with a Wi-Fi device. The just cited virtualized infrastructures may take advantage of such a feature as well, since they can move from one physical location to another, through a process commonly called migration. Even if the migration of a virtualized infrastructure is first of all a technological problem related to the virtualization technology, the migrated virtual infrastructure still has the need to maintain the Internet connectivity to keep on providing the given services. In this chapter we present some solutions to provide mobility to virtualized infrastructures when deployed in a traditional TCP/IP network, both in the context of a datacenter and on a geographical scale.

## 3.1 Local mobility

### 3.1.1 Context and motivation

Cloud Computing is “a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction” [97]. Customers find this paradigm convenient as it relieves them from the responsibility of buying and managing a dedicated computing infrastructure, since resources may be dynamically acquired and released, according to their actual needs. Cloud Providers, on the other hand, can take advantage of scale economies to organize and manage large datacenters, whose resources can be efficiently utilized by partitioning and renting them to a number of customers. In this section we focus on the IaaS model, which provides users with the ability to obtain a set of common fundamental computing resources (e.g., Virtual Machines, storage and networking services) that can be composed to create a customized *virtual infrastructure*. Despite its success, the IaaS paradigm poses new challenges in terms of management of the computing infrastructure: Cloud Providers have the responsibility to manage a large infrastructure that hosts a number of highly dynamic virtual infrastructures operated by different users. A typical commercial IaaS offering provides virtual infrastructures composed by a predefined set of VMs. A customer’s virtual infrastructure may comprise groups of VMs. For instance, Amazon EC2 [7] allows users to define so-called “security groups”. Each VM in a security group can directly communicate with all other VMs in the same security group, while traffic coming from external sources or destined to external hosts must pass through a firewall managed by the Cloud management system.

Modern virtualization technologies play a key role in modern datacenters for Cloud Computing, as they allow an efficient utilization of physical resources. Server virtualization technologies based on Virtual Machines are

able to flexibly consolidate the computing workload of a datacenter over a set of available physical servers, through migration of running VMs from a physical server to another. This feature is particularly useful when the number of managed VMs is high, as it provides the ability to dynamically redistribute the computing workload for both load balancing and hardware maintenance. Current virtualization technologies, however, only support *live migration* of running VMs within a single IP subnet. The networking infrastructure of large-scale datacenters is implemented according to redundant multi-tiered architectures, comprising a number of different IP subnets. Splitting the network infrastructure in several IP subnets limits the scope of migration of VMs to portions of the datacenter, and reduces the possibility for administrators to efficiently balance the load and reduce the energy consumption of the whole infrastructure.

In this section we describe a solution that allows transparent migration of VMs across the whole datacenter by adapting the novel *Service Switching* paradigm, originally proposed for supporting geographic migration of network services [86]. Our solution is based on the coordinated use of NAT rules and ARP proxying that needs to be consistently managed across the layers of the datacenter networking infrastructure. We describe in details how our approach can be easily implemented with current network devices without any modification to their hardware and present an experimental evaluation of an early prototype of our solution.

The rest of this section is organized as follows. Section 3.1.2 illustrates the hierarchical networking infrastructure of modern large scale datacenters. Section 3.1.3 presents the role of virtualization and how live migration of Virtual Machines can be used for an efficient management of physical resources in Cloud-enabled datacenters. Section 3.1.4 illustrates our solution for transparent migration of VMs across different IP subnets. We discuss all the possible communication patterns and how they are preserved in case of migration of one or more Virtual Machines of the same group. Section 3.1.5 illustrates a prototype of our solution using Xen-based Virtual Machines and

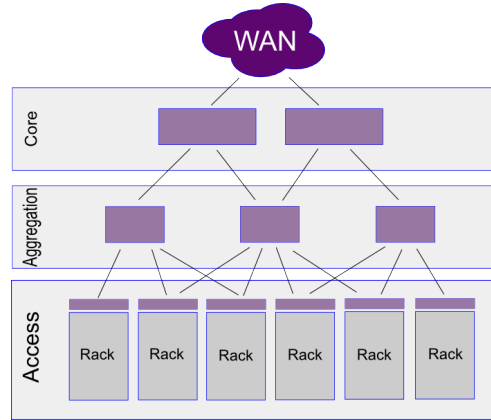


Figure 3.1: Datacenter layered architecture

Linux-based software routers. Our evaluation demonstrates the feasibility of our approach and its correct behaviour in all the scenarios presented in the previous section. In Section 3.1.6 we compare our solution against similar works that have been recently published in the literature. Finally, we draw our conclusions and illustrate our future developments.

### 3.1.2 Datacenter networking

Due to the large numbers of connected devices and the huge aggregated communication requirements, the networking infrastructure of a datacenter providing IaaS services is necessarily organized according to a hierarchical design. Figure 3.1 shows a typical network architecture for a large scale datacenter (adapted from [37]). Commercial datacenter networking solutions typically identify at least three levels of network devices, commonly referred to as *Access*, *Aggregation*, and *Core*. Core level devices connect the datacenter with the Internet, through one or more geographic links.

While the upper levels usually operate at layer 3 of the networking stack, i.e., they act as routers, Access layer devices may be configured to operate at either L2 or L3. The use of L2 devices at all layers of the infrastructure has proven to be unfeasible due to scalability problems deriving from too large broadcast domains. Spanning Tree Protocol (STP), in fact, can take up to 50 seconds to converge in a large network, while the Rapid Spanning Tree

Protocol (RTSP) still requires tens of seconds to converge in some topologies [67]. Hence, L3 solutions at the Access layer are recently preferred for large scale datacenters, as they provide faster routing convergence, contain broadcast domains to a limited size, and simplify troubleshooting and management procedures.

In the context of a datacenter for IaaS services it is common practice to use private IP addresses for internal networks. To guarantee public accessibility of services from the Internet, front-end nodes are associated to a limited set of public IP addresses, which are NAT-ted at the datacenter edge. For the purposes of this work, we do not consider such public IP addresses, and assume that, within the datacenter infrastructure, a VM is uniquely identified by one or more private IP addresses.

### 3.1.3 Virtualization

Virtualization is a widely adopted solution for *resource multiplexing* problems. In general terms, virtualization is a technique in which a software layer multiplexes lower-level resources for the benefit of higher level software programs and systems. Virtualization can be applied to either single physical resources of a computing system (e.g. a single device) or to a complete computing system. When applied in this latter sense, (a.k.a. *Platform Virtualization*), it allows the coexistence of multiple “Virtual Machines” in the same computing host. Platform virtualization is implemented by means of an additional software layer, called *Virtual Machine Monitor* (VMM) (or *hypervisor*), that acts as an intermediary between the system hardware resources and the Operating System. Modern VMMs (such as Xen, KVM, VMware vSphere) add support for *live migration* of Virtual Machines. Live migration is a feature that allows the migration of a running VM from one physical host to another, with a downtime limited to a few dozens of milliseconds. Live migration allows Cloud Providers to dynamically reconfigure the allocation of VMs over the available physical resources, so enabling advanced strategies for workload distribution and energy savings in the datacenter [132].

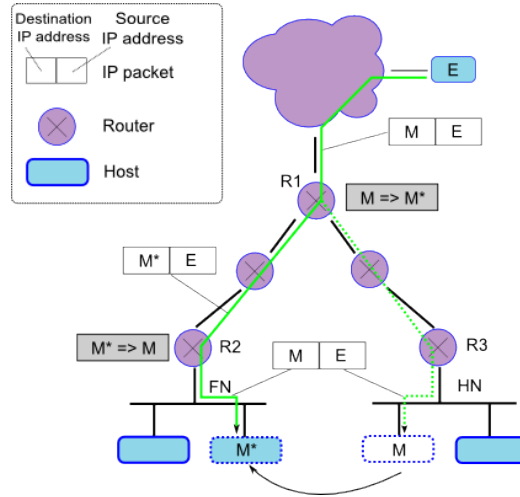


Figure 3.2: NAT-based Service Switching

A VM migration process basically consists in suspending the running VM, copying the status from the source VMM to the destination VMM, and finally resuming the VM at its destination. This process may be optimized with several techniques. Xen, for instance, uses an *iterative pre-copy* strategy, as described in [38].

The Xen live migration process described in [38] assumes that (i) source and destination VMMs both reside in the same LAN, and that (ii) a network-attached storage (NAS) provides a shared storage to VMs, which do not rely on local storage resources.

The problem of migrating virtual machines across the boundaries of a single IP network (or sub-network) is not considered in [38]. Similar constraints also exist for other virtualization technologies [146].

### 3.1.4 Our solution

In this section we illustrate a novel solution that allows transparent live migration of Virtual Machines within a large scale datacenter infrastructure comprising different IP subnets. Our solution is based on a modification of the *Service Switching* paradigm, a technique originally proposed in [86] with the aim of supporting geographic migration of virtualized network services.

Our datacenter-oriented implementation of Service Switching relies on a combination of *live migration* mechanisms [38], of the *Mobile IP* model [112], and of the *Network Address Translation* (NAT) technique.

IP version 4 assumes that a node's IP address uniquely identifies its point of attachment to the Internet: a node must be located on the network indicated by its IP address in order to receive datagrams which are destined to it. IP Mobility Support (or Mobile IP) provides a mechanism that allows *Mobile Hosts* to change their point of attachment to the Internet without changing their IP address. This mechanism relies on two intermediary entities: the *Home Agent* and the *Foreign Agent*. The role of the Home Agent is to maintain current location information of the mobile node, and to retransmit all the packets addressed to the Mobile Host through a tunnel to the Foreign Agent to which the Mobile Host is currently registered. The role of the Foreign Agent, in turn, is to deliver datagrams to the Mobile Host.

In Mobile IP, a Mobile Host interacts with the Foreign Agent to obtain the *Care of Address* (CoA), that is, the Foreign Agent's IP address. This address is notified by the Mobile Host to its own Home Agent, which, in turn, uses this piece of information to establish a tunnel to the Foreign Agent. Hence, the standard Mobile IP model, as described in [112], is not transparent to the Mobile Host, which is required to actively interact with the agents. Mobile Hosts, in fact, need to be able to discover agents and register with them. Agent discovery is performed through special ICMP messages, while host registration is based on the exchange of UDP datagrams sent to a well-known port (434).

As we mentioned before, the Service Switching paradigm is based on a few basic ideas taken from Mobile IP. In our context, a migrated Virtual Machine plays the role of a *Mobile Host*. However, the standard Mobile IP cannot be taken as-is, since one of the requirements for Service Switching is to make the VM migration process completely transparent to the migrated VM, which should be not aware of its migration. The Service Switching paradigm assumes that the VM migration procedure is managed by an ex-



ternal entity that is responsible of orchestrating the *live VM migration* with an *IP address migration* process. While the live VM migration process is strictly dependent on the adopted virtualization technology, the IP address migration process consists in properly configuring the devices forming the datacenter networking infrastructure in order to preserve the reachability of the migrated VM.

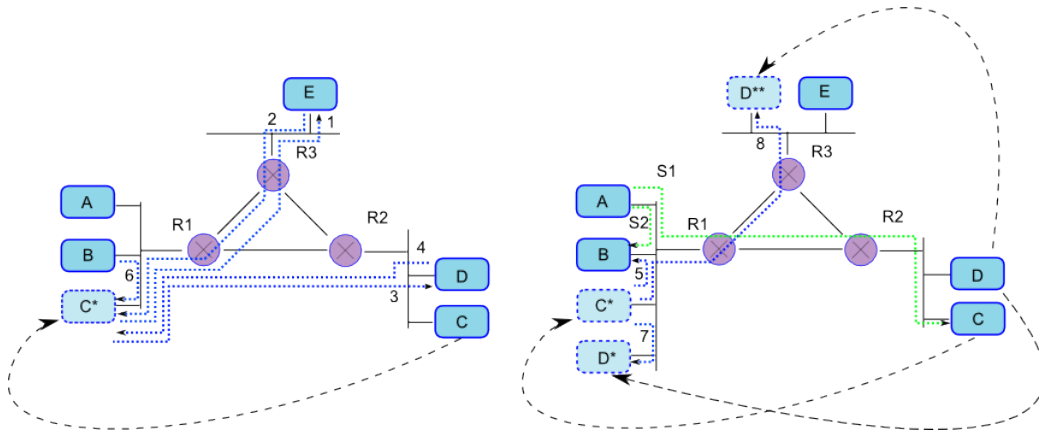


Figure 3.3: Transparent VM migration cases

In the following we describe the Service Switching *IP address migration* process in the context of a hierarchically organized large scale datacenter. We assume a datacenter with a Top-of-Rack layout, in which the access layer of the networking infrastructure is formed by one single switch per rack. A Top-of-Rack switch usually connects 20-40 servers per rack, typically with 1 Gbps links. The same model, however, can be easily adapted to datacenters with a End-of-Row layout. We also assume that access switches are configured to behave as Layer 3 devices, acting as IP routers and interacting with the upper layers of the networking infrastructure through an Interior Gateway Protocol, such as OSPF, RIP or IS-IS.

In order to solve the transparent live migration problem among IP subnets within a datacenter, we must operate at both layer 3 and layer 2 of the networking stack. Layer 3 operations enable the reachability of the migrated VM through plain IP routing and forwarding. Layer 2 operations are necessary to ensure that the migrated VM is reachable from any other host in the

datacenter, regardless of its location, and to avoid any reconfiguration of the VM's network settings (e.g., the default gateway address).

When a Virtual Machine is deployed for the first time, it is allocated in one of the available physical servers in a specific rack. This allocation choice assigns one or more IP addresses to the VM. These IP addresses will be kept for the entire VM lifecycle, even in case of migration. Such IP addresses are referred to as the VM's *Home Addresses*, and the *Home Addresses'* subnet is referred to as the VM's *Home Network*.

Each router involved in the IP address migration process is called a *Service Switch*. In particular, for a given VM, we refer to the router located at the edge of the rack in which resides the physical server where the VM is initially deployed as to the VM's *Home Service Switch*. Likewise, the router located at the edge of the rack hosting the server where the VM is going to be migrated, will be referred to as the VM's *Foreign Service Switch*.

By analogy with Mobile IP, a generic host communicating with a VM will be referred to as *Correspondent Node*. Making the simplistic assumption that a Virtual Machine presents a unique IP address, in order to access a given service implemented by the VM, a Correspondent Node sends packets to the VM's Home Address.

When the *IP address migration* process starts, a Care-of Address is generated in order to identify the new location of the migrated VM. Such Care-of Address is a new IP address that is assigned to the VM on the new IP subnet (the *Foreign Network*). Notice that this new address is not directly known by the VM, but only the datacenter's Service Switches are aware of it. Making the VM unaware of the Care-of Address enables the services on that VM to run without interruptions and without the need for reconfiguration even in case of migrations among different IP subnets.

Because we consider the problem of VM migration only within the limited scope of a single datacenter, where VMs are identified by private IP addresses, we do not concern about using an additional IP address for a migrated VM. In fact, the use of a whole class-A private IP network (10.0.0.0), combined with

subnetting, is largely sufficient even for large scale datacenters. Moreover, by using a VM-specific Care-of Address, we are also able to avoid the use of a tunneling layer to forward network traffic to migrated VMs, by implementing NAT functions at Service Switches level. In the following two subsections we describe in details how a Service Switch should operate at both L2 and L3 in order to guarantee a seamless connectivity for migrated Virtual Machines.

### Layer 3 operations

In figure 3.2 we show an example of IP address migration using NAT functions in Service Switches: an external node with IP address E sends packets to the datacenter's VM with IP address M, which is in the VM's Home Network (HN). Once the migration happens, the Care-of Address  $M^*$  is assigned to the migrated VM. This new address belongs to the VM's Foreign Network (FN), and is only used at the datacenter's Service Switches level, while the VM is totally unaware of it. Simultaneously, NAT rules are added to the R1 (Edge Service Switch), R2 (Foreign Service Switch), and R3 (Home Service Switch) Service Switches: R1 and R3 are instructed to transform M address into  $M^*$  address, so that packets destined to M are routed into the datacenter using the Care-of Address. When packets reach R2, a dual NAT rule is applied, transforming  $M^*$  again in M, hence the packet can be put on the FN and the migrated VM can take it, being unaware of the changed IP network. In addition to NAT rules, the R2 routing table is also updated to take into account the new location of the migrated VM, that is now directly reachable from one of its own interfaces.

### Layer 2 operations

To provide migration transparency we need to take into account both the VM's IP address migration and the VM's network configuration. Layer 3 operations for transparent address migration have been presented in the previous paragraph. Here we present operations needed at layer 2. Assuming that each VM knows its own Home Network (as this can be derived by combin-

ing the VM's IP address with the netmask) and knows its default gateway's IP address, we have to solve the connectivity cases shown in figure 3.3.

In the following we consider eight different communication cases. Each case is identified by a subsection number, that is consistently used in Figure 3.3 to illustrate the corresponding exchange of packets.

**C\* to E** To enable the communication of a migrated VM (C\* in figure 3.3) with a host E sitting in another subnet (that is different from C\*'s Home and Foreign networks), we have to consider the regular IP behaviour in the case of a communication between two hosts living in different IP subnets. The external node resides on a subnet that is different from the VM's one, so the VM's OS, looking at the subnet mask, realizes that the communication must happen through the default gateway. Because of the migration, the VM's default gateway (R2) is actually on a different subnet, and hence unreachable. To solve this problem in a transparent way, we enable the router on the foreign network (R1) to act as Proxy ARP, so that it responds to ARP request on behalf of R2. Hence, packets generated by the migrated VM destined to R2 are taken by R1, that in turn is able to correctly route them.

**E to C\*** The reverse path of the previous case is enabled using layer 3 operations at R3 and R1 as previously described.

**C\* to D** To allow the communication of the migrated VM with an host on the home network, we need again proxy ARP functionalities. The migrated VM tries to send packets destined to home network directly, because it considers home network's hosts as neighbours at layer 2. R1, in this case, must act as proxy ARP for the entire Home Network: each packet destined to the Home Network is taken by R1 and forwarded to the Home Network, using IP routing, where R2 can correctly perform the delivery.

**D to C\*** The communication from an host on the home network with the migrated VM (case 4) is performed configuring R2 to work as proxy ARP for

the migrated VM's *home address*. Because R2 is aware of the VM's *Care-of Address*, it performs NAT on the packets and send them to R1 using IP routing.

**C\* to B** Because the migrated VM is able to establish a communication with a Generic Host as of case 1, it is also able to communicate with a Host on the Foreign Network. If desired, the Service Switch R1 can use ICMP Redirect messages to allow the direct communication of the VM with the host on the foreign network.

**B to C\*** The communication of a foreign network's host with the migrated VM happens as in case 2. Once again, the Service Switch R1 can use ICMP Redirect messages to optimize the communication.

**C\* to D\*** Because both VMs are configured to talk directly to each other, the only operation needed at the Service Switch is to avoid its ARP proxying for the migrated VMs' addresses.

**C\* to D\*\*** This case can be resolved as a simple combination of the previous cases.

### 3.1.5 Evaluation

To evaluate the feasibility of our solution, we realized a software implementation using standard GNU/Linux tools such as *NetFilter/iptables* and Linux Kernel 2.6's proxy ARP support. The implementation has been evaluated on the testbed shown in figure 3.4, comprising four physical servers, one acting as external host, one acting as datacenter edge-router, and two playing the role of virtualization containers. Virtualization containers are physical servers configured with the Xen Hypervisor [14]. Each virtualization container is able to host one or more VMs. One of these VMs acts as a router, with one interface connected to the edge-router and another connected to a

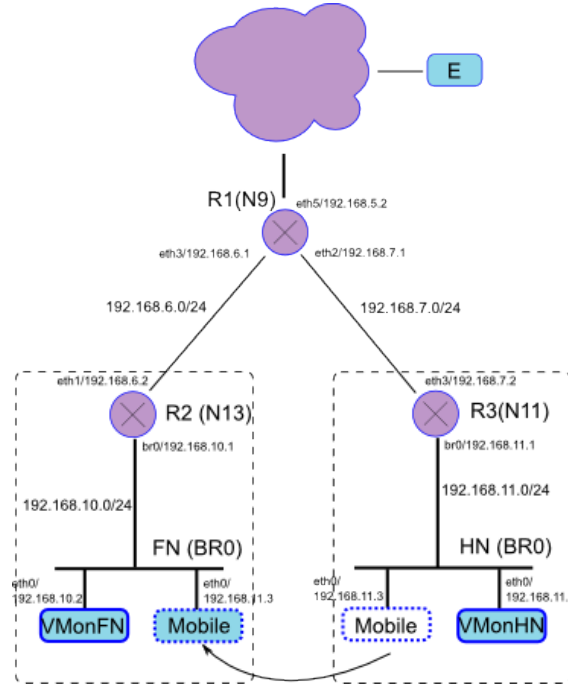


Figure 3.4: Testbed

virtual LAN. Virtual LANs are realized by means of the GNU/Linux software bridge, and connect all VMs hosted by a physical server.

In order to obtain the transparent migration of the “mobile VM”, as shown in Figure 3.4, from the Home Network 192.168.11.0/24 to the Foreign Network 192.168.10.0/24, we need to configure the testbed with 6 NetFilter rules, (two rules for each of the three routers R1, R2, and R3), and 2 additional IP routes, (one in R2, and another in R3). More precisely, R1 and R3 are instructed to perform:

- destination NAT translating 192.168.11.3 in 192.168.10.3;
- source NAT translating 192.168.10.3 in 192.168.11.3.

R2 is instructed to perform the opposite translations, with the rules:

- destination NAT translating 192.168.10.3 in 192.168.11.3;
- source NAT translating 192.168.11.3 in 192.168.10.3.

By applying these rules, a migrated IP address totally disappears from packet headers when packets traverse the datacenter's network on the path between access and core layers. It is worth noting that NAT rules applied at R1 are not strictly necessary: applying rules to just R2 and R3 resembles the plain Mobile IP operations (apart from the use of tunnels). Anyhow, by exploiting the full control of the whole datacenter's network, we can optimize the network traffic flow, performing an early redirection of packets, so to avoid packets go firstly to R3 and then to R2. Notice also that because the migrated VM must discover the "new" MAC address for its default gateway, communications may not work until the VM's ARP cache is renewed. (The same happens for hosts on the VM's Home Network, with the VM's MAC address). To make this process faster, i.e., not to wait for ARP cache expiration, one can force the renewal of the ARP cache entries by using gratuitous ARP messages.

To further evaluate our design, we performed two simple tests. In the first one we used ping to measure the mean round trip time (RTT) between an "external" host and the migrated VM, in three different scenarios: i) before migration; ii) after migration, without configuration of the core layer router (i.e. resembling the "classical" mobile IP configuration); iii) after migration, with core layer router configuration, as previously explained in this paper.

In the second test the migrated VM runs a web server. We measured the response time before and after the migration. We performed 10 http GET requests per second, for a period of 100 seconds, downloading a 1 MB file. Results for both tests are shown in table 3.1.

Notice that due to some configuration bias among machines used to perform our tests, response times are even better after the migration process. More complex migration scenarios, e.g. involving multiple migrated VMs, also were tested in our prototype, but cannot be described here in more details due to space constraints.

Ping test	Mean RTT
non migrated	2.573ms
migrated (scenario ii)	3.192ms
migrated (scenario iii)	2.034ms
Web Server test	Mean Response Time
non migrated	3.4ms
migrated (scenario iii)	3.1ms

Table 3.1: Tests results

### 3.1.6 Related work

The problem of properly re-designing the networking infrastructure of modern datacenters is today under the spotlight of several research groups and big companies. The main challenge and goal is to achieve the ability to assign any server to any service, a property called *agility* in [54]. To this purpose, a few papers have proposed innovative solutions aimed at radically changing the way the network infrastructure of a datacenter is built. For instance, Greenberg et al. propose in [54] an innovative architecture, called *VL2*, that is organized in a flat scheme and operates like a very large switch. VL2 claims to be able to organize any set of servers in the datacenter in a virtual layer 2 isolated LAN. VL2 can be implemented with commodity switches with layer 3 functionality, but it requires modification to the end-system networking stack and a flat addressing scheme, supported by a directory service.

The potential and the costs of live migration of Virtual Machines in Cloud-enabled large scale datacenters has been investigated in [149]. The experimental evaluation conducted by the authors of this paper shows that live migration needs to be carefully managed in SLA-oriented environments requiring more demanding service levels.

In the last few years, other papers have presented similar techniques aimed at allowing live migration of Virtual Machines across different IP subnets [86], [136], [68]. The solution presented in this paper applies to migration within a single datacenter and does not require any modification to the public Internet. Moreover, it does not pose any requirement on the addressing



scheme to be used in the datacenter, and does not require the establishment of IP tunnels.

As we pointed out in section 3.1.5, our solution can be easily implemented with current devices, as it employs standard layer 2 and layer 3 functions, such as IP NAT-ting and ARP proxying. For an efficient implementation of the required behaviour and an easier configuration management of the devices, OpenFlow, a newly proposed open standard API for datacenter devices [102][88], is a useful tool for the actual deployment of our solutions in real large-scale datacenters.

### 3.1.7 Conclusion

Engineering the networking infrastructure of modern datacenters for Cloud Computing is today a very important problem. Cloud-enabled datacenters, in fact, need advanced support for an integrated management of Virtual Machines. In this paper we propose an innovative solution, based on the coordinated use of NAT rules and ARP proxying, for the problem of transparently migrating Virtual Machines across multiple IP subnets within a single datacenter. Our approach can be easily implemented with current network devices without any modification to their hardware. Our initial prototype is completely implemented in software and makes use of standard layer 2 and layer 3 functions.

## 3.2 Global mobility

### 3.2.1 Context and motivation

Originally born as a cluster based network emulation system [41], as described already in section 1.2, NEPTUNE-IaaS is a software system developed at University of Napoli Federico II that allows interactive design of networked virtual infrastructures on geographically distributed datacenters, to help provisioning of “Infrastructures as a Service”. Our system consists of an interactive client/server software system used to provide users with the

possibility of describing and designing the desired virtual infrastructure and of a set of other components that make it possible for services deployed at a given datacenter to be transparently migrated in remote datacenters for load balancing or fault/disaster recovery. NEPTUNE-IaaS is based on the use of Xen for virtualization of computing elements. Xen features are also used to multiplex the communication resources (e.g. network interfaces) available in the cluster nodes among several logically distinct virtualized nodes. Transparent migration of Virtual Machines in NEPTUNE-IaaS is implemented through the adoption of Service Switching, a novel paradigm that aims at extending the concept of virtualization to network services, by decoupling service execution environments and their physical location.

### 3.2.2 NEPTUNE-IaaS

NEPTUNE-IaaS is a software system for provisioning of IaaS services. In the context of NEPTUNE-IaaS, a *Virtual Infrastructure* is a collection of Virtual Machines provided as a service to an end-user. Virtual Machines are deployed on a subset of a cluster's physical nodes and properly configured according to the user requirements in terms of computational resources, software configuration, virtual network topology, and so on. A Virtual Infrastructure presents at least one public IP address, that is used to make the infrastructure accessible from the public Internet (Entry Point). In general, public IP addresses are assigned only to a subset of the nodes of a Virtual Infrastructure. Other nodes are assigned private IP addresses and can be reached only through the Entry Point nodes. A typical Virtual Infrastructure comprises a NAT/firewall node and a set of backend service nodes, whose NICs are assigned private IP addresses. We will describe later in this paper that the necessity of supporting transparent migration of Virtual Infrastructures across geographically distributed datacenters calls for unique assignment of private IP addresses within a Service Switching domain.

To achieve higher degrees of scalability and resource efficiency, Virtual Infrastructures are instantiated by allocating multiple Virtual Machines onto

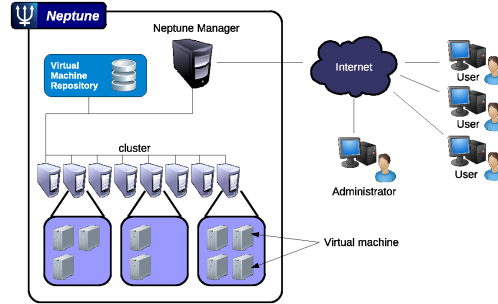


Figure 3.5: NEPTUNE architecture.

each of the cluster's real nodes (*node multiplexing*). Likewise, multiple virtual links are multiplexed onto the same shared physical link by associating each virtual link endpoint to a different virtual NIC (*link multiplexing*). Multiple fully isolated Virtual Infrastructures can be concurrently hosted by NEPTUNE-IaaS in the same datacenter, providing users with the illusion of having allocated a dedicated infrastructure.

### NEPTUNE-IaaS Architecture

A cluster managed by NEPTUNE-IaaS (Figure 3.5) is composed of three components: i) a set of worker nodes providing computational resources used to reproduce emulated networks, ii) a centralized repository providing storage space to worker nodes and iii) a front-end node, *Neptune Manager*. By NEPTUNE-IaaS we intend the whole collection of system software, of which the management software running in the Neptune Manager front-end is the most relevant part. All the physical components of the cluster are connected by two switched LANs, one for “control traffic” (e.g. node configuration) and another for “operational traffic” (i.e. traffic generated by users' applications).

### Virtual Infrastructure life-cycle

A Virtual Infrastructure life-cycle can be described by a Finite State Machine (Figure 3.6). A Virtual Infrastructure life-cycle begins with the definition of a virtual network topology. Once the topology is defined, the infrastructure can be allocated onto the cluster's physical nodes. On user demand, a run-

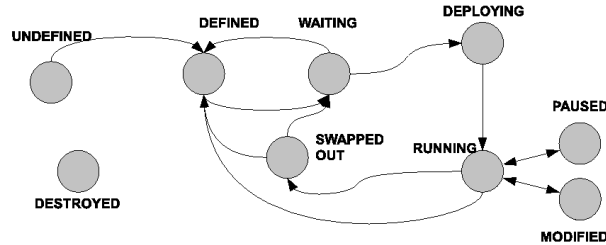


Figure 3.6: Virtual Infrastructure lifecycle.

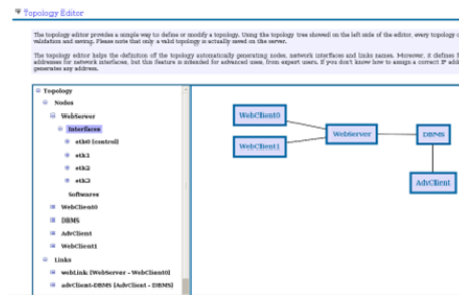


Figure 3.7: Interactive editor.

ning Virtual Infrastructure can be either suspended for future reallocation or definitively terminated. Allocation of infrastructures onto the cluster is made under control of system administrators, who need to explicitly accept users requests. Once accepted, an infrastructure’s topology allocation process starts. Such allocation process is automatic, involving tasks like virtual nodes mapping on cluster’s physical nodes and IP addresses assignments.

To define a Virtual Infrastructure, users can either write a topology description in a custom XML format, defining nodes’ properties (NICs, RAM, software configuration, etc.) and links’ properties (bandwidth, end points, etc.) or use an interactive graphic tool embedded into the web user interface (Figure 3.7). The tool assists the description of any node or link property suggesting available choices to the user. It is also possible for users to select pre-defined topologies for fast infrastructure definition. To define virtual nodes software configuration, users can access a “Virtual Nodes Template Images Repository” and select a VM template for each of the virtual nodes. VM templates can be modified and saved as new templates for reuse.

### Implementation details

Node multiplexing is implemented in NEPTUNE-IaaS by means of Xen [14]. Our current implementation relies on the libvirt virtualization API [125], making it feasible supporting different virtualization technologies in the future. The NEPTUNE-IaaS Management Node is responsible of managing Virtual Machines lifecycle.

Mapping of virtual nodes onto the cluster physical nodes is described by an allocation map which can be generated either manually by a system administrator or automatically, by means of a software module implementing a Lin-Kernighan derived optimization algorithm (described in Section 2.3).

When a virtual network is to be deployed on the physical cluster, Neptune Manager distributes Virtual Machine template instances to the physical cluster nodes. This distribution process is composed of two phases for each virtual node: 1) raw copy of the virtual machine image file containing VM template, and 2) VM creation on the target virtual machine monitor. During this last phase, virtual hardware resources are provided to the virtual node according to node definition provided by the Virtual Infrastructure topology description.

A major problem when dealing with the creation of virtual links is the need to assign IP addresses to both ends of virtual links, according to a general IP addressing scheme. NEPTUNE-IaaS provides an algorithm that automatically assigns subnets to links and IP addresses to their end-points. Furthermore, since several infrastructures can be running on the same shared infrastructure, this algorithm also ensures non overlapping of address spaces used by different infrastructures.

### 3.2.3 The Service Switching Paradigm

Service mobility is a key feature for new generation networks. In distributed service hosting environments, service mobility allows satisfaction of requirements like: efficient management of available resources, computational load balancing, service continuity even in presence of critical conditions. Service

Switching aims at extending the concept of virtualization to network services by decoupling service execution environments and their physical location [87]. Service instances in a Service Switching environment may be dynamically migrated across geographically dispersed datacenters, to achieve more efficient utilization of both network and computing resources. The Service Switching paradigm allows creation and management of *Service Execution Environments* across different datacenters with minimal impact on service continuity.

The architectural implementation of the Service Switching paradigm is centered around a main component, that we call *Service Switch*. Such a component is a network node that, in addition to the plain packet and/or flow switching capabilities, has more advanced features, including the ability to forward packets towards migrated Service Execution Environments. Service Switches can be located both at the edges of a network and in its core. Deployment of Service Switches in the core of the network of course requires cooperation of Internet Service Providers, but allows faster reconfiguration and migration of services.

Our current implementation of the Service Switching model relies on a combination of system-level virtualization technologies and of the Mobile IP model. In the following we firstly introduce a brief description of Mobile IP, and then the Service Switching architecture customized for the NEPTUNE-IaaS context.

IP version 4 assumes that the IP address of a node uniquely identifies its point of attachment to the Internet: a node must be located on the network indicated by its IP address in order to receive datagrams which are destined to it. IP Mobility Support (or Mobile IP) provides a mechanism which allows Mobile Nodes to change their point of attachment to the Internet without changing their IP address [114]. This mechanism relies on two intermediary entities: the *Home Agent* and the *Foreign Agent*. The role of the Home Agent is to maintain current location information of the mobile node, and to re-transmit all the packets addressed to the Mobile Node through a tunnel to

the Foreign Agent to which the Mobile Node is currently registered. The role of the Foreign Agent, in turn, is to deliver datagrams to the Mobile Node.

Service Switching allows services to be deployed at different geographic locations, each of which hosts a cluster of physical machines. A physical cluster is connected to the Internet through a special router, that we call *Edge Service Switch*. In the context of NEPTUNE-IaaS we are interested in transparently migrate a collection of related Virtual Machines (a Virtual Infrastructure, according to the definition we gave in Section 3.2.2). When a Virtual Infrastructure is deployed for the first time, it is associated to one of the available datacenters. This allocation choice assigns one or more public IP addresses to the Entry Points of the Virtual Infrastructures. These IP addresses will be kept for the entire lifecycle of the Virtual Infrastructure, even in case of migration. Such IP addresses are referred to as the Virtual Infrastructure's *Home Addresses*. The Edge Service Switch located at the edge of the datacenter in which the Virtual Infrastructure is initially deployed, will be referred to as the Virtual Infrastructure's *Home Service Switch*. An Edge Service Switch not only behaves as a normal IP edge router, forwarding incoming packets to the VMs hosted in the cluster and outgoing packets to a next hop router according to its current routing table, but it also implements specific traffic flow readdressing mechanisms to support service migration. Such mechanisms have been derived as extensions of the classical Mobile IP model. A generic end user terminal accessing a service will be referred to as *Correspondent Node*.

Making the simplistic assumption that a Virtual Infrastructure presents a unique Entry Point, in order to access a given service, a Correspondent Node sends packets to this latter, using the VI's Home Address as IP Destination Address. Incoming packets will be processed by the VM's Home Service Switch. In case a Virtual Infrastructure had to be migrated to a different datacenter, the Virtual Infrastructure's Home Service Switch creates an entry in its *Mobility Binding Table* (MBT in short) that contains information about the Entry Point of the migrated Virtual Infrastructure. The MBT keeps the

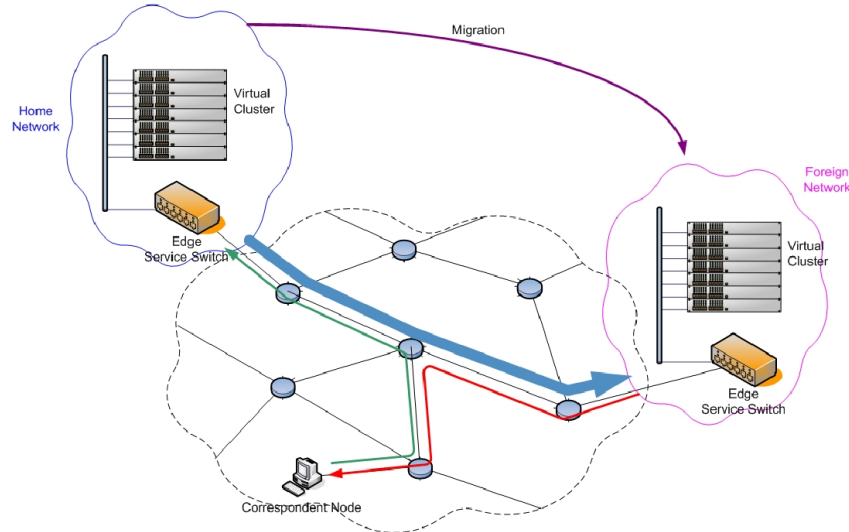


Figure 3.8: Tunneling mechanism implemented on the edge.

association between the VI's Home Address and the corresponding *Care-of Address*. Such Care-of Address is the IP address of the Edge Service Switch associated to the datacenter hosting the migrated Virtual Infrastructure, that we may call the Virtual Infrastructure's *Foreign Service Switch*. Migration of a Virtual Infrastructure is performed through a procedure that consists in updating the Home Network's MBT and in managing the migration of all the VMs belonging to the Virtual Infrastructure. Concerning the datacenter that hosts the migrated Virtual Infrastructure, apart from the configuration of the Foreign Service Switch, no other settings are needed. Migrated VMs keep using their own VI's Home Address as IP source address for outgoing packets, and Correspondent Nodes, being unaware of the migration, keep sending packets to the Virtual Infrastructure's Home Address. Once these packets reach the Home Service Switch, this latter forwards them to the Foreign Service Switch, by encapsulating such packets in a point-to-point tunnel (figure 3.8). The Foreign Service Switch, in turn, de-tunnels the incoming packets and delivers them to the migrated VM. As it happens in the Mobile IP scheme, reverse traffic is sent by the migrated VM directly to the Correspondent Nodes.



### 3.2.4 Related Work and Conclusions

In the last few months the term “Cloud computing” is transforming from a buzzword into real world engineering solutions and commercial products. In this paper we mention two established solutions that have some features in common with NEPTUNE-IaaS: Amazon EC2 and Eucalyptus.

Amazon’s *Elastic Compute Cloud* (EC2) [6] is an IaaS commercial system that first introduced the utility computing model, where computation, storage and bandwidth resources are rent on an as-needed basis. As well as NEPTUNE-IaaS, EC2 is based on Xen. Users select an Amazon Machine Image (AMI), including the machine’s software configuration from a set of AMIs proposed by Amazon, or create a new one from scratch. To each AMI instance (i.e. a Xen Virtual Machine) is associated an “instance type” that defines the resources of the machine in terms of CPU, RAM, HD. Resources are paid on a consumption basis: a machine is paid for each hour of activity, bandwidth is paid per-gigabyte of traffic and so on. Amazon provides two ways to access EC2 services: via a web interface or through web services. A complete set of tools and programming libraries are provided to access these service.

Eucalyptus [99] is an open-source cloud-computing framework, built to be interface-compatible with Amazon EC2: users can interact with Eucalyptus using same tools and interfaces that they use with Amazon EC2. Because the main goal of Eucalyptus is to provide a common open-source framework that enables researchers to do experiments and studies, even by replacing or modifying the implementation of system modules, the system is based on three components, each with a well defined Web-service interface. The software architecture is hierarchical: the base level is composed by Instance Managers (IM), responsible to manage virtual machines running on top of a physical machines, the middle layer contains Group Managers (GM), each of which manages a set of IMs residing on the same physical subnet. The top layer is the Cloud Manager (CM), that manages all the GM making high-level scheduling decisions and represents the entry-point to Eucalyptus for

users as well as for administrators.

NEPTUNE-IaaS has some features in common with both EC2 and Eucalyptus, but also some important differences. In particular, we want to highlight that NEPTUNE-IaaS provides tools to interactively design virtual networked infrastructures and supports transparent and efficient migration of infrastructures across geographically dispersed datacenters. NEPTUNE-IaaS is an ongoing project, whose future development include more complex management procedure to handle migration of complex virtual infrastructures in a reliable way. Integration of NEPTUNE-IaaS with storage services, such as those provided by Amazon's S3 are also being investigated.

# Chapter 4

## Issues in managing flexible resources

Flexible infrastructures provide several advantages and new features, but at the same time raise new challenges and issues. In this chapter we try to investigate two issues related to security and management of the flexible resources. In particular, we first try to understand how to monitor a cloud computing system to guarantee that the users are not exploiting the system to e.g., perform denial of service attacks. We try to apply well-known tools in the new context looking for possible shortcomings and opportunities and designing possible approaches. Moreover, we present a solution to manage the access to heterogeneous resources geographically distributed and managed by different authorities. This second study was conducted in the context of planetary network testbed provisioning, but we believe the findings are applicable to any other distributed system that manage geographically dispersed resources.

### 4.1 Security Issues

#### 4.1.1 Context and motivation

Cloud Computing is experiencing an impressive market growth, which could be indeed jeopardized by concerns about the risks related to potential misuse of this model aimed at conducting illegal activities. In fact, the cheap avail-

ability of significant amounts of computational resources can be regarded as a means for easily perpetrating distributed attacks, as it has recently been observed in several security incidents involving Amazon's EC2 cloud infrastructure [148], moreover, as it has been evidenced by recent issues related to Wikileaks [5], Cloud providers need also to take the issue of network neutrality into account, keeping provided services and distributed content on different layers. The sheer power of attacks from EC2 is indeed raising serious concerns in the community of system administrators and security experts [147]. Furthermore, evidences in recent research works have shown how it is possible to exploit some properties and features of a common cloud computing infrastructure, in order to perform attacks against competitors in an industrial scenario [127]. From the Cloud provider's perspective, threats should be faced, in order to protect Cloud customers, so, strategies for protecting the assets of customers must be designed, in order to grant the service level agreed upon. A Cloud provider should be able to face external threats, as well as internal attacks; he should protect its customers from attacks coming from the outside, and prevent its customers from damaging other network users.

Among the fundamental tools for defending computational and networking infrastructures from malicious behavior are Intrusion Detection Systems (IDS). In classical enterprise settings, an IDS is normally deployed on dedicated hardware at the edge of the defended networking infrastructure, in order to protect it from external attacks. In a cloud computing environment, where computing and communication resources are shared among several users on an on-demand, pay-per-use basis, such strategy is not effective: attacks may be originated within the infrastructure and also be directed against resources located within the cloud infrastructure itself. Hence, a proper defense strategy needs to be distributed. More generally, we envision a simple attack taxonomy based on the attack's source and target location. We define three attack types:

1. From "outside" to "inside" - the target of the attack is the Cloud

- provider or one of its customers, and the source is outside the Cloud [5];
2. From “inside” to “outside” - Cloud resources are exploited to attack an external target [148];
  3. From “inside” to “inside” - The attack is generated by a Cloud’s customer, and the target is the Cloud provider itself or another customer [127] of the same Cloud provider.

In this paper, we study the performance and impact of lightweight Network IDSs, with the aim of being able to detect each of the aforementioned attack types. We address the issue of detecting Denial of Service attacks targeting SIP-based systems; more in detail, we will address the issue of detecting SIP (Session Initiation Protocol) flooding attack instances targeting services hosted within a cloud. To this purpose, we study the impact of security tools deployment in different locations of the cloud, trying to expose the peculiarities of their employment in a cloud infrastructure. In particular, we will evaluate the discrepancies in management cost overhead, due to the employment of one among the possible deployment strategies for the selected security tools. We investigate the usage of both centralized and distributed strategies to detect and block attacks, or other malicious activities, originated by misbehaving customers of a Cloud Computing provider or by external nodes attacking cloud machines and services.

#### 4.1.2 Cloud computing

Despite its success, the Cloud Computing paradigm poses new challenges in terms of security of the computing infrastructure: cloud providers have the responsibility to manage a large infrastructure that hosts a number of highly dynamic virtual infrastructures operated by different users. Technologies like system virtualization have become for the first time widely adopted to offer computing resources as a service, allowing the dynamic spawn of virtual machines in the datacenter’s networking infrastructure.

## Security Issues

Resource rental on a per-usage basis shifts the responsibility of system management and administration towards specialized teams of experts, virtually reducing security risks typically due to system misconfiguration, lack of proper updates, or unwise user behavior. Despite that, the cloud computing paradigm introduces novel risks due to its inherent resource sharing requirement. Peculiar vulnerabilities, indeed, are introduced by the employment of virtualized host machines sharing common physical resources, by the availability of cheap large scale computation/communication/storage facilities and by the dynamicity of the cloud computing environment. Furthermore, misconfigured remote data storage can expose users' private data and information to unwanted access, or privacy infringement. Each of the security risks enumerated before needs to be dealt with by using a specific technique, trying to respond to the manifold known threats, as well as to the novel challenges which will emerge in the future. That is why an integrated approach, taking different aspects of security-related issues into account, is necessary in order to protect user data and preventing malicious actions both targeting cloud users and originating from within the cloud from being performed.

In the following, we will describe the Eucalyptus cloud computing architecture, trying to expose the security risks related to its employment. The deployment of an intrusion detection system in such a scenario will be described in details, and the resulting performance and computational overhead will be evaluated experimentally.

### 4.1.3 The Eucalyptus Cloud Computing System

Eucalyptus [64] is an open-source framework for cloud computing that implements the paradigm commonly referred to as *Infrastructure as a Service* (IaaS) [99]. Eucalyptus has been designed to be interface-compatible with one of the most popular commercial cloud service, namely Amazon EC2 [7]. The system is based on three components, each with a well defined Web-service interface. The software architecture has been organized according to

a three-level hierarchy. The bottom layer consists of Node Controllers (NC), responsible of managing virtual machines running on top of a physical machine. The middle layer contains Cluster Controllers (CC). Each CC manages a set of NCs residing on the same physical subnet. The topmost layer is the Cloud Manager (CM), that manages all the CCs and takes care of high-level resource scheduling. The Cloud Manager is the entry-point to the whole Eucalyptus system for end users as well as administrators. To create instances (the name given to virtual machines in the Eucalyptus and Amazon EC2 terminology) Eucalyptus supports both KVM [78] and Xen [14] virtualization technologies. In this work we will just take Xen into account, since it is the reference technology used also in Amazon EC2. Eucalyptus allows four different networking configurations, but among others, that are mainly targeted for testing environments or small installations, the most interesting for our purposes is the "Managed Mode". In *managed mode* Eucalyptus provides all the functionality present in Amazon EC2, including instances' subnetworks isolation. Network isolation is obtained through the use of VLANs [63], which impose appropriate configurations in data center's switches. Following the Eucalyptus terminology, each instance's network is referred to as a *security group*. Each user is bound to at least one security group, but association to multiple groups can be defined as well if needed. When configuring Eucalyptus for *managed mode*, the administrator must define an IP subnet entirely dedicated to the cloud. Moreover, the administrator must define the number of IP addresses available for each security group, actually defining how subnetting is performed. To guarantee access to external networks, each security group includes the cluster controller among its hosts. Instances are configured to use the CC as default gateway (Figure 4.1). The CC provides both DHCP and NAT services. NAT is realized using standard GNU/Linux's *netfilter* functionality. Features like *elastic IPs* are provided by means of rules configured on the CC's configured as a NAT. Eucalyptus exploits software bridges and Xen's virtual Network Interface Cards (NIC) to build virtual networks: when a security group is firstly created, i.e. when the first instance

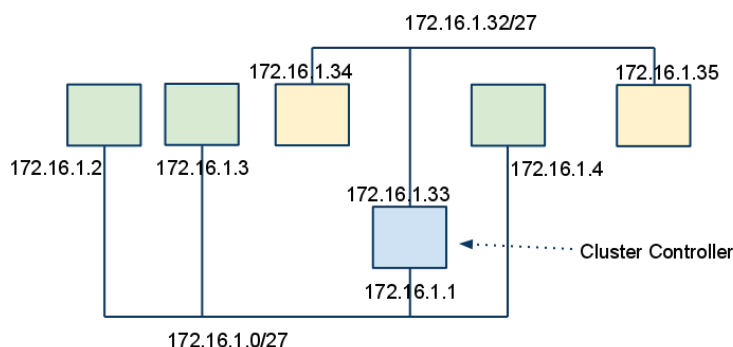


Figure 4.1: Eucalyptus managed mode networking: lv3 view

of a security group is allocated, Eucalyptus tags the physical NIC with the security group's VLAN tag and creates a software bridge for each physical machine; the bridge is actually created in the management virtual machine which starts at the boot of a physical machine. Such machine is usually named Dom0 in Xen context. The tagging process creates an abstract NIC to which tagged traffic is forwarded; such interface is then attached to the software bridge. Since Xen creates a new pair of "connected virtual ethernet interfaces", with one end of each pair in the virtual machine and the other end within Dom0, each newly created instance's virtual NIC that resides in Dom0 is attached to the corresponding security group's bridge (Figure 4.2). Access to security groups is controlled by the CC's firewall. By default, a security group is not accessible from external networks, and allowed traffic must be specified in terms of source network/address and port number through the Eucalyptus' API. E.g., in order to host a public web server in a security group, a rule to allow HTTP traffic from any network must be specified and added to the security group.

#### 4.1.4 Detecting Attacks in a Cloud Computing System

As stated earlier, cloud computing infrastructures have recently been the subject of technical news reports about severe attacks to several SIP-based communication infrastructures. What emerged by such reports about recent



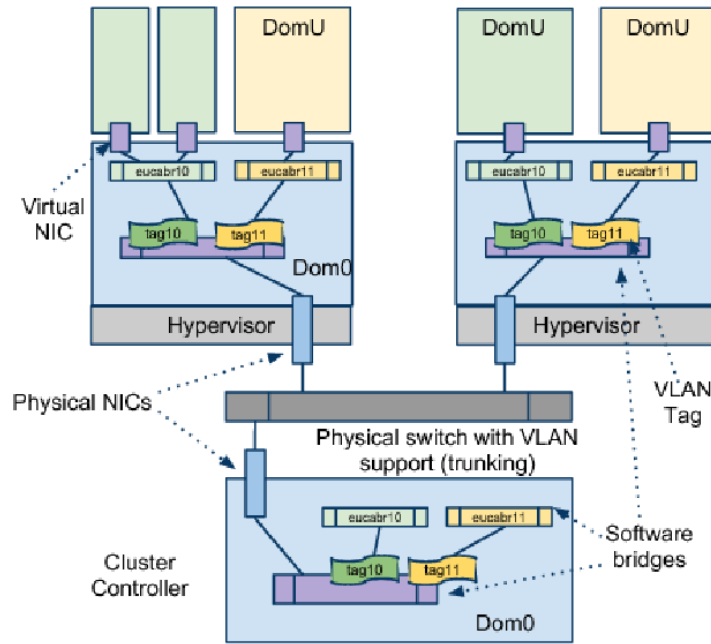


Figure 4.2: Eucalyptus managed mode networking: infrastructure view

security incidents is the lack of a structured and well organized security system deployment in cloud computing infrastructures. The aim of this paper is to present the deployment experience of a production level, state of the art solution for intrusion detection (Intrusion Detection System - IDS). According to the analyzed data source, IDS can be classified in network and host based. Network based IDS analyze traffic flowing through a network segment, by capturing packets in real time, and analyzing and checking them against some “classification” criteria. IDS can be further characterized with respect to the type of detection mechanism implemented. Namely, IDS can explicitly model attacks, anomalies and unwanted behavior, thus implementing the misuse-based detection paradigm, or conversely model normal and expected events, consequently detecting as anomalous what doesn’t conform to such “normality” model.

We have tried different deployment schemes, trying to exploit the advantages of distributed systems, and the inherent characteristics of a cloud

computing architecture. We used a network based, signature based IDS. The employed IDS is network based since we want to deal with network-based attacks, and try to detect them by observing network, transport and application layer activity of cloud customers and external users. Furthermore, we use a signature based IDS in order to show how the careful deployment of well known, already available solutions could mitigate a severe problem in such a distributed computing framework as cloud computing. We will evaluate the tradeoffs between computational overhead and granularity of analysis, in terms of detection capabilities, percentage of total traffic analyzed, and cpu and memory consumption, as well as packet loss.

### **Building the proposed architecture**

In this paper we will address the problem of deploying multiple instances of an IDS within a cloud computing system, allowing to rely on multiple security observation points. Eucalyptus, the cloud computing system used for the implementation of the experimental scenario, is characterized by the presence of a frontend, which also operates as a NAT, traversed by all traffic flowing to, and coming from, virtual hosts inside the cloud. Therefore, it is reasonable to think of an IDS installation which only relies on a single IDS installed near the frontend. Such an IDS would be able to see all the traffic related to virtual hosts hosted in the cloud, and provide a very good point of observation. The availability of such a large and significant amount of data, indeed, is obtained at the cost of high computational resources consumption. In fact, by forcing the analysis of all traffic to be performed at a single point, the machine physically hosting the IDS can be easily overloaded, losing packets, and producing inaccurate detection results. On the other hand, it is possible to deploy a network based IDS close to each of the physical machines. This configuration helps in reducing the load each IDS is subject to, thus helping to overcome the issue of packet loss. In fact, even though a virtual host can be subject to a Denial of Service attack, IDS's installed on other physical machines will not be affected, therefore preserving their detection

power intact.

### **Snort**

The IDS we chose to deploy in the proposed architecture is Snort [129]. Snort is a popular signature-based network intrusion detection system, mainly implementing the misuse based detection paradigm. Its modular architecture makes it easily extendable, and has fostered the integration of anomaly based detection plugins as well. In general, Snort's behavior is determined by rules, describing significant characteristics of events or specific attack signatures. Snort rules are organized in several groups, trying to separate them both in terms of the targeted attack type or application scenario. For the sake of efficiency, rules are mainly structured in two parts: a header part, and a body part. The rule header contains information about the type of action to perform when the rule is matched. Such actions include, but are not limited to, the possibility of generating an alert. Furthermore, the header contains information about source and destination IP addresses and ports, thus allowing to apply each rule to restricted subsets of the analyzed traffic flows. The rule body, instead, contains information about the type of action to perform on packets in order to check whether they match the rule; furthermore, it can also contain byte sequences to check against the packet's payload. Typically, attack signatures are searched for in the payload of packets, and rules matching the content of such payloads are logged using one of the many available logging facilities, alongside with information allowing the identification of the traffic flow transporting attack-related traffic.

In order to identify which rules have to be evaluated for a packet, a fast multi-pattern search for the longest content string of each rule of a packet's port group is performed on the packet's payload. If this initial string matching algorithm finds a potentially matching rule, other mandatory fields of the rule (e.g., source and destination IP addresses) are checked and, upon success, the optional conditions of that rule are validated. This processing can include an expensive pattern matching operation which uses all the key-

words of a rule and also validates their position. This two-phase approach has the advantage that not all rules need to be fully evaluated. Therefore, any deployment strategy allowing to reduce the load of each instance of the IDS, yet preserving the overall detection capabilities of the IDS ensemble, is worth investigating, since it can allow for a more effective detection of ongoing attacks.

### 4.1.5 Experimental evaluation

#### The reference testbed

In order to perform the experimental evaluation of the proposed attack detection scenario, we installed a testbed, depicted in figure 4.3 simulating the complete scenario for effective detection of SIP flooding attacks targeting hosts in a cloud computing environment. The testbed consists of six physical machines, two of these host a total of eight virtual machines, managed by Eucalyptus. The Eucalyptus controller plays the role of both a NAT traversed by traffic flowing to and from virtual hosts within the cloud, and as a cloud management station, controlling virtual hosts deployment and working as a concentration node for traffic on the configured VLAN's. In particular, in our experiments, we used two VLAN's, or in Eucalyptus' jargon two "security groups". As depicted in figure 4.3, we deployed two Asterisk [13] SIP

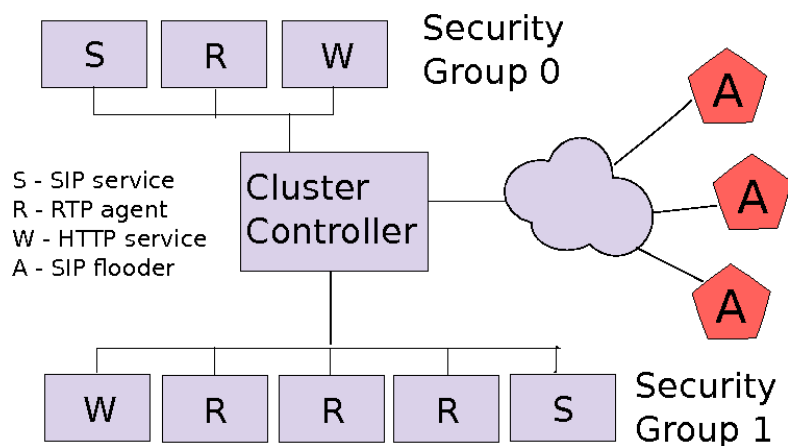


Figure 4.3: Experimental testbed - deployed services

servers, one for each security group, several RTP-using agents, as well as several attack instances generating SIP flooding traffic, implemented by using the “inviteflood” [57] tool. The aforementioned machines and services conveniently emulate the attack scenario, but are not realistic enough in themselves. For this reason we also deployed one Apache web server per security group, stressed by the hammerhead web stresser. This was useful for recreating a more realistic cloud scenario, where different services are likely to be hosted. In order to further differentiate traffic properties, we also installed D-ITG [74] as a background traffic generator. D-ITG inject traffic with specific properties in communication channels between a sender and multiple receivers; traffic can be of different types, and several properties such as inter packet time and packet size distribution can be configured. The implemented cloud consists of a machine implementing the frontend, and two physical machines implementing the physical nodes. Each of the physical nodes hosts four virtual machines, each hosting one testbed component, as represented in figure 4.4.

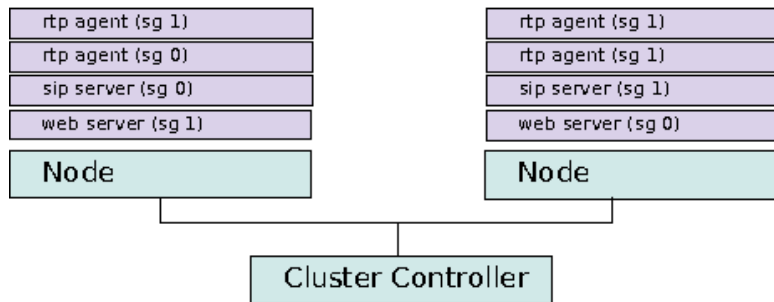


Figure 4.4: Distribution of services in nodes

## Experimental evaluation

In order to evaluate the dependency of IDS performance on its position in the network, we imagined two different test scenario. In the former, we installed the IDS close to the Cloud Controller, thereby allowing it to sniff and analyze all the traffic flowing to and coming from virtual hosts. At the Cloud Controller’s side, VLAN tags are removed by the virtual bridge, as

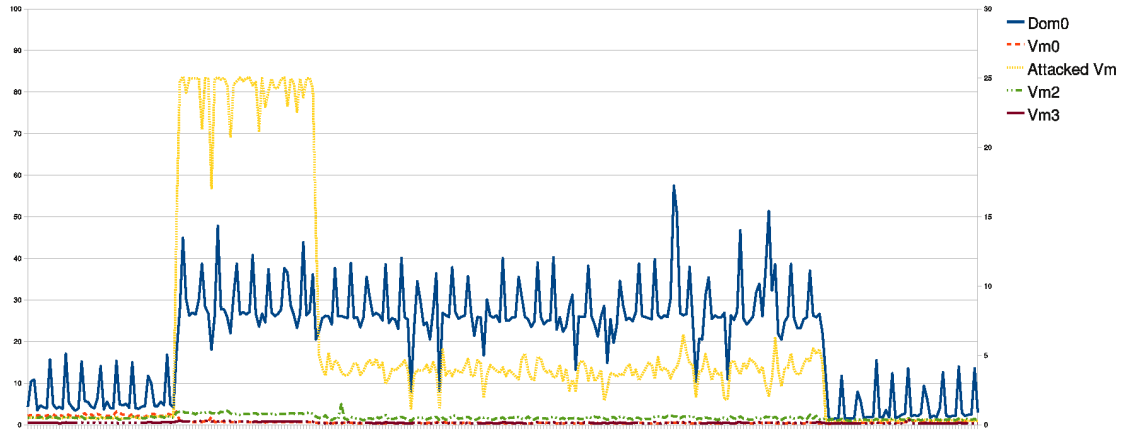


Figure 4.5: Attacked physical machine CPU load

discussed in previous sections. Therefore, all traffic is visible and the correspondent VLAN is indistinguishable. In the latter, instead, a Network IDS has been installed close to each of the two physical machines. Since each physical cloud node can potentially host virtual hosts belonging to different security groups, the IDS has to perform VLAN tag stripping, before being able to correctly analyze each packet. For each configuration, we evaluated the detection capabilities of the IDS with respect to the selected INVITE flooding attack. In both cases, the IDS's were able to correctly detect attack instances, issuing alerts communicating the result of packet analysis. Yet, it is interesting to point out how "expensive" such detection process is, showing some interesting properties and giving some insight. In particular, we observed CPU usage, in order to show whether the system hosting the IDS still has some resources to dedicate to detection during a flooding attack. Such an evaluation is useful since modern attacks consist very often in coordinated actions aimed at hitting big targets and totally disrupting networks and services. Furthermore, in the case of cloud computing, where the typical customer can be a company outsourcing service hosting, unfair competition between enterprises can become a good motivation for perpetrating dramatically effective attack campaigns. The first thing to point out in the tests' result discussion is that in both scenarios we were able to detect

that a SIP flooding attack was in act. We are confident that such a result is caused by the relatively small impact of the attack itself, which was indeed able to saturate the resources of the SIP server, but not our cloud's physical resources. In figure 4.5 we show the CPU load of the physical machine hosting the virtual machine containing the SIP server under attack. The graph clearly shows a significant increment in CPU usage due to both the presence of the virtual machine and of the administrative domain (Dom0). This remains true until the Dom0 is able to perform both packet forwarding actions (i.e. forwarding packets from the physical NIC to the virtual machine's virtual NIC) and packet analysis through the IDS. When Dom0 reaches its physical performance limit, it is no more able to forward packets to the attacked virtual machine, and that's why such machine's CPU load decreases significantly. Clearly, the shown Dom0 performance pattern is caused by the absence of countermeasures subsequent to attack detection. It is worth pointing out that during the attack, other virtual machines running concurrently with the attacked one underwent a performance degradation, because of the Dom0's overload. On the other hand, the second physical machine was totally unaffected by the attack. Looking at the second scenario, where the IDS is deployed close to the Cluster Controller, we must take into account that each performance degradation is reflected on the entire cloud. Figure 4.6 shows the impact of running Snort co-located with the Cluster Controller. The CPU "system-level" load is caused by the packet forwarding activity, while the "user-level" load is mainly caused by the IDS's activity. During the attack the IDS uses a double amount of CPU time with respect to the system's CPU time, even though our attack instance is not very powerful. Since an overloaded Cluster Controller is a bottle-neck for the cluster, we should avoid to add such big load on it. Even installing the IDS on a separate machine next to the Cluster Controller would result in an overloaded machine, since it should analyze all the traffic, therefore being prevented from operating properly.

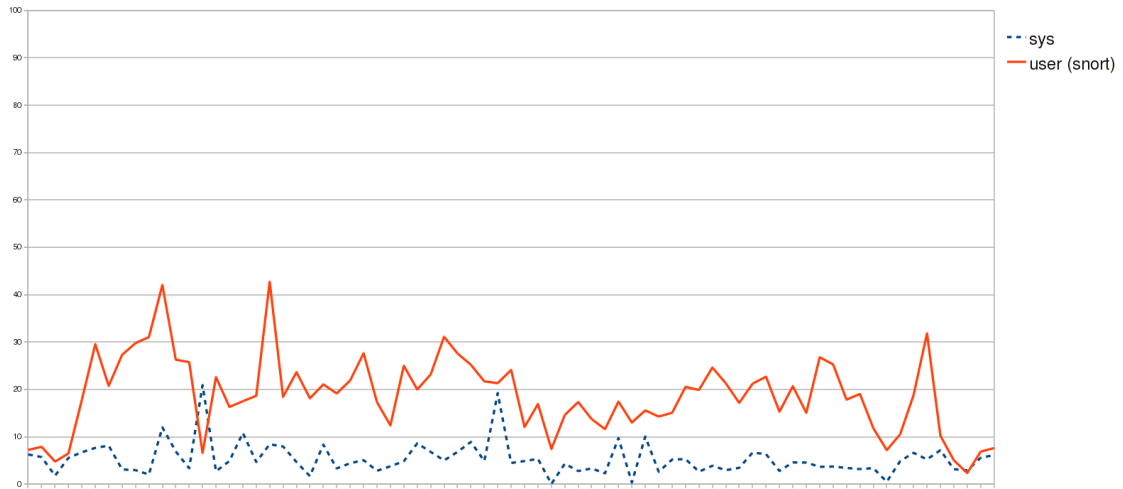


Figure 4.6: Cluster Controller CPU load

#### 4.1.6 Related works and Conclusions

The application of IDS in cloud systems is a new research field that is gaining interest due the spread use of cloud computing services and the increasing number of both attacks targeting cloud services and originating from inside a cloud computing infrastructure, exploiting it as an infrastructure for deploying attacks. Due to the young age of this research field there are a few papers on the topic. Current researches are mainly targeted at defining a new IDS model that can take advantage from additional information provided by the Cloud Infrastructure itself. In [145] an example of a distributed IDS for cloud environments is presented. The proposed IDS is designed to work in a Cloud system providing services according to the Platform as a Service paradigm, and is structured as an added service of the cloud system's infrastructure. Further works can be found as applied at computational GRIDs. In these works the reference system architecture is somewhat different from Cloud, but there are also some similarities, and hence there are solutions that could be applied in the cloud context as well. E.g., in [140] a solution based on the analysis of data gathered from traditional IDS and monitoring systems (e.g. Snort) deployed in the GRID's network is described.



In this paper we have shown the practical implementation experience of different deployment strategies of a well known IDS in a cloud computing system, in order to provide a fast and cheap solution to the intrusion detection problem in cloud environments. Depending on the deployment choice, several benefits and shortcomings have been pointed out and discussed. A single IDS could be placed close to the Cluster Controller, being able to monitor all traffic flowing to and from the cloud computing infrastructure. In this scenario, the single IDS is heavily overloaded, thus allowing for coordinated attack actions to disrupt the IDS's functionality by means of specifically crafted traffic before starting the real attack. On the other hand, by deploying IDS's next to the physical machines, each IDS would be able to control a smaller portion of traffic, thus being hardly overloaded. This deployment scenario needs a properly designed correlation phase in order to gather meaningful information from different security tools spread across the monitored network, which will be the subject of future work. The choice of the best deployment strategy, obviously, depends on the characteristics of the application scenario, and on the administrator's and users' requirements.

## **4.2 Resources access management**

### **4.2.1 Context and motivation**

The ultimate success of the Wireless Mesh Network paradigm (WMN) in large scale deployments depends on the ability to test it in real world scenarios [2]. Due to the inherent difficulty of capturing all the relevant aspects of the real behavior of these systems in analytical or simulation models, research on WMNs has always heavily relied on experimental testbeds. In fact, the creation of such experimental testbeds has been an active area of research in wireless mesh networking over the last ten years [23]. However, it is difficult (and costly) to setup a large-scale wireless mesh testbed to experiment with new applications, services and protocols. Also, wireless mesh networks are usually employed as access networks to the Internet, hence testing new

solutions thoroughly requires to take the complexity of the real Internet into account.

To allow for a realistic evaluation of new applications, services and protocols specifically designed for wireless mesh networks, we analyzed the existing projects that enable to share and manage testbeds and resources over a large geographic area. On the one hand, PlanetLab is universally known to be an open platform to conduct realistic experiments on a planetary scale [36]. On the other hand, OMF (*cOntrol and Management Framework*) is a well-established software platform that supports the management and automatic execution of experiments on a networking testbed. Originally developed for the ORBIT wireless testbed at Winlab, Rutgers University [124],[108], OMF is now deployed in several testbeds in Australia, Europe, and in the U.S. [123].

In this section we present a contribution towards the interconnection of geographically distributed OMF-based wireless testbeds through PlanetLab. Our approach allows the making of experiments involving the use of resources provided by a local wireless testbed in combination with other resources provided by other remote sites connected to the PlanetLab planetary-scale testbed. This allows running experiments on wide-area infrastructures, involving several kinds of technologies, both in the core of the network, where they cannot be controlled by experimenters, and at the edges, where they can be selected to compare several kinds of access networking technologies, such as WiFi, WiMAX, UMTS, Wireless Mesh Networks.

Heterogeneous testbed infrastructures have many similarities with other systems e.g., cloud computing ones, since they provide flexible infrastructures with an on-demand access paradigm. The set of issues and problems that have to be solved in this context are, hence, quite similar to the ones faced for providing more general flexible infrastructures.

The contribution we present into this section is in line with current ongoing efforts towards the so called “federation” of experimental infrastructures. A testbed federation has been recently defined as the interconnection

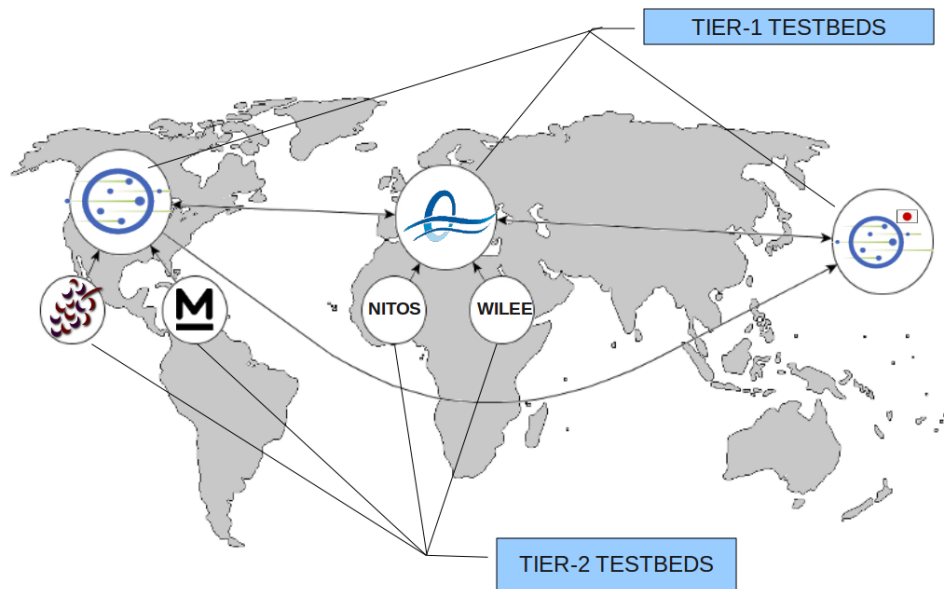


Figure 4.7: Hierarchical federation of heterogeneous testbeds.

of two or more independent testbeds for the creation of a richer environment for experimentation and testing, and for the increased multilateral benefit of the users of the individual independent testbeds [82] and it currently appears as the most reasonable way to build large-scale heterogeneous testbeds. Roadmaps envisioned by the most significant research initiatives focusing on future research infrastructures, such as GENI [51], [44] and FIRE [50], assign a key role to federation of existing testbeds. Actually, we envision a hierarchical federation model, as depicted in Figure 4.7, in which global scale Tier-1 testbeds, federated among them in a peer-to-peer way, act as “aggregators” of local Tier-2 testbeds. In this view, we assume PlanetLab and PlanetLab Europe as existing Tier-1 testbeds, whose federation is already in place and operational since 2008.

Federation of heterogeneous testbeds involves a number of both technical and organizational issues. With regards to the technical challenges, they

comprise the problem of sharing user credentials, as well as armonising usage models and resource management policies among testbeds. Our contribution accounts for such problems and we will describe hereinafter how we dealt with them. Thus, our contribution can be viewed as a preliminary effort in the direction of the *federation* of two different kinds of testbeds that we feel are of extreme importance for researchers working on wireless mesh networks.

In particular, in this section we present how we integrated some basic mechanisms for accessing the resources provided by a OMF-based wireless testbed from a PlanetLab node co-located with the OMF-testbed. Our system allows the seamless integration of the OMF-resources into the global scale PlanetLab infrastructure, creating a synergic interaction between the two environments. In particular, thanks to our contribution PlanetLab users may run experiments involving resources provided and controlled by the OMF wireless testbed.

The rest of the section is organized as follows. In paragraph 4.2.2 we briefly describe the architecture of PlanetLab, its usage model and resource management techniques. Likewise, in section 4.2.3 we briefly describe the architecture of OMF, its usage model and resource management techniques.

In paragraph 4.2.4 we describe the integration steps that we developed to allow for distributed experiments running at two PlanetLab sites and involving two wireless mesh testbeds, both based on OMF. In particular, we describe a software system that is able to manage resource scheduling for both resources included in the OMF-based testbeds and in the PlanetLab nodes.

In paragraph 4.2.5 we describe the two OMF-based testbeds involved in our validation experiments: the NITOS wireless testbed located at the University of Thessaly and the WILEE testbed located at Univ. of Napoli in Italy.

In paragraph 4.2.6 we illustrate how we used the integrated testbed setup to conduct an experiment aimed at evaluating a peer-to-peer traffic optimization technique. This is a typical distributed experiment in the PlanetLab

wired environment, but in our case it involves the usage of a wireless mesh as an access network, which would not be possible by means of the resources made available in PlanetLab.

In paragraph 6.2.5 we compare our contribution against similar integration efforts that have been proposed in the past years.

#### **4.2.2 PlanetLab: architecture, usage model and resource management**

The most relevant large scale distributed testbed for networking research as of today is PlanetLab [36]. PlanetLab is a geographically distributed testbed for deploying and evaluating planetary-scale network applications in a highly realistic context. Nowadays the testbed is composed of more than 1000 computers, hosted by about 500 academic institutions and industrial research laboratories. One of the main limitations of PlanetLab, however, is its lack of heterogeneity. Nearly all PlanetLab nodes are server-class computers connected to the Internet through high-speed wired research or corporate networks. As a consequence, it has also been noted that the behavior of some applications on PlanetLab can be considerably different from that on the Internet [77], [122]. Several efforts have been done in the last few years to add different kinds of networking technologies to PlanetLab (e.g. UMTS integration in PlanetLab is described in [24]) or to integrate new kind of terminals (e.g. the integration of non-dedicated devices made available by residential users is described in [42]). However, it is now clear that PlanetLab can be usefully complemented by a variety of other testbeds, in particular when experimentation with wireless technologies is required.

##### **Architecture**

Figure 4.8 shows a conceptual view of the current architecture of the PlanetLab testbed, whose node set is the union of disjoint subsets, each of which is managed by a separate authority. As of today, two such authorities exist: one is located at Princeton University (PLC) and the other is located at Uni-

versité Pierre et Marie Curie UPMC in Paris, France (PLE). An experiment in PlanetLab is associated to a so-called *slice*, i.e. a collection of virtual machines (VMs) instantiated on a defined subset of all the testbed nodes. Each testbed authority hosts an entity called *Slice Authority* (SA), which maintains state for the set of system-wide slices for which it is responsible. The slice authority includes a database that records the persistent state of each registered slice, including information about every user that has access to the slice [116].

Testbed authorities also include a so called *Management Authority* (MA), which is responsible of installing and managing the updates of software running on the nodes it manages. It also monitors these nodes for correct behavior, and takes appropriate action when anomalies and failures are detected. The MA maintains a database of registered nodes at each site. Each node is affiliated with an organization (owner) and is located at a site belonging to the organization.

### Usage model

To run a distributed experiment over PlanetLab, users need to be associated with a *slice*. Slices run concurrently on PlanetLab, acting as network-wide containers that isolate services from each other. An instantiation of a slice in a particular node is called a *sliver*. Slivers are Virtual Machines created in a Linux-based environment by means of the VServer virtualization technology. By means of so-called *contexts*, VServer hides all processes outside of a given scope, and prohibits any unwanted interaction between a process inside a context and all the processes belonging to other contexts. VServer is able to isolate services with respect to the filesystem, memory, CPU and bandwidth. However, it does not provide complete virtualization of the networking stack since all slivers in a node share the same IP address and port space. The adoption of VServer in PlanetLab is mainly motivated by the need of scalability, since up to hundreds of slivers may need to be instantiated on the same physical server [138]. Figure 4.9 shows the internal view of

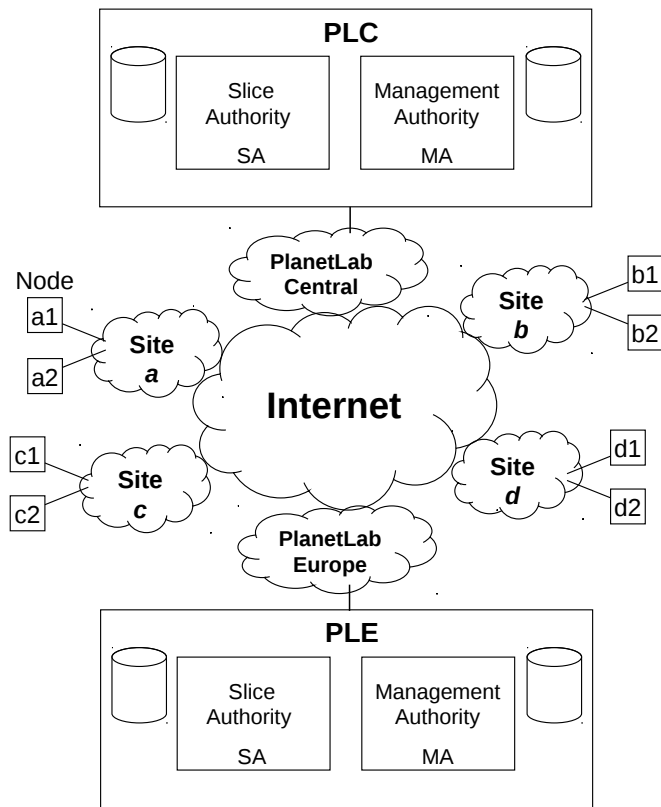


Figure 4.8: Conceptual PlanetLab architecture.

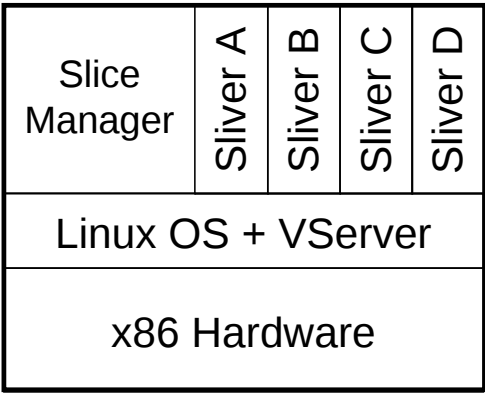


Figure 4.9: Internal view of a PlanetLab node.

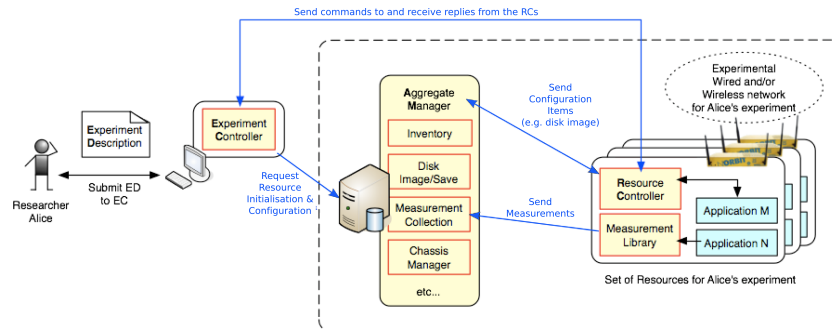


Figure 4.10: OMF architecture overview.

a PlanetLab node.

## Resource management

In PlanetLab, slice creation and resource allocation are decoupled. When a slice is first created, a best effort service is associated with it and resources are acquired and released by the slice during its entire lifetime. Therefore, slices are not bound to sets of guaranteed resources. Such an approach has been deliberately chosen in the original PlanetLab design. PlanetLab, in fact, has not been designed for controlled experiments, but to test services in real world conditions [110], [117].

### 4.2.3 OMF: architecture, usage model and resource management

OMF (*Control and Management Framework*) is a Testbed Control, Measurement and Management Framework. In the following of this section we will briefly describe OMF architecture, usage model and resource management. We also describe how experiments may coexist in the same OMF testbed, thanks to the NITOS scheduler.

#### Architecture

The components of OMF (Fig. 4.10) work together to automatically perform all the phases needed to execute the experiment, from the provisioning of



resources to the collection of experimental data. The most important component is the *Experiment Controller* (EC), which is also the interface to the user. It accepts as input an experiment description and takes care of orchestrating the testbed resources in order to accomplish the required experiment steps. It interacts with the *AggregateManager*, the entity responsible of the resources of the testbed as a whole, and provides some basic services to the EC, such as checking the status of a node, rebooting a node, etc.

The EC also interacts with the *Resource Controllers* (RCs) installed on the testbed nodes. These latter entities are responsible of performing local configuration steps, e.g. configuring the channels on the WiFi interfaces, and of controlling the applications, e.g. the traffic generator. The communication between the EC and the RCs is based on a publish/subscribe paradigm, where the EC publishes the messages on a XMPP server [130] and the RCs pick the messages addressed to them.

An important companion library of OMF is OML (*OMF Measurement Library*), which is used to automatically filter and collect experiment data on one or more measurement servers. OMF is able to instrument the OML library, in order to configure and guide the collection of experiment data.

## Usage model

In order to perform an experiment, users have to gain access to the machine hosting the Experiment Controller (EC). The execution of an experiment can be requested to the EC by submitting an experiment description in the domain-specific OEDL language, which is derived from Ruby. The experiment description usually consists of two parts: i) a *declarative part*, where the set of required resources and applications, with their configuration, are given ; ii) an *imperative part*, where the actions to be performed on the testbed are stated, e.g. the starting of an application. Execution of actions can depend on events which are defined by the platform, e.g. all the nodes are up and running.

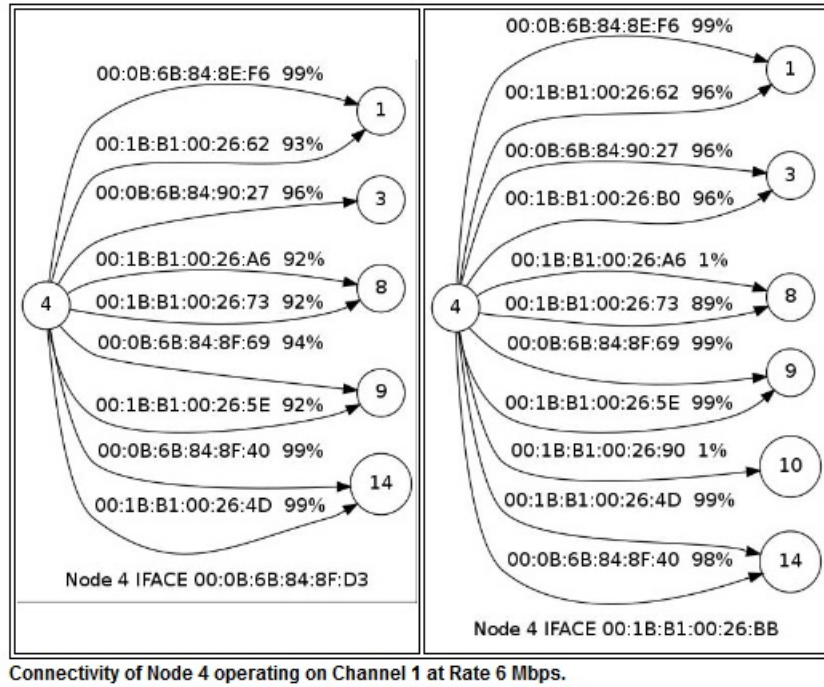


Figure 4.11: Link Quality for node 4.

## Resource management

OMF, in its basic form, assigns resources to users following a FCFS strategy: the user supplies an experiment description and the system tries to assign the resources requested by the experiment if they are available.

OMF can be customized, though, to support some kind of reservation of resources. In ORBIT a Scheduler interface is provided to support the reservation of the entire testbed. The user books the testbed in advance and during the reserved time slot is the only one allowed to log into the testbed console, i.e. the machine which hosts the Experiment Controller, and run his/her experiments.

In the NITOS and WILEE testbeds a different Scheduler, i.e. the NITOS Scheduler, is employed. Differently from ORBIT, different users can perform experiments in parallel on the same testbed. This is achieved by assigning a different subset of nodes and wireless channels to each user. These subsets

are reserved in advance through the Scheduler and the access to them is enforced during experiment time so that users can have access only to the resources, i.e. nodes and wireless channels, they had previously booked. To achieve that, modifications to OMF were required, as explained in the following section.

### **The NITOS scheduler**

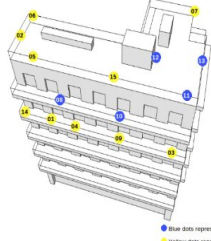
Currently OMF does not include any scheduling algorithm to synchronize the execution of experiments. Also, permissions to access the testbed resources are not checked. However, in a public, multiuser environment, we need a system that is able to assign resources only to the users that have the right to use them, while providing the experimenters with a way to specify the resources that they need for their experiments. In our work, resources are divided in two categories: nodes and spectrum. Thus, we provide a tool which is used by the experimenters to reserve nodes and spectrum for a specified time interval (whose duration must not exceed some limit). By slicing, we mean the partitioning of the testbed based on some criteria. With spectrum slicing, we aim to partition the testbed into smaller, virtual, testbeds which are using different spectrum and, hence, they do not interfere with each other in the entire testbed infrastructure. Using spectrum slicing, our tool makes the testbed available to users who would like to use different resources (spectrum, nodes) at the same time [10]. For example, many users can use the testbed simultaneously since we can allocate a particular group of channels to a group of nodes that can be assigned to one user.

### **The NITOS Connectivity Tool**

Before describing the NITOS Scheduler and how users select nodes and frequencies, we briefly present a NITOS tool that provides updated information on the channel link quality in order to help users decide which nodes are the most appropriate for their experiments. Since the NITOS testbed deployment is not RF isolated, the link quality between any pair of nodes may

Welcome to the NITOS Testbed Reservation Tool

Current Server Time: 2010-05-27 00:18:00



● Blue dots represent Disabled Nodes  
● Yellow dots represent On-line Nodes

Click on a date to select the day you want to start the reservation.

May 2010						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
						01
02	03	04	05	06	07	08
09	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Choose the time you want to start the reservation and for how long (max 4 hours).

**Select Start Time and Duration**

Start Date: 2010-05-30    yyyy/mm/dd

Start Time: 12 :00    hh :mm

Duration: 2    hours

(a) Fill in date and time

---

Available Nodes and Channels between 2010-05-30 12:00:00 and 2010-05-30 14:00:00 :

Nodes	802.11a	802.11b/g	
	Channel 36: <input type="checkbox"/>		
	Channel 40: <input type="checkbox"/>		
Node 1: <input type="checkbox"/>	Channel 44: <input type="checkbox"/>	Channel 1: <input type="checkbox"/>	
Node 2: <input type="checkbox"/>	Channel 48: <input type="checkbox"/>	Channel 2: <input type="checkbox"/>	
Node 3: <input type="checkbox"/>	Channel 52: <input type="checkbox"/>	Channel 3: <input type="checkbox"/>	
Node 4: <input type="checkbox"/>	Channel 56: <input type="checkbox"/>	Channel 4: <input type="checkbox"/>	
Node 5: <input type="checkbox"/>	Channel 60: <input type="checkbox"/>	Channel 5: <input type="checkbox"/>	
Node 6: <input type="checkbox"/>	Channel 64: <input type="checkbox"/>	Channel 6: <input type="checkbox"/>	
Node 7: <input type="checkbox"/>	Channel 100: <input type="checkbox"/>	Channel 7: <input type="checkbox"/>	
Node 8: <input type="checkbox"/>	Channel 104: <input type="checkbox"/>	Channel 8: <input type="checkbox"/>	
Node 9: <input type="checkbox"/>	Channel 108: <input type="checkbox"/>	Channel 9: <input type="checkbox"/>	
Node 10: <input type="checkbox"/>	Channel 112: <input type="checkbox"/>	Channel 10: <input type="checkbox"/>	
Node 11: <input type="checkbox"/>	Channel 116: <input type="checkbox"/>	Channel 11: <input type="checkbox"/>	
Node 12: <input type="checkbox"/>	Channel 120: <input type="checkbox"/>	Channel 12: <input type="checkbox"/>	
Node 13: <input type="checkbox"/>	Channel 124: <input type="checkbox"/>	Channel 13: <input type="checkbox"/>	
Node 14: <input type="checkbox"/>	Channel 128: <input type="checkbox"/>		
Node 15: <input type="checkbox"/>	Channel 132: <input type="checkbox"/>		
	Channel 136: <input type="checkbox"/>		
	Channel 140: <input type="checkbox"/>		

(b) Resources Availability

Figure 4.12: Resources Reservations

unexpectedly vary at any point in time due to external interference. For this reason, the static distribution approach, that is used in RF isolated wireless testbeds [107], is not efficient for these deployments. Therefore, there is the need for updated information in terms of measurements of link quality, that will bring a more accurate channel quality estimation.

A management tool called NITOS connectivity tool has been developed for assessing channel quality information and measuring channel connectivity among Wi-Fi interfaces. We have implemented the NITOS connectivity tool based on TLQAP [See [9]], which is a protocol used to assess the con-

nectivity and the quality of a link by estimating the packet delivery ratio (PDR) for all requested channel, rate and transmission power combinations. Specifically, TLQAP builds a measurement history log, creates a channel utilization profile and stores that information in a database that is used for link quality information retrieval by the NITOS connectivity tool.

The NITOS Connectivity Tool is comprised of three entities: a web interface, a database and a set of .dot scripts. Through the web interface, the user selects a node he/she wants to use in the experiment, an operating frequency (among those specified by the IEEE 802.11a/b/g standards) and a transmission rate. The database storing the information on the channel link quality (that is periodically updated by running TLQAP) is queried to retrieve the requested information. The result (a set .dot files) is presented to the user through a set of graphs, each of which is related to a Wi-Fi interface of the selected node. Fig. 4.11 shows the graphs corresponding to the two wireless interfaces of node 4. Each graph shows the links between a wireless interface on the selected node and the interfaces of the neighbor nodes. Upon each link, the MAC address of the neighbor's interface and the PDR of the link are reported.

### Scheduler Scheme

Slices are dynamically created on the testbed upon the user reservation. A user first reserves nodes and channels for a specified time range and then logs in to the testbed and executes his experiments. Once the reservation procedure is concluded, the system is aware of the resources that the user needs and the time range that he will keep them. During this time range, no other user can use any of the reserved nodes or the reserved channels.

Existing Wi-Fi testbeds open to the public only allow exclusive reservations in a given time period, i.e., only one user can use the whole testbed. Our scheduler instead allows multiple users to share the testbed at the same time. Indeed, the scheduler guarantees that they use distinct nodes and distinct frequencies, so that their experiments do not interfere with each other.

We now describe the reservation procedure. First of all, the user has to set the date and time that he would like to reserve a slice. The time is slotted with each slot duration set to 30 minutes. Then, he checks for the available resources in terms of nodes and channels. Fig. 4.12(a) shows a user checking for available nodes on May 30, 2010 for 2 hours starting at 12:00 pm. Also, a map of the building is shown, in order to give the user a better perspective of his reservation.

The scheduler keeps all reservations in a database. A reservation is a set of nodes, channels and a time range. When a user checks for available nodes, the scheduler searches its database for any possible record in the time range that the user specified. Then, it only returns the available set of nodes and channels, i.e., the nodes and channels that have not been selected by any other user in the specified time range (Fig. 4.12(b)). In this way, the system ensures that both the time and the frequency division requirements will be met. After the user has made and confirmed its selection, the scheduler database is updated. From this point on, the scheduler is responsible for ensuring that the user will only use the reserved slice for the specified time period.

### **OMF extension to support NITOS scheduler slicing features**

The scheduler mainly consists of two parts: a user interface, which is responsible for guiding the user through the reservation process making sure that he does not make a reservation conflicting with reservations made by other users, and a system component, which controls the slices by ensuring that this user's experiments will only use the reserved resources. The user interface role has been illustrated in the previous subsection, while the system encapsulation of the scheduler will be illustrated in this subsection.

So far we have described the part of the scheduler which is focused on the experimenter and his choices at reservation. However, we also need to ensure that the experimenters will stick on their choices and, even if they try, the system will not allow them to use any resources that they have not reserved.

For this purpose, we have chosen to extend OMF. Here, we give a detailed description of the additions and the extensions we had to make inside this framework to integrate spectrum slicing support.

Firstly, we need a way for OMF and the scheduler's database to communicate. For this purpose, we have added one more service group to the Aggregate Manager named scheduler and one more service to the inventory service group. Next, we show what these services are responsible for. First of all, the inventory service group is developed inside OMF and provides a set of webservices that provide general information about the testbed (such as node names, IP addresses, etc). This information is stored in a database residing on the testbed server and the inventory service group reads this database to return the proper response. Our addition here is a service which gets a node location (i.e., its coordinates) based on its IP address. Note here that the information on the node location is the same on both the scheduler's and the testbed's database and, thus, we can use it to do the matching (coordinates do not refer to real data, but on an internal mapping that helps partitioning the testbed into groups while also allowing the identification of each node by OMF). We have added this service because, when an experiment is executed, OMF does not know a node's location, but only its IP address.

Now that scheduler knows the exact location of the node, it can use the scheduler service group to get any information needed from the scheduler's database. Namely, the services provided by this group provide functionality to get a node reservations based on its coordinates, the spectrum that this reservation contains and the user that owns it. Furthermore, it provides services that can do the matching between a channel or a frequency number and the respective spectrum identification number as it is stored in the database. All this information will be used by the Resource Controller, which decides whether to allow the user to use the channel or not.

Thus, RC is responsible for deciding whether the resources declared in the experiment should be allocated to the experimenter. In order to decide, the RC has to ask the scheduler's database if the specified resources have been

reserved by the experimenter. So, when the experiment sets the wireless card channel, this information is passed to the RC, which now knows the channel along with its own IP address. All he needs is the user identification to check with the scheduler's database if this channel (and, of course, node) should be allocated to that user.

However, this is not straightforward, since the user usually logs into the node as root (keep in mind that the experiment loads his own image to the nodes, so he has full privileges on them). So, we need to track where did he use the username that he also used for registering. The scheduler is designed in such a manner that, when a user registers to the system, then an account with the same username and password is automatically created to the testbed's server. The user uses this account to both access the user interface and the testbed server (using secure shell connection). This can solve our problem, since we can say for sure that the user that is running the experiment is logged into the console with the same username that he has made his reservation.

This information, though, relies on the testbed server, while the RC runs on the client side, i.e., on the nodes. We need to pass that information from the server to the clients. This is done by the Experiment Controller, the OMF service that is running on the server side and is responsible for controlling the experiment execution. Using its built-in message passing mechanism, EC tells the RC the username of the experimenter and now the last one has almost everything he needs to do the matching, except the date. The system should not rely on the experimenter to keep the clock of his clients synchronized with the testbed. This is why, EC sends, along with the username, the current date and the RC adjusts its clock to match the server's clock.

At this point, RC has all the information needed to check with the scheduler if the requested resources should be allocated to the experimenter. Using the web services we described above, the RC checks if there is a reservation at that time for that user and if the spectrum reserved at this reservation



matches the channel that the experimenter has requested to assign to the network card through his experiment.

If all data match, then the RC lets the experiment execution move on. Otherwise, it notifies the EC that a resource violation has taken place and stops its execution (without assigning the channel to the node network card). When the EC receives that message, the execution is terminated immediately and an ERROR message is thrown back to the experimenter describing the resource violation. Then the user is prompted to reconfigure its experiment with the permitted frequencies that he is allowed to use and he has already reserved during the scheduling process (see 4.2.3).

#### **NITOS scheduler advantages**

NITOS scheduler provides all the appropriate tools to allow slicing to its resources. Because of the external deployment of NITOS testbed, interference from external WMN links cannot be avoided. For that reason, NITOS Connectivity tool aids in identifying resources that best fit to the users experiment requirements. Moreover, NITOS Scheduler and its tools can be modified with minor changes and adapted to any wireless testbed that needs usage efficiency no matter if it is located in an isolated environment or it is located among external WMNs. In this way, NITOS scheduler aims to achieve better utilization of testbed resources, while also enables users to deploy their experiments in a more efficient way.

#### **4.2.4 PlanetLab and OMF integration**

Our main goal is to integrate a global scale PlanetLab infrastructure with a local OMF-based wireless testbed. In particular, we aim at using the OMF-based testbed as an access wireless mesh network for a set of PlanetLab nodes co-located (i.e. in range of wireless transmission) with it.

As described in the introduction, we recognize a value in this integration, as a first necessary step for the federation of these two kinds of infrastructures, and because it adds new capabilities to the PlanetLab environment. Our

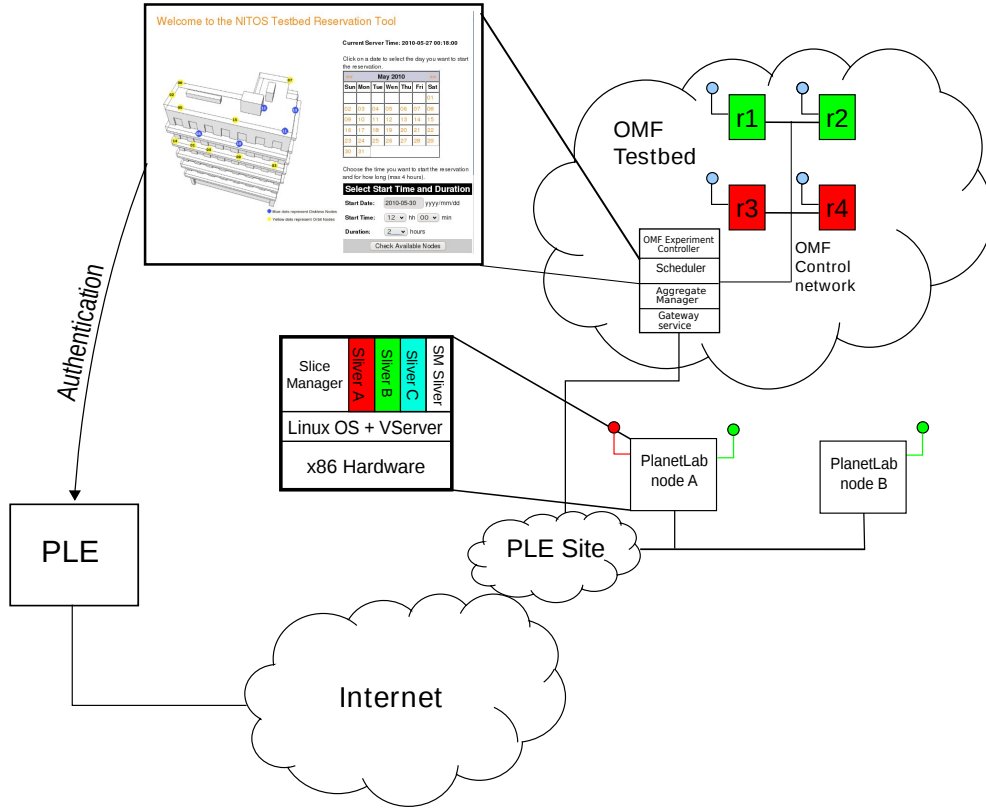


Figure 4.13: OMF-PlanetLab integrated architecture.

system allows the seamless integration of the OMF-resources into the global scale PlanetLab infrastructure, creating a synergic interaction between the two environments.

### Integrated architecture

The architecture we propose is depicted in Fig. 4.13. It consists of the following elements:

- A PlanetLab site  $S$  whose nodes are equipped with one or more WiFi interfaces that allow them to be connected to a local wireless OMF testbed. In the following these nodes are called *PlanetLab Edge Nodes* (PL-Edge Nodes).
- The PlanetLab Europe Central server (PLE), which hosts the informa-

tion on the PlanetLab Europe testbed, e.g. user accounts, slices.

- The OMF testbed and its components: the Aggregate Manager, the Experiment Controller and the Gateway Service.
- The extended NITOS Scheduler, used to manage the reservation of resources shared through booking.

The Gateway Service is implemented in a Linux box and acts as a *Network Address Translator* (NAT). It is needed for enabling Internet access to the OMF testbed's nodes, whose NICs are assigned private IP addresses.

The PL-Edge nodes are multi-homed PlanetLab nodes which can act as clients for the OMF wireless testbed. The lack of proper support for multihoming in PlanetLab led us to the developement of *sliceip*, a tool for allowing the definition of slice-specific routing tables that will be presented later.

The purpose of the extended NITOS scheduler is to allow the reservation of bookable resources in the integrated scenario. These resources comprises both OMF wireless nodes and channels, and PlanetLab non-virtualized resources, i.e. the WiFi interfaces. To do that, the extended NITOS scheduler interacts with the *OMF Console*, in order to enable or disable access to slices to the Experiment Controller, and with the PlanetLab nodes, in order to enable or disable the access to specific slices to the wireless interfaces. The communication with the PlanetLab nodes is performed by means of a management sliver, called *SM Sliver* (Scheduler Management Sliver), which accepts requests by the Scheduler through a secure ssh connection and performs the association between the slices and the wireless interfaces. We remember that we allow only one slice at a time to have access to a wireless interface, in order to limit interferences among experiments.

The Scheduler performs authentication of the user on the PLE, thus allowing access to the Tier-2 OMF wireless testbed to PlanetLab Europe users. Local users, i.e. users of the wireless testbed, are supported and their credential are stored on the Scheduler. These class of users however, i.e. users

of the Tier-2 testbed, have not access to the global infrastructure, i.e. the Tier 1 testbed.

In the OMF wireless testbed private IP addressing is used. Therefore, in order to allow experiments involving nodes located elsewhere on the public Internet, a node acting as a NAT router is needed. This function is performed by the Gateway Service. In the case of experiments involving OMF nodes located at different PL-OMF sites, site-to-site IP tunnels might be established between PL-OMF Edge Nodes. This process would be easy to be managed if these nodes were VINI nodes.

After user authentication the OMF Scheduler, by means of cron scripts, enables/disables access to OMF testbed nodes from the user's slice.

### Usage model

In the following we list the sequence of steps needed to execute an experiment using an OMF testbed at site S as access network for PlanetLab. The experiment is going to be executed over a specific time interval  $T = [T\_START, T\_END]$ .

1. PlanetLab user U adds one or more PL-OMF Edge Nodes (OP) to his/her slice;
2. U logs into the Scheduler at site S and books the resources (nodes, channels, WiFi interfaces of OP nodes) he needs for his/her experiment over time interval T, providing the slice identifier. According to PlanetLab's resource management scheme, booked resources are actually associated with such slice rather than with the user that performed the reservation;
3. While time is in T, each slice's user is allowed to access the OMF EC (Experiment Controller) to perform his/her experiment involving the OMF resources assigned to him/her.

## Multihoming support in PlanetLab

While trying to support the proposed usage model, we run across a serious limitation of the PlanetLab management software. Such a limitation is about the correct managing of multi-homed nodes, i.e. nodes connected to more than one access network. This has not been a problem for a long time, as PlanetLab mainly consisted of just a set of hosts connected to Internet through a single, high speed corporate connection. In such a scenario, there is no need for users to be able to modify the routing table, as the route for the Internet is only one. In recent times, though, some attempts to enhance the heterogeneity of PlanetLab have been made. In the context of the OneLab European research project, different kinds of wireless access technologies (such as UMTS, WiMAX and WiFi) have been made available to a subset of nodes connected to PlanetLab Europe, in addition to the default wired connection to the Internet. In [24], the software tools that have been developed to manage a UMTS connection in that context are described.

In this paper we describe a generalization of that software, allowing it to work with any kind of network interface.

## The *sliceip* tool

In order to fully exploit the possibility of multi-homed PlanetLab nodes we developed a tool called *sliceip*. The purpose of this tool is to enable slice-specific routing tables in PlanetLab. Using this tool, the user is able to define routing rules which apply only to traffic belonging to his/her slice. This is required for users to be able to choose which interface to use for their experiments. For instance, a user can specify that he or she wants to reach a certain destination on the Internet, e.g. another PlanetLab node, through the WiFi interface. For achieving this result, he or she would add a routing rule in his/her own routing table by means of our tool, in the same way he or she would do with conventional tools like *ip* or *route*. This is not possible in PlanetLab, because PlanetLab users do not have the superuser privileges required to modify the routing table of the node. Even if they had

such privileges, any modification they performed on the routing table would interfere with all the experiments running on that node, thus breaking the isolation among experiments. With *sliceip*, instead, we give to the user the ability to define his/her own routing table, with no effects on experiments performed by other users.

*sliceip* enables slice-specific routing tables by leveraging a feature of the Linux kernel and a feature of the VNET+ subsystem of PlanetLab [121]. The Linux kernel has the ability to define up to 255 routing tables. To have some traffic routed with a particular routing table, it is necessary to associate that traffic to it by means of rules applied with *iproute2*. The rules can specify packets in terms of the destination address, the netfilter mark, etc. In our case, we set the netfilter mark of packets belonging to the user's slice (i.e. the packets that are generated or are going to be received by an application running on that slice) by exploiting a feature of the VNET+ subsystem of PlanetLab. By means of an *iptables* rule, we instruct VNET+ to set the netfilter mark equal to the slice id to which they belong. We then add an *iproute2* rule to associate packets belonging to the slice to the slice-specific routing table. We also set an *iptables* SNAT rule (*Source Network Address Translation*) in order to set the source IP addresses of packets that are going out through a non-primary interface (the primary interface is the one the default routing rule points to). This rule is required because the source ip addresses of packets are set after the *first routing process* happens. In fact, in case more than a routing table is used, the routing process follows these steps: 1) the interface for sending the packets is decided following the rules of the main routing table and the source ip addresses are set accordingly (this is the first routing process); 2) if the user changes the mark of the packets in the *mangle chain* of *iptables* and a rule is defined for routing those packets with a different routing table, a *rerouting process* is triggered. This rerouting process follows the rules of the selected (i.e. the slice-specific) routing table and the interface to be used is set accordingly; 3) the packet is sent out using the selected interface. During the step 2, the source ip addresses of packets

are left unchanged, so we need to change them explicitly before the packets are sent during the step 3.

The user interacts with *sliceip* by means of a frontend that resides in the slice. This frontend extends the syntax of the *ip* command of the *iproute2* suite with the following two commands:

- *enable <interface>*: initialise the routing table for the user's slice, set the rule to mark packets belonging to the user's slice, add a rule to associate those packets with the routing table of the slice and add the SNAT rule for <interface>;
- *disable <interface>*: remove the SNAT rule for <interface>, remove the rule to associate the packets to the routing table of the slice and remove the rule that marks the packets of the user's slice.

### Extension of the NITOS scheduler to manage PlanetLab resources

In order to support the reservation of bookable Planetlab resources, i.e. the WiFi interfaces of the PL-edge nodes, we had to extend the NITOS Scheduler and make some additions to the management software of the PL-edge nodes.

The Scheduler has been extended to show among the available resources also the WiFi interfaces of the PL-Edge Nodes and to allow the user to reserve them. Reservation records are kept in the Scheduler database and it is Scheduler responsibility to make sure that reservations made by two users do not overlap.

In order to enforce the assignment of the interface to the slice, when the reservation time starts, the Scheduler interacts with the *Scheduler Management Sliver* allocated on the PL-edge node. Such interaction is performed through a secure ssh connection. By means of *vsys* [18], the *Scheduler Management Sliver* is able to execute a script in the root context. This script makes the actual assignment of the WiFi interface to the slice by setting some *iptables* rules which block all packets that are about to go out through the WiFi interface and do not belong to the slice for which the WiFi interface has been reserved.

The Scheduler checks the user's credentials by means of the PLC API and enables/disables access to OMF testbed nodes from the user's slice for the specific time and duration. In particular, the Scheduler interface is extended to support authentication of users by means of PLC managed usernames and passwords, while access to the OMF EC is performed by means of users' public keys linked to the slice, retrieved using the PLC API.

## **4.2.5 Experimental setup**

### **The NITOS testbed**

It is important to give an overview of the hardware facilities that comprise the heterogeneous profile of NITOS testbed. NITOS is a wireless testbed located in the University of Thessaly campus. NITOS as the main wireless testbed in the Onelab2 project, aims to provide all the software and hardware facilities that can gather multiple wireless communication technologies under a common structure. The main technology that is available in NITOS for implementation and testing is WiFi. Large scale testbeds are likely to feature hardware of different architecture and performance. NITOS testbed features 3 different types of computer main boards, 2 types of wireless media as well as 2 other types of peripherals. More specifically the NITOS testbed features 10 Alix embedded PoE nodes with 500Mhz i386 CPUs, which are primarily used for development of networking systems, 10 Orbit AC powered nodes (1 Ghz i386 CPUs and 1 Gb ram) and 20 Commel AC powered nodes that feature 2.4 GHz core duo CPUs (x86\_64). Wireless media includes 50 Atheros 5212 interfaces and 10 Atheros 5001 interfaces. Orbit nodes are equipped with high quality USB cameras that can be used for video enabled experiments and 6 commel nodes are attached with GNU Radio peripherals that support PHY layer experimentation.

### **The WILEE testbed**

The WILEE (WIrELeSS Experimental) WiFi Mesh Testbed is located in the Computing Department of University of Napoli Federico II. It consists of



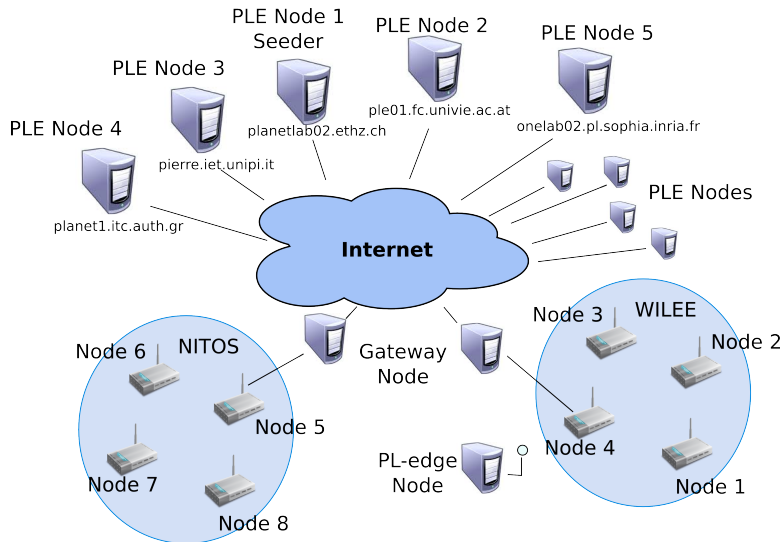


Figure 4.14: Experiments setup.

three Soekris net4826-48 Single Board Computers and eight Netgear WG302Uv1 access points. It also features a node belonging to a private PlanetLab deployment which acts as the PlanetLab Edge node and a Linux machine acting as gateway towards the Internet.

The Soekris net4826-50 SBC is based on the AMD Geode SC1100 CPU (at 266Mhz), has 128 Mbyte DRAM memory, a 128 Mbyte Flash disk, a FastEthernet interface and two 802.11a/g Atheros wireless cards. The Netgear WG302Uv1 access point features on an Intel XScale IXP422B network processor (at 266Mhz), has 32 Mbyte DRAM memory, a 16 Mbyte flash disk, a FastEthernet interface and two 802.11a/g Atheros wireless cards.

#### 4.2.6 Experiments

In this paragraph we describe an experiment aimed at investigating a problem that is frequently studied on top of PlanetLab, i.e. peer-to-peer traffic optimization. The peculiarity, in our case, is that we create a distributed setup for our experiment involving the use of our wireless mesh testbeds as access networks to the Internet. In fact, we intend to investigate this problem, and compare its solutions, in the specific context of WMNs, where

specific cross-layer approaches can be part of the solution.

### Testing overlay routing strategies in WMN-based access networks

An increasing number of popular Internet applications, such as Bittorrent, Skype, GoogleTalk, and P2P-TV relies on the peer-to-peer paradigm. These applications produce more than 50 percent of the overall Internet traffic. One of the inherent characteristics of peer-to-peer systems is that they build *network overlays* among their peers, and route traffic among them along the virtual links of such an overlay. Peer-to-peer routing decisions are made at the application layer, independently of Internet routing and ISP topologies. Hence, overlay routing decisions collide with those made by underlay routing, i.e. ISP routing decisions [79]. As a consequence of such a dichotomy, several inefficiencies may result. For instance, it is not uncommon that adjacent nodes of an overlay network are in different ASes. Such a topology arrangement leads to traffic crossing network boundaries multiple times, thus overloading links which are frequently subject to congestion, while an equivalent overlay topology with nodes located inside the same AS could have had same performance. Such a behavior is undesirable for ISPs, also because their mutual economic agreements take into account the volume of traffic crossing the ISP boundaries.

From what we described above, it emerges that overlay routing, and peer-to-peer applications, may benefit from some form of underlay information recovery, or in general from cross-layer information exchange. Aggarwal et al. in [1] suggest that such a cooperation would be beneficial for both ISPs and users. When creating an overlay network, the choice of the nodes to be connected, i.e. the network topology, can be done by taking advantage of information from the underlay network. Different strategies have been proposed recently in the literature that attempt to introduce some cooperation between the two routing layers [1][152]. Given the role of access networks played by wireless mesh networks, it is interesting to experiment with such techniques when peers are attached to different WMNs connected to the In-

ternet. Our contribution in this section makes such experiments possible. In the next subsection, we report the results of experiments carried out to show that our approach makes it very simple to perform realistic experiments to test overlay so we describe an experiment aimed at evaluating a traffic optimization solution for a BitTorrent file-sharing peer-to-peer system. BitTorrent is used to efficiently distribute files of large size from one or more initial *seeds* to a population of large numbers of downloaders, forming what is referred to as a *swarm*. Files are exchanged in smaller *chunks* that can be individually retrieved. One of the peculiarities of BitTorrent is that downloaders, a.k.a. *leechers* in BitTorrent terminology, also contribute to spread the content to other peers. As soon as a peer obtains all the chunks of the desired file, it becomes a seed on its own. We have designed and implemented a solution that aims at incentivizing traffic exchange in a BitTorrent system between peers that are located within the same Autonomous System. Our solution does not require any modification to the BitTorrent protocols, nor to the application used by end users. The only modified component of a typical BitTorrent system is the *Tracker*, i.e. the system that is contacted by peers to obtain a list of other peers to contact, in order to retrieve chunks of the file to download. In our system, the tracker returns to peers a sorted list of peers to be contacted, where the sorting criterion is by-increasing-AS-distance. In other terms, as soon as a peer contacts the tracker, the tracker determines the AS-number associated with the IP address of that peer, and returns a list of peers whose first items are the closest peers in the swarm (in terms of AS distance), while the last items are the furthest peers. Our experiment is aimed at evaluating our tracker-based solution when a significant fraction of peers are connected to the Internet through the same wireless mesh network. Our objective is to show that in this case, by adopting our strategy, a substantial amount of traffic is reduced through the wireless mesh gateway, i.e. the node connecting the wireless mesh to the wired Internet. To this purpose we created a slice involving ten PlanetLab Europe nodes and the PlanetLab edge node situated at the edge of the WILEE testbed. To

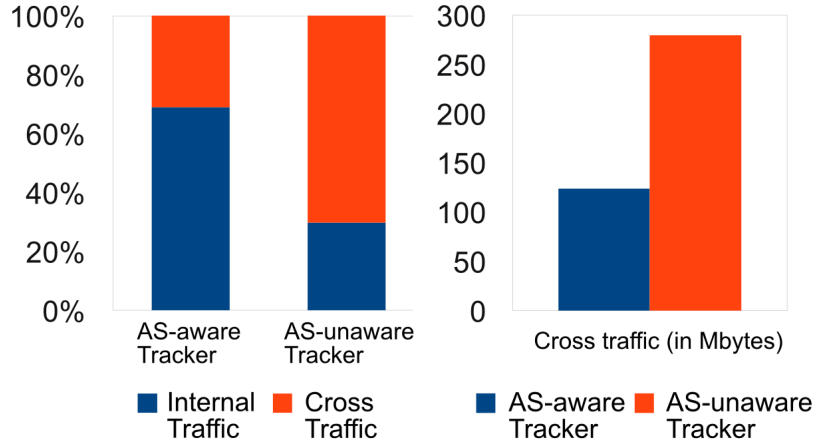


Figure 4.15: Experiments: internal vs. cross traffic (percentage of total traffic) on the left; cross traffic volume on the right.

this slice, some bookable resources, i.e. four wireless nodes from the WILEE testbed and the WiFi interface of the PL-edge node, were added to the slice by using the extended NITOS Scheduler at the WILEE site. In the same way, other four nodes belonging to the NITOS testbed were added by using the extended NITOS Scheduler at the NITOS site.

The wireless nodes were configured by using the facility offered by OMF to form two single-channel WMNs and, in case of WILEE nodes, also to provide Internet access to the PL-edge node. A Bittorrent client (*TransmissionBT*) was installed on the PlanetLab Europe nodes, on the PL-edge node and on the wireless nodes. One of the PlanetLab Europe nodes was chosen as the seeder of the Bittorrent swarm, which consisted of a file of approximately 50 megabytes. The scenario of the experiments is illustrated in Fig. 4.14.

We performed a set of experiments by employing alternatively a standard Bittorrent tracker (*Quash*) and the same tracker modified by us in order to take into account the distance between peers in terms of ASes.

At the end of each experiment we measured the traffic belonging to connections which were either originated or destined to nodes located behind the OMF gateways, i.e. the NITOS and WILEE wireless nodes and the PL-Edge

Table 4.1: Traffic matrix for an experiment with the modified Tracker.

	N1	N2	N3	N4	PL-Edge	N5	N6	N7	N8	PlanetLab
N1	0	2.34	1	0	0.44	0	0	0.81	0	41.03
N2	0	39.77	0.06	0.06	1.39	0	0	0	0	5.69
N3	13.99	3.9	0	1.89	27.19	0	0	0	0	0
N4	13.36	3.61	5.27	0	26.45	0	0	0	0	0
PL-Edge	40.7	4.23	0.64	0.09	0	0	0	0	0	0
N5	0	0	0	0	0	0	0.13	0.09	0	45.03
N6	0	0	0	13.2	0	29.79	0	0	3.55	0
N7	0	0	0	0	0	20.12	23.91	0	2.29	0
N8	0	0	0	0	0	8.17	1.95	0.5	0	37.05

Table 4.2: Traffic matrix for an experiment with the standard Quash Tracker.

	N1	N2	N3	N4	PL-Edge	N5	N6	N7	N8	PlanetLab
N1	0	0	0	2.88	0	0	0	0	0	43.37
N2	0	0	0	5.5	62.3	0	0	0	0	4.38
N3	0	0	0	0	4.73	0	0	0	0	48.84
N4	44.43	7.52	0	0	0	0	0	0	0	0
PL-Edge	0	0	7.88	0	0	0	0	0	0	46.88
N5	0	0	0	0	0	0	0	0	22.97	24.29
N6	0	0	0	0	0	0	5.31	0	0	40.82
N7	0	0	0	0	13.65	0	0	0	0	37.53
N8	0	0	0	0	10.82	19.88	0	0	0	16.14

node. Our objective was to demonstrate that the traffic crossing the WMNs boundaries was minimized by using our modified tracker. In Fig. 4.15 we report the results averaged on 10 repetitions. The figure shows that the amount of traffic flowing through the OMF Gateways was significantly lower in case the modified tracker was used. If we compare the overall amount of bytes exchanged by peers, the results show that, in case the modified tracker was used, the file was downloaded in average from the outside slightly more than once for each WMN, and then disseminated in the WMNs among nearby nodes. In case the unmodified tracker was employed, instead, it is as though the file was retrieved almost three times by each WMN (about 280 Mbytes downloaded from the outside by the two WMNs), thus indicating a non-optimum peer selection strategy. Tables 4.1 and 4.2 report the traffic matrices for two experiments. On the rows are the receiving nodes, while on the columns are the sending nodes. N1, N2, etc. stand for Node1, Node2, etc., while Planet-

Lab is a meta node which comprises all the PlanetLab nodes. All the values are in Mbytes. It can be seen that, in case the modified tracker is used (Table 4.1), traffic is exchanged mainly between nodes located inside the same WMN, while in case the standard tracker is used (Table 4.2), wireless nodes often download from nodes which are outside their WMN.

While conducting the experiment, some real world issues arised and made evident the usefulness of having such an heterogeneous network scenario.

The first problem was about the private addressing of the WMN and the need to NAT the traffic generated from the wireless nodes and destined to the Internet. This was, however, not sufficient, as the Bittorrent protocol requires that the clients be reachable from the outside on public IP-port pairs. For this reason, we had to setup a NAT-PMP service on the gateway node [139]. Through this protocol, clients are able to request a port to be forwarded from the gateway node, so that they can accept incoming connections from other peers on the gateway IP and the assigned port.

Clients, therefore, announce themselves to the Tracker with their public IP-port pair. This requires, in turn, that the connections between two wireless nodes go through the gateway machine and be source NATted, at the gateway node, even if they do not involve a node on the Internet. Solutions to this problem require modification to the Bittorrent client, e.g. in order to implement a local peer discovery process.

#### 4.2.7 Related work

In this section we have presented an architectural solution to integrate a number of local OMF-based wireless testbeds with the global-scale PlanetLab environment. Our solution is a first technical solution towards the federation of these two kinds of testbeds. The problem of heterogeneous testbeds federation is under investigations of both the GENI initiative in the US and the FIRE initiative in Europe. For instance, federation between PlanetLab and EMULAB is currently being investigated in the context of the GENI initiative, as reported in [100]. An attempt to add heterogeinity in PlanetLab

by integration of ORBIT testbeds is in [83]. In this paper, the authors propose two models of integration. The first model (PDIE, *PlanetLab Driven Integrated Experimentation*) is intended to serve PlanetLab users who want to extend their experiments to include wireless networks at the edge without changing the PlanetLab interface, while the second model (ODIE, *ORBIT Driven Integrated Experimentation*) is intended to serve ORBIT wireless network experimenters who want to augment their experiments by adding wired network features without major changes to their code.

Our proposed model of integration is more similar to the PDIE model, with a difference with regard to the connectivity model between the two environments. In order to integrate an OMF-testbed in PlanetLab, the authors propose the use of a gateway PlanetLab node, whose function is to open tunnels between itself and the selected nodes in the OMF testbed. Differently from our approach, the gateway node is not a client of the OMF testbed, but merely creates the tunnels. Our approach does not employ tunnels. A similar approach was taken in [58]. The authors aimed at integrating the VINI virtual network infrastructure [17] with OMF-based testbeds. The intention was to enable Layer 3 experimentations by allowing users create virtual topologies spanning both wired and wireless links. Also this approach relies on the use of tunnels.

Our approach intends to recreate in the testbed the same operational situation that exists in real networks, in which a private addresses mesh is connected to the Internet through NATing gateways. Our integrated experimental facility allows experimentation of low level mechanisms within the wireless mesh environment provided by the OMF testbed, and end-to-end mechanisms and applications in the global hybrid integrated environment. These features create a synergy between the two kinds of facilities. As we mentioned in the introduction of the paper, for achieving a full-fledged federation of the two environments, other issues need to be fully solved, such as the creation of a single sign-on mechanisms for the two environments.

#### **4.2.8 Conclusions**

The availability of large scale testbeds integrating several local wireless mesh testbed in a realistic global-scale environment is necessary to test WMNs in the wild. In this section we present an integration architecture for experimenting with local OMF-based wireless testbeds in the context of PlanetLab. We also present some test case experiments we run on our initial implementation of the integrated architecture. In particular, we describe an experiment aimed at evaluating a BitTorrent traffic optimization system. Our experiment combines both wireless nodes of two OMF based testbeds and PlanetLab nodes located across Europe. The possibility of running this kind of experiments in such a hybrid experimental scenario highlighted several real-world issues, such as the impact on performance of NAT traversal systems, that are worth to be further investigated and that could only be reproduced thanks to our integrated environment.



## Chapter 5

# Programmable networks

The need for the flexible handling of networking infrastructures is recognized by both academy and industry: on one hand researchers need flexibility in order to prove new ideas through real deployments, on the other hand companies need flexibility to better manage the infrastructure to reduce the costs and to enable new business opportunities. This shared need made the development of Software Defined Networking (SDN) surprisingly fast in first instance, starting a technology hype that is still growing at the time of reading. The mutual effort in developing this approach materializes into the OpenFlow technology[101], that was born in an academic context to be then embraced by industry. SDN suggest an approach in managing the network that separates the network control plane from the data plane, enabling different path of evolution for the two planes. Differently from what happen with legacy network devices, whose behavior is defined by the control plane implemented in the device itself, in SDN the device behavior is programmed through a well defined interface, used by the control plane that can be implemented in a separated entity. The main outcome of this architecture is that the network behavior can be defined by software programs, providing a shift in the way network are designed and managed similar in the way it happened with electronics when general purpose computers were introduced. In this chapter we further extend this concept analysing the scalability issues in SDN control plane, introducing an approach similar to the one used

to characterize general purpose computer programs. Then, we introduce an algorithm and a tool to help the task of network programming, providing at the same time an example of the new fields opened by the SDN architectures.

## 5.1 SDN's control plane scalability

### 5.1.1 Introduction

The emerging Software Defined Networking paradigm promises to unleash new opportunities for building and managing future networks. The basic SDN concept is as simple as powerful and relies on separating the network control plane from the data plane with the purpose of gaining in flexibility in the infrastructure management. In a computer network, the control plane is responsible of defining how data traffic is handled and resources are used at a global level, while the data plane is in charge of actually forwarding network packets within each device. In traditional network architectures, devices have to play a role in both planes. For instance, an Ethernet switch forwards packets to a given port based on a forwarding table that is created by learning source MAC addresses and VLAN settings; an IP router participates into routing decisions and, at the same time, forwards packets to adjacent routers or end-systems. The impressive growth of the Internet in the last decades proves that the traditional approach could scale to a very large extent. Nonetheless, many researchers have pointed out that the traditional networking model is exhibiting symptoms of a progressive “ossification”, that is a strong difficulty in accommodating architectural changes and in supporting new applications. Since applications are experiencing a much faster evolution, Internet capabilities have been extended, from time to time, by means of patches and work-arounds, that usually aim at solving a single application-specific issue. This way of proceeding inevitably leads to great complexity of network management operations, with the consequently growth of operational costs.

In an SDN architecture, control and data plane are decoupled. Network

switches are not involved in control plane operations anymore. Control plane functions are assigned to “Controller” entities, communicating with network switches through well-established interfaces and protocols. By doing so, the operation of the whole network may be completely re-programmed by simply modifying the software control logic implemented in Controllers. Hence, *“network intelligence and state are logically centralized, and the underlying network infrastructure is abstracted from the applications. As a result, enterprises and carriers gain unprecedented programmability, automation, and network control, enabling them to build highly scalable, flexible networks that readily adapt to changing business needs and open the field to new applications and communication paradigms”* [106]. Programmability is actually one of the most important properties of SDN: the control plane behavior can be defined by writing “network programs” that manage a set of switches, providing rich network applications and features. In some way, SDN is bringing into computer networks the same shift that in past decades has transformed many consumer electronic devices from special purpose appliances into general purpose machines. With the advent of cloud computing, the need for affordable, flexible and scalable management procedures is becoming an urgent need. The great potential of SDN appears appealing to most device manufacturers. For the final success of this approach, however, it still has to prove its robustness and scalability in real world scenarios.

OpenFlow is one of the most popular SDN-enabling technologies. OpenFlow was born as a mean to enable network experiments on campus networks[89], and its first deployments were actually universities’ networks. Over time, the advantages of an SDN approach to networks have been explored, leading to applications of OpenFlow to other environments, such as enterprise networks, as in the OpenFlow implementation of the Ethane architecture for network security[32]. More recently, OpenFlow has been also applied to challenging scenarios like datacenter networks[141] and wide-area networks[69], as well as for providing new features, such as mobility support, in traditional TCP/IP networks[22]. The Open Networking Foundation [104] (ONF), that is respon-

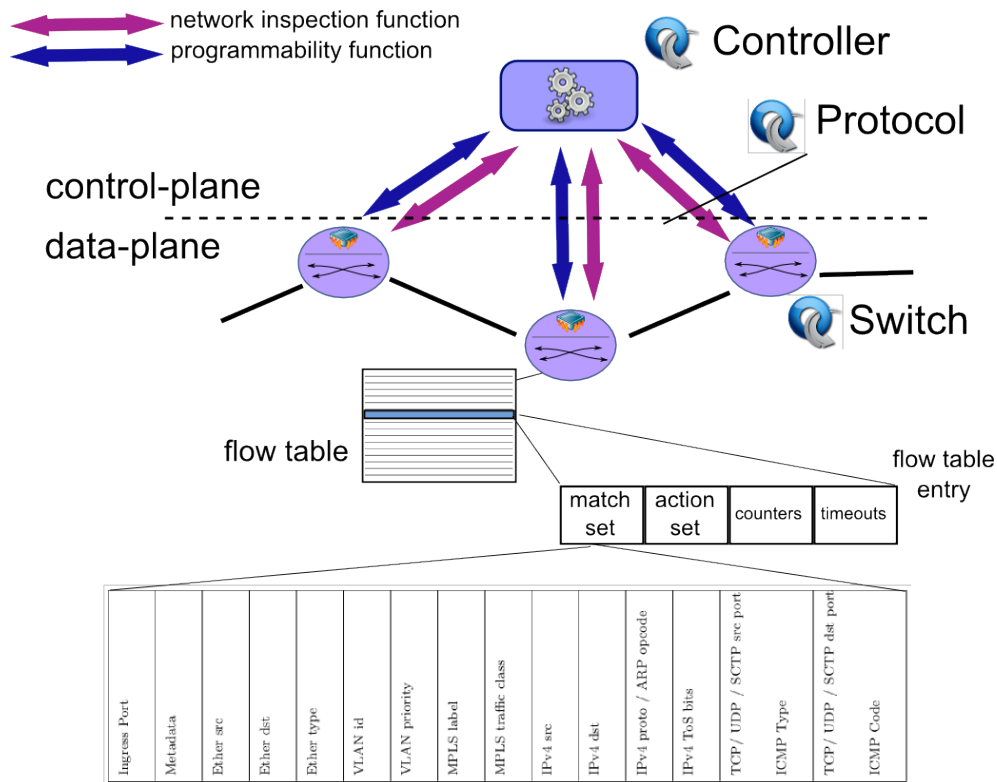


Figure 5.1: OpenFlow architecture

sible for the OpenFlow specification, currently involves a number of academic and industrial partners. An increasing number of device manufacturers have implemented OpenFlow in their products and Google recently declared the adoption of OpenFlow in its networks[150].

In this section we discuss in depth the scalability issues of OpenFlow and present possible solutions.

### 5.1.2 OpenFlow

The OpenFlow specification defines the behavior of an “OpenFlow switch”, as well as the programming interface and the protocol used to interact with such switches [104]. Switches are connected to an external Controller through a communication channel. How this channel is implemented is out of the scope of the OpenFlow specification. Usually, it is implemented as a TLS

secure channel established on top of a dedicated “control network” relying on TCP/IP. OpenFlow (OF) provides Controllers with two features, i.e. the ability of programming and inspecting switches. An OF switch programming is performed using a rule-based approach. The role of a Controller is to provide a set of rules to be installed at different switches by means of the OpenFlow protocol. In the OpenFlow terminology, the rules used to program a switch are called *flow table entries* (FTE). An FTE is defined by the *match set*, that defines to which network flows the entry is applied, the *action set*, that defines the elaborations and the forwarding decision that must be applied to the matched flows, a *priority*, to relatively order the entries installed in a switch, and an *expiration time*, specified as a timeout. A typical FTE provides a semantic like “forward network flows with network destination address 1.1.1.1 to switch port 3”. By means of rules, an OF switch may be instructed to forward packets to a special “controller port” for allowing the Controller to inspect packet headers. For instance, an OF switch may be configured to forward to the Controller any packet that is not handled by an installed FTE, so that appropriate entries can be devised and installed on the switch to handle subsequent packets of the same flow. When an FTE is installed in a switch, it is also associated with a set of counters that measure some statistics, e.g. the number of packets or bytes processed by that entry. By gathering switch statistics and FTEs counters, the Controller is able to collect data on network status.

Hence, the typical operations of an OpenFlow network may be simply described as follows. When a new flow is initiated by a network end-point, the first packet of the flow is intercepted by the OpenFlow switch to which the end-point is connected. If no flow table entries are found in the switch to handle the packet, the switch forwards the packet to the Controller. The Controller processes the packet and decides what table entries need to be installed in the controlled switches to correctly handle the flow. Of course, a proactive approach may be pursued by the Controller to install flow table entries.

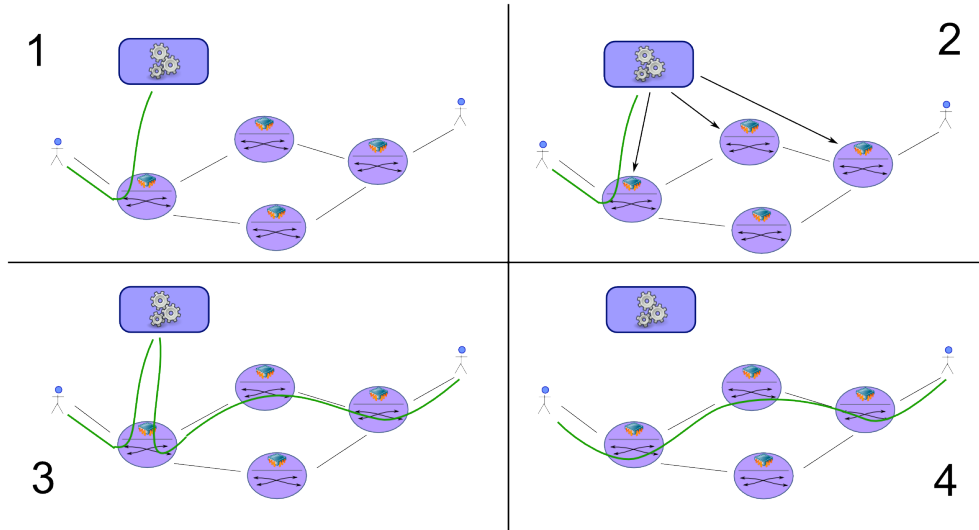


Figure 5.2: OpenFlow network flow handling example: a just started flow is handled through the reactive programming of switches

### 5.1.3 OpenFlow Scalability

Scalability is the ability of a system to handle an increasing workload. Since the system we are analyzing is a network, the workload is mainly represented by network traffic, even if some operations may provide additional workload not directly related to data traffic, e.g., in a network that supports mobility, the number of end-point movements is part of the overall workload. The aim of this section is to understand what are the factors that have an influence on the scalability of the OpenFlow architecture, and what are the scalability issues that arise because of such factors. In the OpenFlow architecture scalability is influenced by three factors:

- **Network topology:** like in traditional networks, the topology dictates physical limits to the ability of the network to sustain network traffic. Since the network topology design for scalability is a well-explored topic, from the OpenFlow perspective we are interested just in the total number of OF switches in the network;
- **Control Network:** it defines the way the Controller communicates with OF switches. The control network can be realized either in-band or

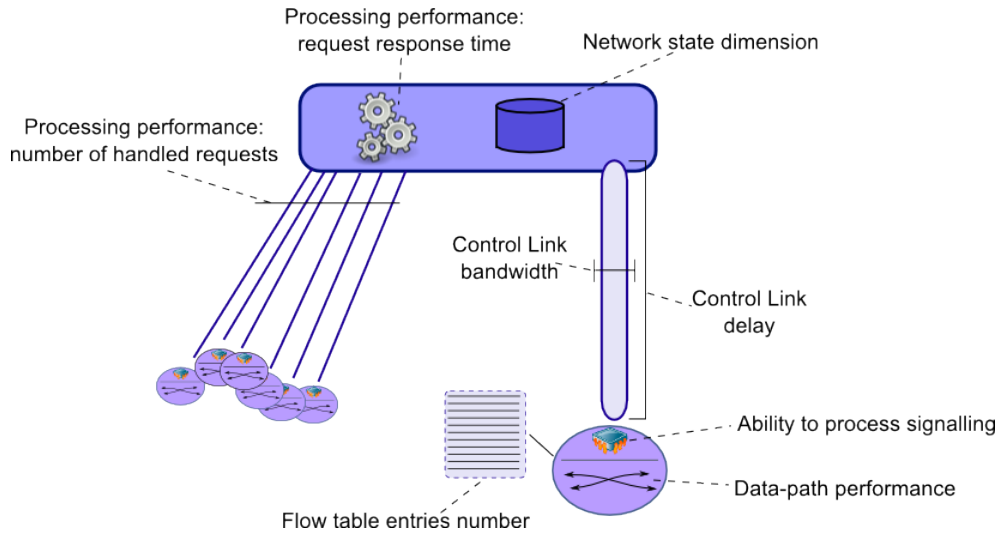


Figure 5.3: OpenFlow scalability issues

out-of-band. Usually it uses a traditional out-of-band TCP/IP network to connect switches to the Controller;

- **Control Logic:** that is the control program implemented by the Controller. OpenFlow scalability is greatly influenced by the way the control logic stresses the OF architecture. Some insights on control logic factors are presented in next sections.

Scalability issues are the architectural limits that make a system unable to support an increased workload. The above three factors may raise one or more scalability issues. A few specific scalability issues of the OpenFlow architecture have been already investigated by some recent papers with the aim of identifying potential bottlenecks. In the OpenFlow architecture the control plane, i.e. the Controller, and the data plane have to interact more frequently than in traditional networks, and their communications take place through a control link. Hence, scalability bottlenecks can be: i) the Controller ability to handle network events on time, ii) the switch ability to send and receive OpenFlow control messages and iii) the control link itself, whose bandwidth and delay must be taken into account[94]. Moreover, the peculiar role of the control plane in the OpenFlow operations is to be taken

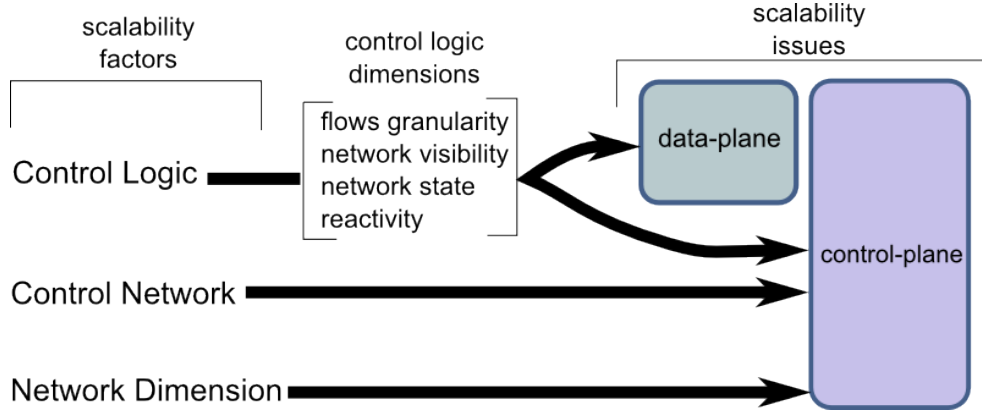


Figure 5.4: OpenFlow scalability factors, issues and their relationships

into account by network performance models, which need to consider that a variable number of network packets could be processed at the Controller[66]. Also the data-path itself, i.e., the part of the switches that handle traffic using flow table entries, requires a careful analysis, since flow table entries are much more flexible than traditional switches and routers data structures, at the cost of an increased complexity in their management[20].

To simplify the study of the issues presented so far, we can split the overall scalability problem into two sub-problems of *data plane scalability* and *control plane scalability*, following the architectural split between data plane and control plane. The former is related to the OF switches, the latter includes the Controller and the control network.

### Data plane scalability

Data plane scalability is related to OF switches and their interconnections. We already stated that we will not take into account the network topology in details here, since the scalability of a network from the topology perspective is a well-known and deeply explored field. We are actually reducing the scope to OF switches considered one by one, that makes the data plane scalability problem easier to understand and to analyze. An OpenFlow switch has limited resources that can limit its functions. Each switch could experience different issues depending on the way it is designed and implemented.



- **Data-path performance:** it is the ability of the switch to fast forward network packets. For simplicity, we can think of it as the time the switch takes to move a packet from one port to another, but, actually, since the switch could perform more complex actions than just forwarding (e.g., header rewriting), we can generalize it defining the data-path performance as the time the switch takes to process a network packet. OpenFlow increases the complexity of the packets handling, since the way a packet is handled is specified through flow table entries. When a packet is processed by a switch, a lookup action is performed to find the table entry that has to be applied. Entries could have wildcard in their match sets, making not always possible the use of hash algorithms or other search optimizations [20].
- **Signalling:** OpenFlow switches are programmed through the OpenFlow protocol. The number of signalling messages exchanged among the switch and the Controller is mainly dependent on the control logic. Anyway, the ability of the switch to handle such messages in an efficient way is one of the enabling factors for advanced network applications [94].
- **Network state:** the network state in an OpenFlow switch is actually represented by flow table entries. Depending on the implementation, the switch can host a fixed number of flow entries. In some hardware implementations, the flow entries are managed through several flow tables with different capabilities, both in terms of match set, e.g., support for wildcards, and actions set, e.g., support for rewriting actions.

### Control plane scalability

Control plane scalability is related to the Controller and to the control network. The Controller is in charge of: managing a set of switches through the installation of flow table entries; gathering traffic information; processing network packets coming from switches.

- **Processing Performance:** the Controller is usually implemented as a software application. Like any other software application, there are plenty different strategies to implement it and to provide peculiar performance characteristics. The most important properties for an OpenFlow Controller are the ability to respond to many requests at the same time and the time it takes to answer each request[66];
- **Signalling:** the control network has to transport all the signalling traffic of the network. Bandwidth and delay provided by such network assume a big importance in the overall architecture scalability as well as the relative location of the Controller in respect to the location of the switches it controls[94];
- **Network state:** the Controller has to manage the network state related to the implemented control logic. Since the Controller is logically centralized, and it could manage a big number of switches, the dimension of the network state could be a scalability limiting factor[22].

#### 5.1.4 Control logic dimensions

To understand the scalability problems of an OpenFlow network, we have to analyze the possible OpenFlow control plane behaviors. Since in OpenFlow the actual control plane behavior is defined by the Controller, OpenFlow networks scalability is greatly influenced by the way the control logic stresses the architecture. Using again the general purpose computer metaphor, in a general purpose computer scalability depends both on the hardware architecture and on the software running on top of such architecture, likewise in OpenFlow, it depends both on the network architecture and on the control logic implemented by the Controller. Given the flexibility of the OpenFlow approach, we can try to perform an high level classification of the characteristics of different control logics, identifying coarse-grained dimensions to classify such applications, the same way computers' programs are classified as CPU-bound, I/O bound and so on.

- **Flows Granularity:** defines the granularity of the network flows managed by the control logic. The granularity is defined after the header fields of current data packets. For example, a flow identified by the solely destination MAC address is coarse grained, while a flow identified by the combination of IP addresses and port numbers is much more fine grained;
- **Network Visibility:** an OF application may need detailed network traffic information or links statuses, for, e.g., load balancing or route reconfiguration. Depending on the control logic, the quantity and frequency of switches status update may vary;
- **Network state:** network state is related to the information the control logic has to manage, in order to provide its functions. Typical examples of network state are routing tables, end-points identity information, etc.;
- **Reactivity:** OpenFlow provides a mean to reactively program switches, through the forwarding of network packets to the controller. A control plane can range from being fully reactive, when each OF table entry is installed in response to a packet coming to the controller, to proactive, when all entries are installed before network traffic arrives to the switch.

### 5.1.5 Scalability solutions

To provide scalability to OpenFlow networks it is necessary to solve both data plane and control plane scalability issues. Data plane scalability solutions are related to the application of techniques aimed at increasing the capabilities of switches, such as the application of optimized algorithms for the flow tables memory management and lookup operations. Such solutions are not much different from the ones used in traditional network devices, even if OpenFlow data structures, i.e., flow table entries, are more flexible and complicated than traditional network devices ones. However, the main issue for OpenFlow

devices is about the need to move data between the data-path and the control plane, e.g., to send the new flow's first packet to the Controller or to answer with flows statistics information. How to handle this issue is still an open research field, in particular for the hardware implementations, and no specific solutions have been already presented.

Data plane scalability issues can be solved, or at least masqueraded, though a smart use of the network devices in combination with a control logic that takes into account the scalability issues of the data plane. Hence, scalability of the data plane could be traded with an increased complexity of the control logic. In the end, the control plane scalability is the core element in the overall scalability of the OpenFlow architecture.

OpenFlow control plane is logically centralized in the Controller entity whose implementation is usually a software based system. As explained in previous sections, the Controller implements a control logic that can vary significantly depending on the provided network functions and the way they are implemented, so, scalability solutions are closely related to the particular application. For instance, depending on the implementation, the Controller could be either a centralized or a distributed system. However, some operations are common to all Controllers, since they are all involved in the handling of the OpenFlow protocol, that includes: handling of network events like switches and links status updates, handling of network packets forwarded from the switches, delivery of OpenFlow messages to the switches, etc.. These operations have to be handled by any OpenFlow Controller implementation in a scalable way, and their scalability is closely related to the overall Controller scalability. Hence, to ease the development of the Controller, there are a lot of OpenFlow Controller Frameworks to handle such common operations in an efficient way. Control logic developers can access such operations through an easy-to-use application programming interfaces (API). Examples of such frameworks are NOX, Beacon, Floodlight, Trema, Maestro just to cite some of them. An OpenFlow Controller, hence, is usually implemented as a program running in the context of the chosen framework.

Taking into account this split between OpenFlow Controller Frameworks and actual control logic, we organize control plane scalability solutions in on two logical layers: Scalable Controllers and Scalable Architectures.

The Scalable Controller layer is about optimizing the handling of operations that are common to all the Controllers, independently from the actual control logic. In this layer scalability is mainly implemented at the OpenFlow Controller Framework (OCF) level. Scalable OCFs are either centralized or distributed systems, in both cases the control logic could be involved into the scalability issues handling, by providing some insight through configurations or through an appropriate use of the exposed OCF's API. In the centralized case, that is a common deployment scenario, the majority of OCFs try to optimize the handling of the OpenFlow protocol with advanced optimization techniques based on work batching, scheduling and multi-threading[29], [141]. To further scale the Controller to handle heavier workloads, it is possible to use several machines to run the Controller in a distributed way. Some OpenFlow Control Frameworks are providing such an approach using different strategies, with different involvement of the control logic into the distribution of the Controller operations. For example, Hyperflow[142] provides a Distributed OpenFlow Controller Framework that separates the network in partitions. Each partition is assigned to a controller instance and all the instances are synchronized by means of a publish/subscribe mechanism. In this approach each controller instance runs the same control logic, while the framework distributes the network events and OpenFlow messages to the appropriate controller instance. Hence, HyperFlow provides transparent scalability of the Controller. To provide its features, Hyperflow distributes consistently the network state updates and OpenFlow messages. Other approaches, such as Onix[72], require a direct involvement of the control logic in the scalability issues handling, in order to use the mechanisms the framework realizes to scale the system. Onix defines a Network Information Base (NIB) that is read and written by the control logic. The NIB is actually distributed using different strategies defined by the developers, that choose the storage

system type for the NIB data, according to their needs in terms of speed, consistency and reliability. All the state synchronizations and operations on the physical network are performed through the NIB, so, by partitioning the NIB among several controller instances, it is possible to share the overall load. A third approach is based on a service view of the Controller: the control logic is implemented as a collection of services that collaborate among them. Each service is able to run on a separate controller instance, hence, several instances can be used to share the workload. Clearly this approach requires a careful design of the control logic, whose implementation as a collection of services, and the way such services are distributed, dictates the actual scalability of the system, i.e., services should be able to be executed as much as possible in a parallel manner[135].

While the Scalable Controller layer is mainly related to the design of OCFs, the Scalable Architectures layer is about the design of the control logic. The two layers are complementary: scalable controller layer provides efficient building blocks for scalable architectures. There are no general solutions for scalable architectures, since they are related to the specific control logic, nonetheless, there are some high-level approaches that can be applied in different contexts. The aim of a scalable architecture is identifying the functions provided by the control logic and separate responsibilities for the provisioning of such functions among different controller instances. In any case, the final outcome is actually a partitioning of the controlled network, so that several controller instances can cooperate to provide the control logic. For example, in a geographical deployment, edge networks could be under the control of dedicated controllers, whose coordination is assigned to a “main” controller that has only a global view of the network. Such a hierarchical organization of controllers is based on a principle of aggregation: the network is partitioned, each partition is assigned to a controller instance, each controller instance presents an aggregated view of the partition to the higher level controller[72]. Another approach is the selective distribution of the network information through the use of federated controller instances. This architec-

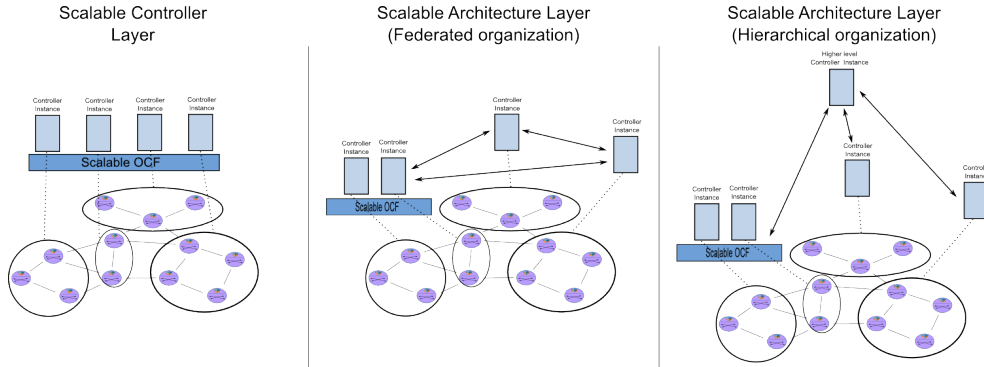


Figure 5.5: Scalability solutions

tural approach is based on the observation that, in order to provide some functions, not all the state information in a network partition is needed to orchestrate network-wide operations. The identification of the solely needed information in combination with the orchestration of collaborating controller instances, for the proper exchange of such information, can provide a better usage of the control plane resources and increase the scalability[22].

### 5.1.6 Conclusions

Software Defined Networking is a promising paradigm for future networks management, and OpenFlow is emerging as a successful industry-supported SDN implementation. Scalability is one of the enabling factor for new technologies successful application, and OpenFlow scalability is still an open research field. This section presented the OpenFlow architecture, its scalability issues and currently used approaches to overcome such issues. OpenFlow scalability problem can be divided into two separated problems of data plane scalability and control plane scalability. In both problems, scalability is highly influenced by the way the control logic is implemented, that can be classified according to some high-level dimensions, the same way it happens for computer programs, in order to understand the effects on the underlying infrastructure. Currently adopted solutions vary according to the application context. They can be categorized in two different layers: *Scalable Controller* layer provides solutions to optimize the OpenFlow protocol operations han-

dling, that is a common function realized by all Controllers, while *Scalable Architecture* layer provides solutions for designing the control logic. In both cases the applicability to real-world applications still needs to be evaluated in details. What is clear so far is that the potentialities of the OpenFlow architecture can be exploited in different contexts, but the way they are exploited, and the corresponding design challenges, may vary considerably, making the solutions highly related to the application context.

## 5.2 Network programming

### 5.2.1 Introduction

Software Defined Networking (SDN) introduces a new architecture that separates the network *control-plane* from the network *data-plane*. Traditional *network equipment, i.e., network switches, is not involved in control-plane operations anymore: the control-plane can be executed on separated devices that communicate with network switches through well-established interfaces and protocols, introducing greater flexibility in the network, e.g., allowing the programmability of the control-plane*. Programmability is actually one of the most important properties of SDN: the control-plane behavior can be defined writing “network programs” that manage a set of switches, providing rich network applications and features. Programming the network, instead of configuring it, on one hand gives a powerful tool to develop revolutionary applications, but on the other, it also increases the difficulty of providing efficient and reliable networks: the flexibility of a program, like in the case of classical computer software, can lead to errors that are even more complex to handle, given the inherently distributed and asynchronous nature of a network. In some way, SDN is bringing into computer networks the same shift that in past decades electronic devices had from special purpose machines, to general purpose ones. With SDN, the current special purpose network can become a general purpose, hence programmable, network. Programmability, in a new context like computer networks, calls for new programming mod-



els, tools and languages. In this sense, to continue with the general purpose computer metaphor, SDN are still in the “machine language” phase, where the programming languages are rudimentary, strictly connected to the hardware, and the main part of network programming is still performed manually, without the aid of any tool. In last years, OpenFlow [101] has been a successful SDN implementation. The Open Networking Foundation<sup>1</sup> (ONF), that is responsible for the OpenFlow specification, currently involves a number of academic and industrial partners, making OpenFlow a promising technology, with a good number of industrial players already implementing the specification in their products. OpenFlow network switches configuration is executed using a rule-based approach: control-plane programs are written in a traditional high-level programming language, e.g., Python or Java, to provide a set of rules to be installed at different switches as output. In this paper we present a set of definitions to characterize the interactions of rules installed in an OpenFlow switch, and an algorithm to automatically detect such interactions, with the aim of aiding the development and debugging of OpenFlow network applications.

### 5.2.2 Related work

OpenFlow network programming is a problem addressed in some other works. Frenetic [49] is a high-level language based on the functional programming paradigm, that provides the programmer with an omniscient, centralized view of the network. A run-time system, linked to the language, “translates” the high-level instructions in a set of low-level packet processing rules, and manages them interacting with network equipments. NetCore [95] is an evolution of Frenetic, that extends the high-level language and provides some improvement in the compilation algorithms and run-time system, trying to speed up the network performance. To test the correctness of OpenFlow applications, NICE was proposed in [30]. NICE is a tool for automatic OpenFlow applications testing, that combines model checking and concolic

---

<sup>1</sup><https://www.opennetworking.org/>

execution to explore the state space of OpenFlow programs written for the NOX [55] controller platform. FlowChecker [3] uses manually built binary decision diagrams to encode OpenFlow rules and then applies model checking in order to detect OpenFlow switches misconfigurations. The definitions and the algorithm we present in this paper are targeted at identify and automatically detect OpenFlow rules interaction in a single switch, so our work can be seen as a basis for new OpenFlow tools targeted at applications development and debugging, that, differently from [3], requires no representation conversion of the rules.

### 5.2.3 Rule-based programming

OpenFlow-enabled Switches (OFS) behavior is configured using OpenFlow Rules (OFR), installed by means of the OpenFlow (OF) protocol [101]. An OFR is defined by the *match set*, that defines to which network flows the rule is applied, the *action set*, that defines the elaborations and the forwarding decision that must be applied to the matched flows, a *priority*, to relatively order among them the rules installed in a switch, and an *expiration time*, specified through the use of timeouts. According to the OF specification [105], only the highest priority OFR that matches a packet is applied to that packet.

Because of this definition, to define a switch behavior it is necessary to look at the whole set of installed rules. In fact, looking at a single rule does not suffice to understand the behavior of the switch in respect to the flow identified by the match set of such rule, since other rules, with higher priorities, can introduce a different behavior.

The problem is usually raised up in the process of developing an OF application, that is basically the process of defining when, where and what OFRs have to be installed at managed OFSes. This issue can be even more problematic if we are trying to extend an already developed OpenFlow application, or if we are combining several applications at the same time.

### 5.2.4 Interactions definition

To characterize the behavior of an OFS, we define OpenFlow rules interactions extending the work presented in [4]. An interaction is a particular relation between two OpenFlow rules. An interaction may be expected, i.e., the network programmer is aware that rules are interacting, or it may be raised by unexpected relations between rules, e.g., as result of a programming error. To define the possible interactions that can occur between two rules, we firstly define the relations that can be in place among match sets and among action sets. Then, based on such relations combination, we define rules interaction types.

#### Match sets relations

A match set is composed by a number of match fields. Typical match fields are *l2 source [destination] address*, *l2 protocol type*, *l3 protocol type*, *l3 source [destination] address*, etc. All match fields can have a wildcard as value, that means *any value*. Some match fields can have partially wildcarded values, e.g., an l3 address can be associated with a bitmask to specify which bits are wildcard. Because of the presence of wildcards, there are four different relations among two match fields of the same type. The relation between match field  $f_0$  and match field  $f_1$  can be one of the following: *disjoint*, when match fields have different values ( $f_0 \neq f_1$ ); *equal*, if  $f_0$  value is the same of  $f_1$  ( $f_0 = f_1$ ); *subset*, if  $f_0$  value is a subset of the value of  $f_1$  ( $f_0 \subset f_1$ ), e.g.  $f_0$  has a defined value, while  $f_1$  value is a wildcard; *superset*, when  $f_0$  value is a superset of the value of  $f_1$  ( $f_0 \supset f_1$ ), e.g.,  $f_0$  value is an IP address in the form *192.168.0.0/16*, while  $f_1$  value is *192.168.1.0/24*.

Using the defined match fields relations, we are now able to define the relations between two match sets. The relation between match set  $M_0$  and match set  $M_1$  can be one of the following:

*Disjoint*:  $M_0$  and  $M_1$  are *disjoint* if every field  $i$  in  $M_0$  is *disjoint* with the correspondent field in  $M_1$  ( $M_0 \neq M_1$ );

$$M_0 \neq M_1 \text{ if } \forall i \ f_i^0 \neq f_i^1, f_i^0 \in M_0 \wedge f_i^1 \in M_1$$

*Exactly matching:*  $M_0$  and  $M_1$  are *exactly matching* if every field  $i$  in  $M_0$  is *equal* to the correspondent field in  $M_1$  ( $M_0 = M_1$ );

$$M_0 = M_1 \text{ if } \forall i \mid f_i^0 = f_i^1, f_i^0 \in M_0 \wedge f_i^1 \in M_1$$

*Subset:*  $M_0$  is a *subset* of  $M_1$  if one field  $j$  of  $M_0$  is *subset* of the correspondent field of  $M_1$  and any other field  $i$  in  $M_0$  is *equal* or *subset* of the correspondent field in  $M_1$  ( $M_0 \subset M_1$ );

$$M_0 \subset M_1 \text{ if } (\exists j \mid f_j^0 \subset f_j^1) \wedge [\forall i \mid (f_i^0 = f_i^1) \vee (f_i^0 \subset f_i^1)] , \\ i \neq j \wedge f_i^0, f_j^0 \in M_0 \wedge f_i^1, f_j^1 \in M_1$$

*Superset:*  $M_0$  is a *superset* of  $M_1$  if one field  $j$  of  $M_0$  is *superset* of the correspondent field of  $M_1$  and any other field  $i$  in  $M_0$  is *equal* or *superset* of the correspondent field in  $M_1$  ( $M_0 \supset M_1$ );

$$M_0 \supset M_1 \text{ if } (\exists j \mid f_j^0 \supset f_j^1) \wedge [\forall i \mid (f_i^0 = f_i^1) \vee (f_i^0 \supset f_i^1)] , \\ i \neq j \wedge f_i^0, f_j^0 \in M_0 \wedge f_i^1, f_j^1 \in M_1$$

*Correlated:*  $M_0$  is *correlated* with  $M_1$  at least one field  $j$  of  $M_0$  is *superset* of the correspondent field of  $M_1$  and any other field  $i$  in  $M_0$  is *equal* or *subset* of the correspondent field in  $M_1$  ( $M_0 \sim M_1$ );

$$M_0 \sim M_1 \text{ if } \exists j \mid f_j^0 \supset f_j^1 \wedge \exists i \mid f_i^0 \subset f_i^1 , i \neq j \wedge f_i^0, f_j^0 \in M_0 \wedge f_i^1, f_j^1 \in M_1$$

### Action sets relations

An action set contains zero or more actions. An action has a type and a value, typical action types are *forward to port X*, *rewrite network source [destination] address*, *pop [push] VLAN tag*, etc. An action can be *equal*, *related* or *disjoint* in respect to an other action. An action  $a_0$  is *equal* to an action  $a_1$  ( $a_0 = a_1$ ) only if they have the same types and values, if the types are equal but values are different, the actions are *related* ( $a_0 \sim a_1$ ). An action  $a_0$  is disjoint from action  $a_1$  ( $a_0 \neq a_1$ ), if their types are different. Depending on the relations of the contained actions, the relation between action set  $A_0$  and action set  $A_1$  can be one of the following:

*Disjoint:*  $A_0$  is *disjoint* from  $A_1$  if for any action in  $A_0$ , such an action is disjoint from any action in  $A_1$  ( $A_0 \neq A_1$ );

$$A_0 \neq A_1 \text{ if } \forall i,j \ a_i^0 \neq a_j^1, a_i^0 \in A_0 \wedge a_j^1 \in M_1$$

*Related:*  $A_0$  is *related* to  $A_1$  if there is at least one action from  $A_0$  that is related to an action of  $A_1$  ( $A_0 \sim A_1$ );

$$A_0 \sim A_1 \text{ if } \exists i,j \ a_i^0 \sim a_j^1, a_i^0 \in A_0 \wedge a_j^1 \in M_1$$

*Subset:*  $A_0$  is a *subset* of  $A_1$  if all the actions contained in  $A_0$  are equal to actions contained in  $A_1$ , and  $A_1$  contains more action than  $A_0$  ( $A_0 \subset A_1$ );

$$A_0 \subset A_1 \text{ if } \forall i,j \ a_i^0 = a_j^1 \wedge |A_0| < |A_1|, a_i^0 \in A_0 \wedge a_j^1 \in M_1$$

*Superset:*  $A_0$  is a *superset* of  $A_1$  if all the actions contained in  $A_0$  are equal to actions contained in  $A_1$ , and  $A_0$  contains more action than  $A_1$  ( $A_0 \supset A_1$ );

$$A_0 \supset A_1 \text{ if } \forall i,j \ a_i^0 = a_j^1 \wedge |A_0| > |A_1|, a_i^0 \in A_0 \wedge a_j^1 \in M_1$$

*Equal:*  $A_0$  is *equal* to  $A_1$  if all the actions contained in  $A_0$  are equal to actions contained in  $A_1$ , and  $A_1$  and  $A_0$  contains the same number of actions ( $A_0 = A_1$ );

$$A_0 = A_1 \text{ if } \forall i,j \ a_i^0 = a_j^1 \wedge |A_0| = |A_1|, a_i^0 \in A_0 \wedge a_j^1 \in M_1$$

### Interaction types

To define interactions between two OFRs we look at rules' priorities, match sets relations and action sets relations. Considering an OFR  $R_x$ , with match set  $M_x$  and action set  $A_x$ , and an OFR  $R_y$ , with match set  $M_y$  and action set  $A_y$ , assuming that  $R_x$  priority is always smaller than  $R_y$  priority (If it is not the case we will explicitly point it out), The interaction between  $R_x$  and  $R_y$  can be of one of the types listed in table 5.1. Here we provide a brief description of them:

**Duplication:** assuming that the priorities of two rules are equal, they are duplicated if they have the same match and action sets.

Table 5.1: OpenFlow Rules interactions

MATCH SET	ACTION SET	PRIORITY
Duplication		
$M_x = M_y$	$A_x = A_y$	$prio(R_x) = prio(R_y)$
Redundancy		
$M_x \subset M_y$	$A_x = A_y$	$prio(R_x) < prio(R_y)$
$M_x \supset M_y$	$A_x = A_y$	$prio(R_x) < prio(R_y)$
$M_x \sim M_y$	$A_x = A_y$	$prio(R_x) < prio(R_y)$
$M_x = M_y$	$A_x = A_y$	$prio(R_x) < prio(R_y)$
Generalization		
$M_x \supset M_y$	$A_x \neq A_y$	$prio(R_x) < prio(R_y)$
$M_x \supset M_y$	$A_x \sim A_y$	$prio(R_x) < prio(R_y)$
$M_x \supset M_y$	$A_x \subset A_y$	$prio(R_x) < prio(R_y)$
Shadowing		
$M_x \subset M_y$	$A_x \neq A_y$	$prio(R_x) < prio(R_y)$
$M_x \subset M_y$	$A_x \sim A_y$	$prio(R_x) < prio(R_y)$
$M_x \subset M_y$	$A_x \supset A_y$	$prio(R_x) < prio(R_y)$
$M_x = M_y$	$A_x \neq A_y$	$prio(R_x) < prio(R_y)$
$M_x = M_y$	$A_x \sim A_y$	$prio(R_x) < prio(R_y)$
$M_x = M_y$	$A_x \supset A_y$	$prio(R_x) < prio(R_y)$
Correlation		
$M_x \sim M_y$	$A_x \neq A_y$	$prio(R_x) < prio(R_y)$
$M_x \sim M_y$	$A_x \sim A_y$	$prio(R_x) < prio(R_y)$
$M_x \sim M_y$	$A_x \subset A_y$	$prio(R_x) < prio(R_y)$
$M_x \sim M_y$	$A_x \supset A_y$	$prio(R_x) < prio(R_y)$
Inclusion		
$M_x = M_y$	$A_x \subset A_y$	$prio(R_x) < prio(R_y)$
$M_x \subset M_y$	$A_x \subset A_y$	$prio(R_x) < prio(R_y)$
Extension		
$M_x \supset M_y$	$A_x \supset A_y$	$prio(R_x) < prio(R_y)$

**Redundancy:** redundant rules have the same effect on the subset of flows matched by both rules, hence, in some conditions (e.g., no interactions with third rules), depending on the rules priorities, one of the rules could be deleted without affecting the datapath behavior, or the rules could be aggregated.

**Generalization:** rules have different actions, but  $R_x$  matches a superset

of the flows matched by  $R_y$ . So, action set  $A_y$  will be applied to flows matched by  $M_x \cap M_y$ , while to the flows matched by  $M_x - M_y$  the action set  $A_x$  will be applied.

**Shadowing:** if  $R_x$  is shadowed by  $R_y$ , then  $R_x$  is never applied, since all the flows are matched by  $R_y$  before that  $R_x$  is examined.

**Correlation:** the rules have different match sets, but the intersection of these match sets is not void, so, to flows that are in the intersection only higher priority rule's action set ( $A_y$ ) will be applied. Note that this anomaly is different from the *shadowing* interaction, since for some flows the lower priority rule ( $R_x$ ) is still applied.

**Inclusion:** *inclusion* interaction is similar to *shadowing*. The lower priority rule is never applied “as is”, but its actions are still applied in combination with the actions of another rule (of higher priority). I.e.,  $R_x$  is never applied, but, since the action set  $A_x$  is a subset of the action set  $A_y$ , the actions of  $A_x$  are still applied, but only in combination with the actions of  $A_y$ .

**Extension:** *extension* interaction is similar to *generalization*. A rule with lower priority is extending the action set applied by another rule, adding more actions. Only to the flows matched by  $M_x - M_y$  the extended actions are applied.

### 5.2.5 Interactions detection

To detect the interactions presented in previous sections, we designed an *interactions detection algorithm* (IDA). The algorithm takes two rules,  $R_x$  and  $R_y$ , assuming  $prio(R_x) \leq prio(R_y)$ , and detects the interaction generated by the composition of the rules. We assume that  $R_x$  and  $R_y$  are data structures containing all the information we need regarding an OpenFlow rule, i.e., match set, action set and priority. The algorithm uses two auxiliary algorithms to find match sets and action sets relations, respectively, these algorithms are represented by the functions *matchset\_relation*( $R_x, R_y$ ) and *actionset\_relation*( $R_x, R_y$ ).

**Algorithm 1** *matchset\_relation*( $R_x, R_y$ )

---

```

relation  $\leftarrow$  undetermined
field_relations  $\leftarrow$  compare_fields( $R_x, R_y$ )

for field in match_fields do

  if field_relations[field] = equal then
    if relation = undetermined then
      relation  $\leftarrow$  exact
    end if

  else if field_relations[field] = superset then

    if relation = subset or relation = correlated then
      relation  $\leftarrow$  correlated
    else if relation  $\neq$  disjoint then
      relation  $\leftarrow$  superset
    end if

  else if field_relations[field] = subset then

    if relation = superset or relation = correlated then
      relation  $\leftarrow$  correlated
    else if relation  $\neq$  disjoint then
      relation  $\leftarrow$  subset
    end if

  else
    relation  $\leftarrow$  disjoint
  end if

end for

return relation

```

---

Algorithm *matchset\_relation*( $R_x, R_y$ ) finds the relation between match sets, by firstly comparing match fields one by one, using the *compare\_fields*( $R_x, R_y$ ) function. Then it cycles among the fields' relations to evaluate the match sets relation, by applying a state machine where each state corresponds to one of the match sets relation (plus “undetermined”) and transitions corresponds to match fields relations <sup>2</sup>.

Algorithm *actionset\_relation*( $R_x, R_y$ ) is not presented in these pages, since its implementation is simpler and can be easily derived by the definitions of action sets relations presented in previous sections.

Algorithm *interaction\_detection*( $R_x, R_y$ ) uses the just defined functions to apply the interaction types definitions in order to find if the rules generate an interaction and, in that case, which type of interaction. We implemented a prototype of our algorithm in Python, integrating it into the NOX controller platform [55]. We performed a first evaluation of our implementation

---

<sup>2</sup>We are using a “foreach” notation in this *for* cycle, putting in the *field* variable, one by one, any value found in the *match\_fields* variable, that contains a list of the all possible fields name



**Algorithm 2** *interactions\_detection*( $R_x, R_y$ )

---

```

interaction  $\leftarrow$  None

ms_relation  $\leftarrow$  matchset_relation( $R_x, R_y$ )
as_relation  $\leftarrow$  actionset_relation( $R_x, R_y$ )

if priority( $R_x$ ) = priority( $R_y$ ) and ms_relation = exact and as_relation = equal then
    interaction  $\leftarrow$  duplication
else if ms_relation  $\neq$  disjoint then
    if ms_relation = correlated then
        if as_relation = equal then
            interaction  $\leftarrow$  redundancy
        else
            interaction  $\leftarrow$  correlation
        end if
    else if ms_relation = superset then
        if as_relation = equal then
            interaction  $\leftarrow$  redundancy
        else if as_relation = superset then
            interaction  $\leftarrow$  extension
        else
            interaction  $\leftarrow$  generalization
        end if
    else if ms_relation = exact then
        if as_relation = equal then
            interaction  $\leftarrow$  redundancy
        else if as_relation = subset then
            interaction  $\leftarrow$  inclusion
        else
            interaction  $\leftarrow$  shadowing
        end if
    else if ms_relation = subset then
        if as_relation = equal then
            interaction  $\leftarrow$  redundancy
        else if as_relation = subset then
            interaction  $\leftarrow$  inclusion
        else
            interaction  $\leftarrow$  shadowing
        end if
    end if
end if

return interaction

```

---

using randomly generated rule sets. We generated rule sets defining three parameters: (i) number of rules in the set, (ii) number of non-wildcard match fields, (iii) probability of generating the same value for match fields belonging to different rules. This last parameter provides a mean to set the number of interactions in the rule set, e.g., a low probability corresponds to fewer interactions in the rule set. Action sets were also generated randomly, selecting from 1 to 3 actions per action set. For each rule set, we run the algorithm several times to extract a mean of the running times. The testbed machine is an Ubuntu Linux virtual machine, equipped with 2 GB of RAM and running on a dedicated cpu-core of an Intel CPU E7600 @ 3.06GHz. The execution times are shown in figure 5.6: our first implementation is pro-

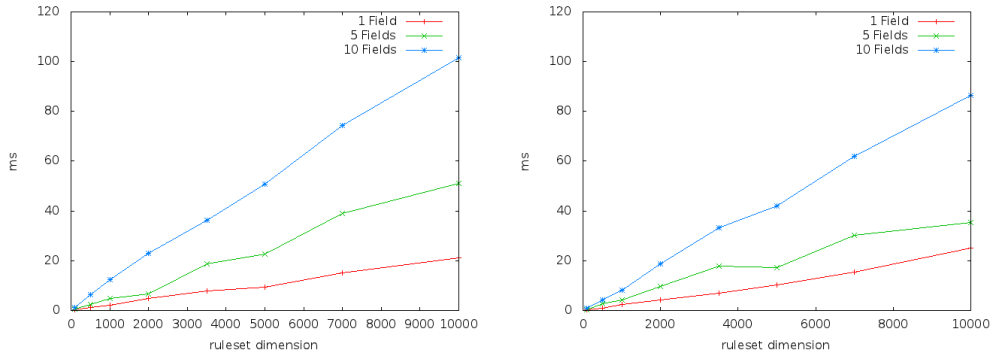


Figure 5.6: Interactions detection algorithm performance with low (left) and high (right) match sets overlapping values probability. Rules in rule set have from 1 to 10 non-wildcard fields.

viding a linear increasing of the execution time with the growing of the rule set dimension. Moreover, the greater is the number of wildcard match fields, the lower is the execution time. Interestingly, the current implementation provides better performance when the number of anomalies in the rule set is bigger (Right graph of figure 5.6).

### 5.2.6 Use cases

The algorithm and the interactions definition we provided in this paper can be used as a basis for OpenFlow network development methodologies and for tools targeted at aiding OpenFlow network programming, analysis, switches management and optimization, etc.. In this section we briefly describe some possible applications, then, we present a concrete use case. During the development of an OpenFlow application, the IDA can be used as debug tool to verify the interactions among rules installed in a switch. In the simplest cases, it can point out rules duplications and redundancies, reducing any overhead in the developed application, or it can detect unexpected rules interactions that would lead to the wrong handling of some traffic flows. IDA could be also integrated in advanced controller platforms as a mean to analyze rules in order to provide some forms of automation in rules management. E.g., rejecting duplicated rules, reordering rule priorities to avoid

Table 5.2: OF-Switch installed rules

#	Matching criteria				Priority	Actions
	DL_DST	DL_TYPE	NW_SRC	NW_DST		
1	$MAC_{routerA}$	IP	*	<i>identifier</i>	200	set NW_DST: <i>locator</i> ; out: $PORT_{routerA}$
2	$MAC_x$	IP	<i>locator</i>	*	200	set NW_SRC: <i>identifier</i> ; out: $PORT_x$
3	$MAC_{routerA}$	*	*	*	100	out: $PORT_{routerA}$
4	$MAC_x$	*	*	*	100	out: $PORT_x$

shadowing, rule splitting to avoid redundancy and correlation, etc.. An advanced controller would require more work on the semantic part of the rules management, but the interactions detection algorithm is the enabling technology to analyze rule interactions. A different application would be the use of IDA to compare a rule against a set of rules that is used as an admission control policy. Specifying the allowed interactions with the given policy rule set, a rule can be checked to be admitted or not. Complex policies can specify which operations are allowed on which flows, using properly specified rules set and allowed interactions. As last application example, the IDA can be used to optimize the rules installed in a switch. Rules can be checked against other rules to find interactions, and, in case, they can be rewritten to split or aggregate them for a better use of the switch hardware resources (E.g., some switches have multiple tables with different properties to install rules). Clearly, we presented only a few examples of possible IDA exploitations. In the following part of this section we briefly introduce a concrete application for the development of a real OpenFlow network application.

### Extending an OpenFlow application

Follow-Me Cloud (FMC) is a technology, developed at NEC Laboratories Europe, that provides mobility features in a TCP/IP network for both users and services, maintaining all the ongoing network connections active. FMC is applied to a TCP/IP network in which L2 access networks are connected to a “core” network, that provides connectivity among them, through OpenFlow-enabled switches (OFS). To provide mobility to a mobile node (MN) that is changing its access network from an “home” to a “foreign” network, FMC requires that a new IP address, belonging to the foreign network, is assigned to MN to work as “locator”. The original IP address of MN, that belongs to

the home network, is still used by MN itself and by any node that is communicating with MN, since it works as “identifier”. When a network node (we call such a node *correspondent node* or CN in short) sends a packet to MN, it uses the *identifier* address as destination address. The OFS connecting the CN’s network (CNet) to the core network performs an address translation, to substitute the *identifier* with the *locator* address. When the packet reaches the foreign network, the OFS at the edge of such network performs a new translation, substituting the *locator* with the *identifier*, in order to deliver the original packet to the MN.

The current FMC implementation uses NOX [55] as controller platform. To make deployment of FMC in a TCP/IP network as easy as the placement of OFSes at the edge of L2 access networks, we decided to extend an OpenFlow learning switch application with FMC functionalities. A learning switch (LS) application provides Ethernet Switch functionalities, by learning MAC addresses and associating them with switch ports. LS installs proper OFRs to forward to the correct port a packet with a given destination MAC address. Rules 3 and 4 from table 5.2 are a typical example of two rules installed by the LS application, to provide connectivity among a node  $X$  and the gateway of the  $X$ ’s network (*routerA*). OFSes are controlled by the learning switch application, ensuring traditional TCP/IP operations, while only the flows directed to mobile nodes are handled by FMC-related OFRs. As an example, the addition of FMC to a CNet’s OFS requires that a destination address translation is performed on any flow destined to an *identifier* address. At the same time, a source address translation must be performed on any flow with *locator* as source network address (See rules 1 and 2 from table 5.2). We have to ensure that FMC-related OFRs are not shadowed by LS-related OFRs. Our FMC implementation uses the interactions detection algorithm to guarantee that newly installed rules are always involved in a generalization interactions with LS-related OFRs, i.e., LS-related OFRs are generalizing FMC-related OFRs. The algorithm has been integrated into the FMC OpenFlow Controller and it is used as a runtime tool, to define the re-

quired priority value to assign to newly generated OFRs, and as debug and validation tool, to check the absence of shadowing interactions into switches.

### 5.2.7 Conclusions and future work

In this section we presented a formal definition for the interactions of OpenFlow rules installed in a OpenFlow Switch, an algorithm for the automatic detection of such interactions and we showed how the algorithm has been used to develop a real OpenFlow application. Furthermore, we evaluated our prototype implementation of the proposed algorithm to understand the actual applicability in other real-world scenarios. Our evaluation shows that the dimension of the managed rule set can limit the applicability of the algorithm as a runtime tool for the definition of a complex management policy, since the execution times, when the number of rules is in the order of thousands, can negatively affects the network performance. Anyway, the algorithm is well suited for the development phase of OpenFlow applications, e.g., as debug tool, or in applications where the number of managed rules per switch is no more than a few hundreds. In future work we plan to improve the algorithm execution times by exploiting, e.g., smart ordering of rules or rule set reduction strategies. Moreover, we are looking for the application of the algorithm to real-world applications in order to validate and extend the rule interactions definition we provided.

## Chapter 6

# Towards service oriented networks

With networks mainly used to access services, the current Internet model based on communicating processes could be limiting and not efficient. In this chapter, we introduce how to build a network oriented to services without changing the current networking infrastructure. The approach is based on the simple concept of separation of network identifiers from network locators: the concept is well applied to network nodes, but it is actually also the first step needed to decouple network nodes from the services they are implementing, making the network identifier to be related to a service name, whose implementation can be realized in several network nodes. The work presented in this chapter provides the identifier and locator separation without changing the current network protocol stack, by introducing SDN devices in the network in strategic locations. Using SDN, the operations realized by the presented solution are actually providing an overloading of the network protocols semantic, for instance, changing the meaning of the IP addresses fields. We present both the architecture and its design together with a scalability analysis, moreover, we introduce the implementation details and a first performance evaluation of sensible metrics.

## 6.1 Follow-Me Cloud

### 6.1.1 Introduction

Rich media services that may be accessed anywhere are expected to play a significant role in the mobile apps environment in the next few years, due to their ability to generate significant revenues. These applications have become technically feasible thanks to the ubiquitous availability of multimedia devices and broadband connectivity. Nonetheless, the ability of provisioning such services to large numbers of mobile users is still a technical challenge for service providers. Service provisioning, today, finds in the emerging Cloud Computing paradigm a flexible and economically efficient solution, in particular for small and medium enterprises that do not want to invest huge capitals for creating and managing their own IT infrastructures.

The basic tenet of cloud computing is that end users do not need to care about where a service is actually hosted, while service providers may dynamically acquire the resources they need for service provisioning in a pay-per-use model. While for most of elastic web applications the relative position of client and server end-systems does not affect the perceived *Quality of Experience*, provided enough bandwidth is available in the end-to-end path connecting clients with servers, rich interactive applications are sensible to other communication metrics, such as delay and jitter. In the absence of explicit QoS control mechanisms in the network, the only way to improve Quality of Experience is to locate servers as close as possible to user terminals. Such an approach, largely exploited by *Content Delivery Networks*, can be further advanced in the era of Cloud Computing. Assuming that several cloud-enabled datacenters are made available at the edges of the Internet, service providers may take advantage of them for optimally locating service instances as close as possible to their users. In such a context, mobility of user terminals makes such location decisions even more difficult.

In this section, we present *Follow-Me Cloud* (FMC), a technology developed at NEC Laboratories Europe to overcome the current TCP/IP archi-

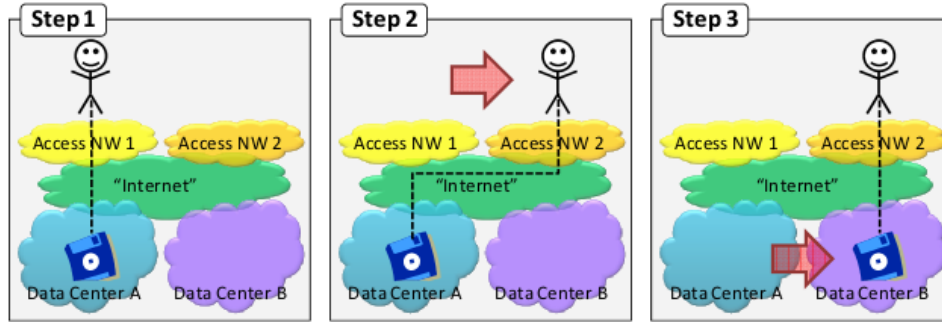


Figure 6.1: Follow-Me Cloud Use Case

architecture mobility limitations and to support novel Mobile Cloud Computing applications, by providing both the ability to migrate network end-points and to reactively relocate network services depending on users' locations, in order to guarantee adequate performance for the client-server communication and, at the same time, have a precise control on the use of network resources by localizing network traffic generated by applications. The FMC architecture is based on cooperating *FMC controllers* located in the networks of collaborating operators. FMC controllers modify packet forwarding in such a way that location changes of users and of services (e.g., through migration of Virtual Machines) are transparently managed by the network infrastructure, without any need of reconfiguring the end systems. Depending on the users' mobility patterns, FMC control-plane decide if, when and where network services have to be migrated.

The coarse flow of actions performed by FMC to manage a migrating service is shown in figure 6.1. In this simplified use case, initially, a user accesses a dedicated service, e.g., a remote desktop application, in the "home" environment (i.e., usually, his/her office or home). He/She then goes on a trip and, while on the move, accesses the service through an app running in a smartphone. Over time, the smartphone gets connected to the Internet through different mobile network operators. After a change of the terminal's network attachment point, FMC may decide to trigger migration of the service instance to a different network location, e.g. a data center, closer



to the new network attachment point. As shown in step 3, the application migrated to the new data center, providing the user with improved Quality of Experience and/or reducing the service provisioning cost for the network operator.

Follow-Me Cloud is implemented using OpenFlow (OF) [101]. It uses the packet filtering and rewriting capabilities of OpenFlow to achieve the seamlessness of migration, and configures network equipment through OpenFlow rules (OFR). OF-enabled networking equipments (i.e., switches) are therefore needed in the network for FMC to work. Nevertheless, OF equipments are only needed at the edges of the network.

In this section we present how FMC achieves network end-points mobility and our solution to solve scalability issues. Cloud services reactive relocation strategies are not discussed here, and will be the topic of future work.

### 6.1.2 Related work

The idea of exploiting the live migration capabilities of modern virtualization technologies for dynamically changing the position of a service instance on a geographic basis has been already proposed in the past (e.g. in [85]). Live migration of Virtual Machines is a common practice in virtualized data centers, in which the internal networking infrastructure may be designed as a large single IP subnet. Migration of VMs across the boundaries of a single IP subnet, on the other hand, is not straightforward, as the TCP/IP protocol stack does not provide the needed flexibility in terms of mobility support for network end-points. A technique for the migration of network end-points in a data-center environment by means of a coordinated set of agents is presented in [21]. A first example of the application of OpenFlow to solve network mobility issues is presented in [45], where OpenFlow is used to migrate VMs among different IP subnets, using a network fully composed of OpenFlow switches.

None of these papers present solutions to the control-plane scalability problems that derive from the application of these techniques in large scale

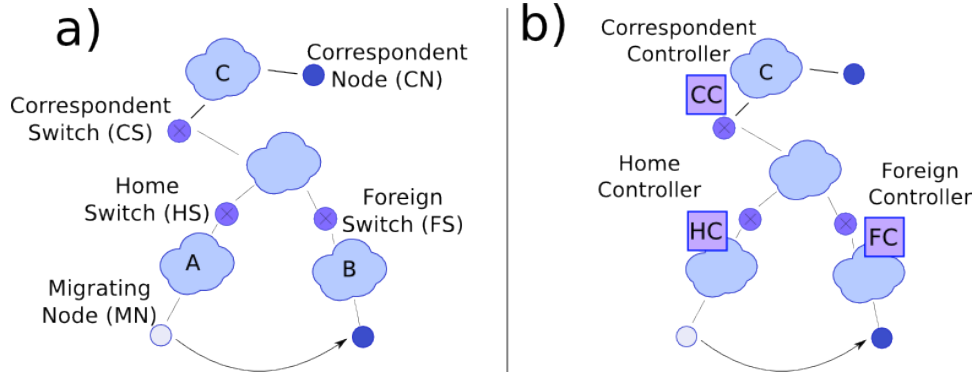


Figure 6.2: a. End-points mobility reference scenario; b. FMC distributed architecture

networks.

Other approaches, like the *Locator/Identifier Separation Protocol* [46] (LISP), are based on the concept of separation among *locator* and *identifier* addresses. The LISP architecture uses an “alternate topology” to distribute ID/LOC mapping information among routers in the network. LISP does not provide mobility support natively, while it is added in its “mobility extensions” that need direct involvement of mobile devices in the ID/LOC mapping.

MobileIP and MobileIPv6 both enable end-point mobility, but require direct involvement of the moving entities. On the other end, Proxy MobileIP provides mobility without involving end-points, by placing mobility aware proxy devices in the network. In any case, in MobileIP the forwarding of the data packets is usually performed through tunnels and/or triangular routing that are far less efficient in comparison to the FMC solution.

### 6.1.3 End-Points Mobility

Follow-Me Cloud (FMC) enables mobility of network end-points among different IP subnets in a TCP/IP network, both in closed environments, such as data-centers, and on a geographic scale, maintaining all the ongoing network communications of the moving entity active and requiring no modifications to the involved end-points. FMC is applied to a TCP/IP network in which ac-

cess networks are connected to a “core” network, that provides connectivity among them, through OpenFlow-enabled switches (OFS). Current TCP/IP network architecture uses a single address to both identify and locate a device on the network, making the network unable to support mobility natively. FMC realizes the split of identifier and locator concepts in the edge network, using the OFSes to enforce the splitting in a transparent manner for network end-points. Figure 6.2.a shows a typical application scenario, with three access networks, and explains also the names used to identify all the network devices involved. Names are assigned from the perspective of a particular migrating node (MN).

Using FMC, the MN can migrate from an access network A, to an access network B, without changing its network configuration (e.g., IP address, Gateway Address). From a network perspective, MN is totally unaware that the access network on which it is residing is changed. All the ongoing communications are kept active, e.g., TCP sessions are not lost. Any correspondent node, i.e., any node that is on an access network different from A or B and that is communicating with MN, is unaware of the MN location change as well. To provide this result, FMC requires that a new IP address, belonging to the B network, is assigned to MN to work as “locator”. The original IP address of MN is still used by MN itself and by any node that is communicating with MN, since it works as “identifier”. For any migrated node, the FMC controller (FMC-C) stores the identifier/locator mapping information, that is used to configure involved OFSes with proper OpenFlow rules. The outcome of FMC operations is that each packet destined to a migrated endpoint, before traversing the core of the network, is processed to substitute the identifier address with the locator address. Then, the locator address is substituted again with the identifier address, before the packet is delivered to MN. A similar address translation is performed on the source address of packets sent by MN. Hence, the *identifier* address is used to send/receive packets in the edges of the network, while the *locator* address is used in the core of the network, to forward packets to the correct location. Border

devices, i.e., OFSes, are in charge of performing the identifier/locator and locator/identifier translations.

#### 6.1.4 Scalability

FMC uses OpenFlow to provide transparent identifier/locator (ID/LOC in short) splitting, introducing some OFRs to support the redirection of packets to the new location of a migrated node. Even though the used approach is trying to be lightweight, requiring no modifications to the traditional IP routing, when a migration happens, several entities are involved in communication at the network control plane. In particular, FMC controllers need to coordinate their activities, and various OFSes need updated forwarding rules.

In this section we are aiming at assessing the scalability of FMC. Our work is mainly focused on evaluating scalability from the perspective of the number of managed OFRs. We see two main issues: (i) how many rules need to be installed on a particular OFS, and (ii) how many rules must be managed by the FMC controllers. We define the former as a *data-plane* scalability issue and the latter as a *control-plane* scalability issue. *Data-plane* scalability affects the ability of applying FMC when an OFS is involved in many IP address migrations, i.e., when a large number of rules must be installed. An OFS has limited capacity in the number of rules it is able to support, that means, from FMC perspective, that there is a limit on the number of concurrent migrations that can be provided for the network served by that switch. Even if this issue can be a serious problem, and it is worth to be investigated, for the purpose of FMC we assume that it is solvable using a careful partitioning of the network, i.e., reducing the dimension of the network served by a single switch. Clearly, using this *scale-out* approach, we are adding more network devices, hence, we are potentially increasing the work load on the control-plane.

*Control-plane* scalability is, in fact, related to the number of devices and to the total number of OFRs that must be managed by FMC controllers. The

number of devices is strictly dependent on the network dimension and on the partitioning level we are applying, e.g., to solve the data-plane scalability issue. The total number of rules is instead directly dependent on the total number of concurrent IP address migrations. To evaluate the total number of generated rules we use the simplistic assumption that there is only one centralized FMC controller<sup>1</sup>. Moreover, we assume that this controller knows in advance on which OFSes rule installation is needed. This way we are able to evaluate the number of rules needed for the packet forwarding redirection, without taking into account implementation-dependent rules. The number of generated rules for the  $i$ -th IP address migration is:

$$(6.1) \quad R_i = r_i^{hs} + r_i^{fs} + \sum_j r_{ij}^{cs}$$

where  $r_i^{hs}$  and  $r_i^{fs}$  are the number of rules installed at the home switch (HS) and foreign switch (FS) for the  $i$ -th IP address migration, and  $r_{ij}^{cs}$  is the number of rules installed at the  $j$ -th correspondent switch that is exchanging packets with the  $i$ -th migrated IP address. The number of rules for each migrated IP address is given by the following formulas:

$$(6.2) \quad r_i^{hs} = \alpha + H_i$$

$$(6.3) \quad r_i^{fs} = \beta + F_i$$

$$(6.4) \quad r_{ij}^{cs} = \gamma + C_{ij}$$

The variables  $H_i$ ,  $F_i$  and  $C_{ij}$  represent the number of nodes from home, foreign and  $j$ -th correspondent networks that are exchanging packets with the  $i$ -th migrated IP address. The constants  $\alpha$ ,  $\beta$  and  $\gamma$  are the fixed number of rules required by FMC for packet redirection per migration<sup>2</sup>. The total

---

<sup>1</sup>The OpenFlow architecture actually suggests a centralized approach to the network control-plane development, through the use of a centralized OpenFlow controller that is connected to a number of OFSes.

<sup>2</sup>In the current implementation  $\alpha = 3$ ,  $\beta = 6$  and  $\gamma = 3$

number of OFRs managed by the FMC controller is the sum over  $i$  of the rules as expressed in (6.1), plus some rules installed once for each HS or FS:

$$(6.5) \quad R = 2N + \sum_i R_i$$

where  $N$  is the total number of HSeS (and since one FS corresponds to each HS, it is also the total number of FSeS) involved in IP addresses migrations. From equations (6.1) and (6.5) it is clear that the number of rules is directly proportional to the number of concurrent migrations, and to the number of nodes on the HN, FN and CNs that are exchanging packets with migrated addresses.

### 6.1.5 Distributed Controller

Providing scalability at the control-plane is a key requirement to make FMC usable in large-scale networks and it is the focus of the work presented in this section. Our aim is to enable the balancing of the FMC operations load among a number of FMC controllers, actually building a FMC distributed controller. Building a distributed controller provides a twofold outcome: apart from scalability, it adds also the flexibility and the distribution of responsibilities required to enable the use of FMC functionalities across administrative boundaries.

The design of the distributed controller follows the principle of distributing knowledge to where it is actually needed. In our case, the needed information at a particular controller is the *identifier/locator* mapping for a given network entity, while the controllers that need to know about this mapping are the ones that are controlling the HS, the FS and CSeS. To manage this information and to distribute it among controllers, we designed the architecture depicted in Figure 6.2.b. With respect to the migrating node, we identify three different roles among FMC controllers: *Home Controller (HC)* that controls the network to which the *identifier* address belongs to; *Foreign Controller (FC)* that controls the network to which the *locator* address belongs to; *Correspondent Controller (CC)* that controls one or more CSeS.

The architecture is flexible enough to enable a single controller to play one, two or all the roles for the same migrating node, e.g., because the same controller is in charge of managing multiple networks. This approach also offers the possibility to adapt the number of controllers used in the network, in order to tackle the actual network load: it is possible to use a scale-out approach, increasing the number of access networks, and consequently reducing the number of nodes per access network. Then, each controller is assigned a number of access networks to manage (the range is from one to all the access networks), with the aim of sharing the overall load.

Since the MN's identifier address belongs to a network managed by HC, this controller is always involved in any MN migration, making it a good candidate to be an "authoritative" repository for the MN mapping information. HC has therefore been selected to be in charge of managing the mapping information for the MN while it is away from its home location.

When MN migrates to a *foreign network* (FN), the HC is notified about the migration and informs the FC that MN is migrating to a network that FC itself is managing. Since FC is in charge of managing the network to which the locator belongs to, we leave to FC the responsibility to generate the locator address for MN<sup>3</sup>. Once the locator is defined, it is sent back to the HC so that both FMC-Cs have the complete information about the ID/LOC mapping to perform the required configurations.

So far, just HC and FC are informed and configured to support the migration. Eventually, a CC will need the information about the ID/LOC mapping, because some end-points connected to CSes are trying to establish communication with MN. While in the case of HC and FC we know which are the HS and FS, since they are source and destination of the migration, we do not have this information for the CSes. At any point in time, a network entity can be involved in a communication session with MN, and, differently from what we did in section 6.1.4, we are not assuming to know in advance

---

<sup>3</sup>How the locator address is actually obtained is out of the scope of this work, anyway, it is possible to interact with a network management system, or with a DHCP server to easily generate it.

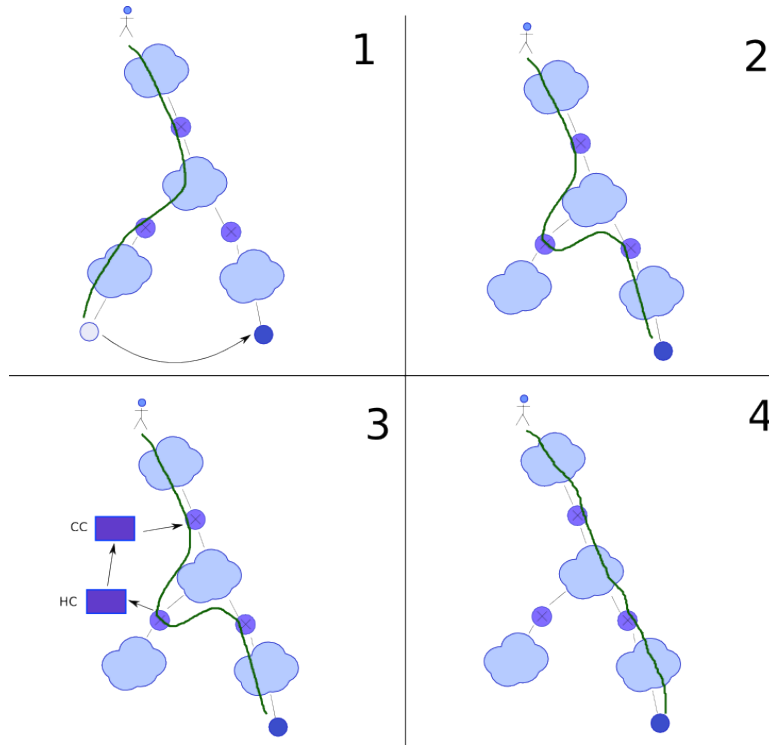


Figure 6.3: Overview of the reactive update of the identifier/locator mapping at the CC.

who is going to start this communication. To handle this issue, our architecture uses a reactive approach: once a new communication with a CN is established, the involved CC is updated properly. We have to distinguish two cases: (i) the correspondent node starts the communication, and (ii) the migrating node starts the communication.

In the first case, when the CN sends a packet to MN, it always uses the identifier address as the packet's destination. Since the CS does not know yet about the updated ID/LOC mapping, it performs no rewriting on the packet, which is therefore forwarded along the route to MN's home network. Once the packet reaches the HS, it is intercepted and a message to update the ID/LOC mapping at the CC is sent. The packet is then forwarded by substituting the identifier with the locator, using triangular routing. Figure 6.3 shows an overview of the operations, while figure 6.4 shows the details of the interactions among HC and CC.



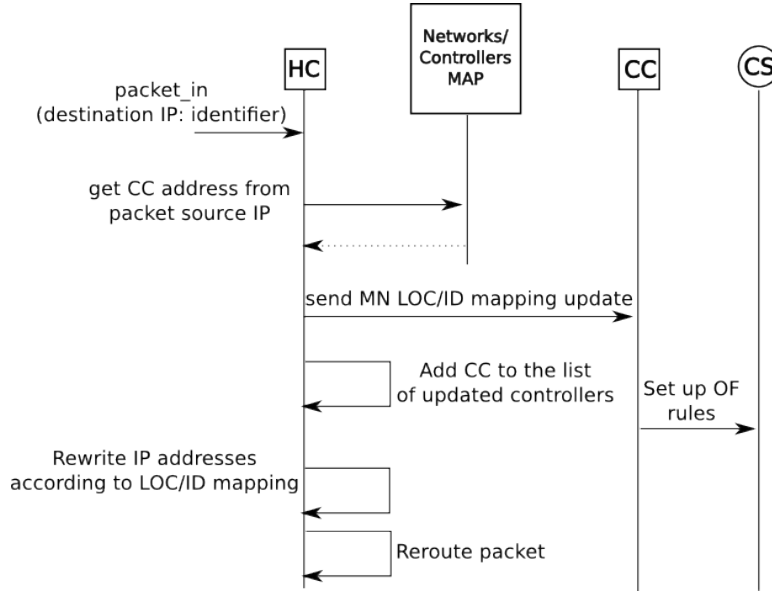


Figure 6.4: Reactive update of the identifier/locator mapping at the CC.

We point out here that it is necessary to store the information of which CC has been updated with the ID/LOC mapping information, since subsequent migrations will trigger an update for all the stored CCs. Otherwise, packets generated by CNs will continue to be directed to the old MN's location.

In the second case, when MN starts the communication with a CN, we have to take into account that the FS is already applying the ID/LOC mapping. Since, on the other hand, the CS does not know yet about the ID/LOC mapping, the packet would reach the CN using the locator address as source, then, CN would use the locator address to talk with the migrated node. If no more migrations happen for MN, there are no problems with this behavior, but, if a new migration happens, then the locator changes, hence, the communication with CN is lost.

Because of this issue, we should ensure that any entity always uses the identifier as destination address. To provide the afore mentioned property, FC has to intercept any packet that is sent by the migrated node and is destined to a CN, whose network's FMC controller is still not updated about the ID/LOC mapping for MN. This way, FC is able to send updated ID/LOC

information for MN to CC. It is worth to note that the FC does not inform HC about updated CC while MN is still located in its administrative domain. Instead, FC maintains a local list of updated CCs that is sent back to HC when a new MN migration happens.

When MN migrates back to its home network, or to a new network, all the ID/LOC mapping information distributed at different FMC controllers are no longer valid. As for any other MN migration, when HC is informed that MN is migrating, it updates the “old” FC about the MN location change. During this interaction, FC sends its local list of updated CC to HC. Then, HC updates the ID/LOC mapping information on any previously updated CC.

### 6.1.6 Evaluation

We tested our prototype implementation both on a Mininet[75] testbed and on a physical testbed equipped with NEC OpenFlow switches. With regard to the scalability of the distributed architecture, to evaluate the total number of rules we have to modify the  $r_i^{fs}$  expression, substituting the definition given in (6.3) with the following formula, to take into account the reactive update of ID/LOC mapping information:

$$(6.6) \quad r_i^{fs} = \delta + F_i + J_i$$

where  $J_i$  is the number of CNets exchanging packets with the  $i$ -th migrated node, and  $\delta$  is a fixed number of OFRs.

In figure 6.5 a comparison of the number of rules managed in the case of a centralized FMC controller is compared to the number of rules managed by the most loaded FMC controller of the distributed architecture, i.e., the FC. The number of rules is evaluated for one migration, with a linearly increasing number of nodes that are exchanging packets with MN and that are located at home networks (HNs), foreign networks (FNs), and correspondent networks (CNets).

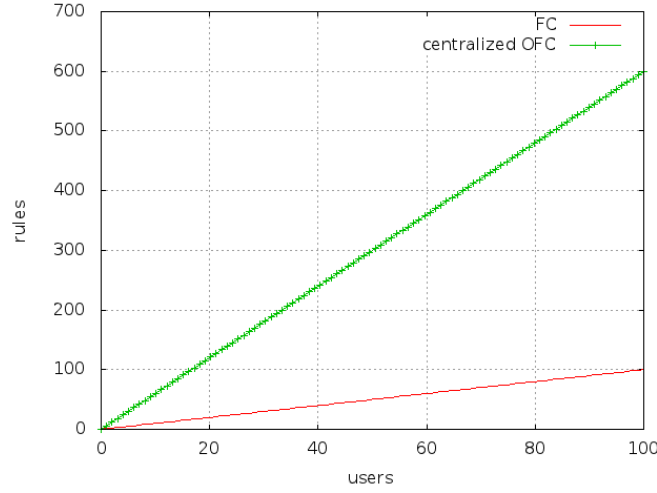


Figure 6.5: Number of rules managed by centralized FMC-C and FC, for one migration.

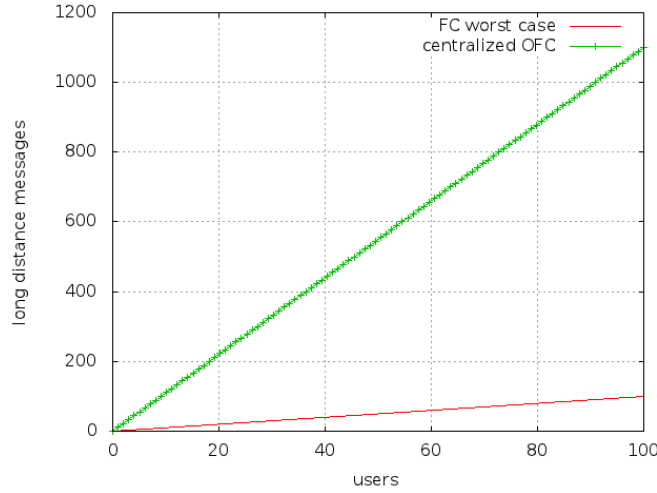


Figure 6.6: Number of “long distance” messages sent or received, for one migration.

Taking into account also the OFRs installation time, we have to consider the issue that arises when an FMC controller is located far away from the switch it is controlling, since the network delay has to be added to the installation time. Distributing the architecture enables a better and faster handling of rules installation: even if FMC-related rule installation still needs to face some network delays because of the coordination among the controllers, the majority of actions are performed locally at the controller, that is typically

placed close to the network it is controlling.

Assume a scenario in which managed networks are far apart and further assume that a single centralized controller is placed near the FS which is the switch that requires the most rule installation messages. With this deployment, any message exchanged between the controller and HSeS/CSes experiences high delay. In the distributed architecture, instead, only inter-FMC controllers messages experience high delay. The number of required “long distance” messages is linearly increasing with the number of nodes in HN, FN and the number of CNets, as shown in figure 6.6. The figure compares the centralized FMC controller case to the FC of the distributed architecture, in the worst case (i.e., when FC is in charge of updating CCs).

### 6.1.7 Conclusions and future work

In this section we introduced Follow-Me Cloud, a technology that provides mobility features in a TCP/IP network for both users and services, using OpenFlow-enabled equipment at the edges of the network. We presented the implemented mobility technique, the distributed architecture used to support the operations and an early scalability evaluation of the developed prototype. The current implementation and developed test-beds provide a solid base for further research and development. In particular, the logic for deciding when and where to migrate services to needs to be realized. In addition to the original use case of supporting mobile users, it turned out that FMC technology can be used for other scenarios as well, including the general problem of cross-operator service migration and the migration of core mobile network components. These are just two examples of potential use cases for Follow-Me Cloud technology, highlighting its applicability in many domains.

## 6.2 Implementing FMC

### 6.2.1 Introduction

Software Defined Networking (SDN) suggests the separation of control and data planes, providing well defined interfaces among them, in order to enable flexible configurability and programmability of the network. Programmability is one of the characterizing properties of Software Defined Networks: the control-plane behavior can be defined writing “network programs” that manage a set of switches, providing rich network applications and features. In some way, SDN is bringing into computer networks the same shift that in past decades electronic devices had from special purpose machines, to general purpose ones. With SDN, the current special purpose network can become a general purpose, hence programmable, network. Programmability, in a new context like computer networks, calls for new programming models, tools and languages. In this sense, to continue with the general purpose computer metaphor, SDN are still in the “machine language” phase, where the programming languages are rudimentary, strictly connected to the hardware, and the main part of network programming is still performed manually, without the aid of any tool. The potentially increased complexity, paid to gain in flexibility, can be tamed by using appropriate abstractions and methodologies, as it happens in the software engineering field. Methodologies, abstractions and tools have to address the complexity taking into account, at the same time, classical networks issues, such as scalability.

OpenFlow is one of the most popular SDN-enabling technologies. OpenFlow was born as a means to enable network experiments on campus networks[89], and its first deployments were actually universities’ networks. Over time, the advantages of an SDN approach to networks have been explored, leading to applications of OpenFlow to other environments, such as enterprise networks, as in the OpenFlow implementation of the Ethane architecture for network security[32]. More recently, OpenFlow has been also applied to challenging scenarios like datacenter networks[141] and wide-area networks[69]. The

Open Networking Foundation [104] (ONF), that is responsible for the OpenFlow specification, currently involves a number of academic and industrial partners. An increasing number of device manufacturers have implemented OpenFlow in their products and Google recently declared the adoption of OpenFlow in its networks[150].

This work presents a practical experience in developing the distributed OpenFlow controller for supporting Follow-Me Cloud (FMC)[22]. We introduce the problems we faced in developing the FMC controller and the solutions we adopted in terms of programming methodology and abstractions. In particular, we highlight the scalability issues to be taken into account while developing a controller, how our design describes the network through an object model and handles operations to provide scalability and extendability.

### 6.2.2 Follow-Me Cloud

There are several issues to be solved in order to make FMC usable. In particular, (i) FMC must scale with the number of users and migrations<sup>4</sup>, and (ii) must be easily deployable in traditional networks.

Scalability is provided by a distributed architecture. The design of the distributed architecture follows the principle of distributing knowledge to where it is actually needed. The needed information at a particular network is the *identifier/locator* mapping for a given network entity, while the networks that need to know about this mapping are the ones that are connected to the core network through the Home Switch (HS), the Foreign Switch (FS) or Correspondent Switches (CS). To manage this information and to distribute it among networks, FMC uses the architecture depicted in Figure 6.2.b and described in paragraph 6.2.2. With respect to the migrating node, we recall that the FMC architecture comprises three different roles: *Home Controller (HC)* that controls the network to which the *identifier* address belongs to; *Foreign Controller (FC)* that controls the network to which the *locator* address belongs to; *Correspondent Controller (CC)* that controls one

---

<sup>4</sup>Scalability of FMC is discussed in [22]

or more CSes. The architecture is flexible enough to enable a single controller to play one, two or all the roles for the same migrating node, e.g., because the same controller is in charge of managing multiple networks. This approach also offers the possibility to adapt the number of controllers used in the network, in order to tackle the actual network load.

To make deployment of FMC in a TCP/IP network as easy as the placement of OFSes at the edge of L2 access networks, FMC managed OFSes provides traditional switching functions in addition to FMC ones. E.g., OFSes work as Learning Switches (LS), i.e., they learn MAC addresses and associate them with switch ports. When a node migration has to be handled, the switch extends the packet handling with FMC functions.

### 6.2.3 Controller design

The presented logical architecture has been implemented as a distributed OpenFlow controller on top of the NOX[55] OpenFlow Controller Framework (OCF), even if we believe that all the concepts can be ported to any other OCF. In this paper, we are referring to NOX as an OCF, even if its authors define it as a Network Operating System (NOS) in [55]. NOX provides a set of helper methods and APIs to interact with OpenFlow switches, while we assume that a NOS should also provide advanced hardware and programming abstractions. Moreover, we use the term “OpenFlow controller” to identify the combination of an OCF with the OF applications running on top of it.

From a programming perspective, to support FMC operations, the controller has to provide these features:

1. it should easily become a distributed application if needed, i.e., different parts of the controller should be able to be moved to different computing nodes (in a different network location);
2. it must be extensible, providing the ability to combine different network functions, even not FMC related;

The raw outcomes of an OpenFlow controller are Flow Table Entries (FTE)

to be installed at switches, and network packets to be forwarded by switches, both generated in response to network events and/or in response to high level control operations. To accomplish such tasks providing the aforementioned features, efficient models and abstractions must be provided. In particular, controller design has to provide both a data model to describe the network and its state, and a control logic programming model to interact with such data model. The design phase has to address also the so called non-functional requirements, e.g., performance and scalability of the implemented system. Using again the general purpose computer metaphor, in a general purpose computer the system behavior depends both on the hardware architecture and on the software running on top of such architecture, likewise in OpenFlow, it depends both on the network architecture and on the control logic implemented by the controller. Understanding the effects brought by different ways of interacting with the OpenFlow network is an important step to drive the design decisions. Hence, during the development of the FMC OpenFlow controller, we performed a preliminary analysis of the different dimensions that characterize an OpenFlow controller. We refer to this dimensions as “*Control Logic Dimensions*”.

### Control logic dimensions

In OpenFlow the actual control plane behavior is defined by the Controller. Given the flexibility of the OpenFlow approach, we can try to perform an high level classification of the characteristics of different control logics, identifying coarse-grained dimensions to classify such applications, the same way computers' programs are classified as CPU-bound, I/O bound and so on.

- **Flows Granularity:** defines the granularity of the network flows managed by the control logic. The granularity is defined after the header fields of current data packets. For example, a flow identified by the solely destination MAC address is coarse grained, while a flow identified by the combination of IP addresses and port numbers is much more fine grained;



- **Network Visibility:** an OF application may need detailed network traffic information or links statuses, for, e.g., load balancing or route reconfiguration. Depending on the control logic, the quantity and frequency of switches status update may vary;
- **Network state:** network state is related to the information the control logic has to manage, in order to provide its functions. Typical examples of network state are routing tables, end-points identity information, etc.;
- **Reactivity:** OpenFlow provides a mean to reactively program switches, through the forwarding of network packets to the controller. A control plane can range from being fully reactive, when each OF table entry is installed in response to a packet coming to the controller, to proactive, when all entries are installed before network traffic arrives to the switch.

In designing the FMC controller, some dimensions were dictated by the mobility technique adopted in FMC, e.g., the flows granularity. The other dimensions may vary according to the taken implementation decisions. In particular, in the FMC implementation, we decided that we can tolerate an increased network state to maintain at controllers, instead of increasing the number of OpenFlow messages exchanged to retrieve the switches' status. At the same time, we adopted a control logic as much proactive as possible, i.e., pre-installing FTEs when it is possible.

### **Hierarchical control**

FMC architecture suggests a hierarchical controller organization. In FMC the hierarchy is mainly related to the geographic locations, and, in particular, there are two hierarchical levels. A (i) local level, related to the handling of the Learning Switch functions and low level FMC mobility technique operations, and a (ii) global level, that provides a global view of the network

and that coordinates the local levels to provide FMC functions on a geographic scale. The two levels differ for both the performed operations and the processed data. The local level handles OFSes directly, providing FTEs and handling network events. The global level task is to coordinate the local levels to provide the required network functions, i.e., in the FMC case the network addresses mobility. Hence, the global level is also the place in which the controller north-bound interface is implemented. The north-bound interface, in the FMC example, provides methods to, e.g., define identifier/locator mappings or to request a node migration<sup>5</sup>.

### Data model

The controller is built around an easily accessible view of the switches, so that advanced functions can be defined using a common network model. We defined the network model using an object-oriented (OO in short) approach. The decision is motivated by the suitability of the OO paradigm for the description of network devices like OF-switches, and by the deep understanding of the OO model by programmers, that are highly involved in the design and management of a Software Defined Network.

The object-oriented network model is composed of the following base classes:

- *Network*: contains a globally unique identifier and a set of OFSwitch objects. It works mainly as a container for OFSwitch objects and for network state that is related to the whole network, i.e., it is part of the global level of the hierarchy;
- *OFSwitch*: is the base class used to represent and manage an OpenFlow switch. It includes both the state of the switch, that can be used programmatically by controller functions, and a set of methods used to handle network events.

---

<sup>5</sup>In the current implementation the north-bound interface is implemented as a REST interface that exchanges JSON serialized data.

The network model is dynamic, OFSwitch objects are added or removed in response to the connections initiated by corresponding switch devices. The model is dynamic also in the sense that an OFSwitch object contains information about the switch current internal state, such as the installed FTEs. The implementation of the described classes can be tuned according to the desired impact on the different Control Logic Dimensions. For example, the OFSwitch can be designed to cache FTEs installed at the represented switch, but can also dynamically retrieve the FTEs from the switch, i.e., sending OpenFlow messages.

### Scalability

To provide scalability, the controller uses several *Controller nodes* to execute the network control logic. At this aim, the network model is extended to include in each OFSwitch class, in addition to switch related network state, also the related control logic. The control logic implemented in an OFSwitch object can operate only on the switch represented by the object itself. The network control logic, hence, belongs to the OFSwitch instances and not to the controller as a whole. Since each OFSwitch instance contains also all the data related to the represented switch (as explained in sec. 6.2.3), each instance can be moved among controller nodes when needed. Each controller node contains at least a Network object, that is in charge of dispatching network events it receives from controlled switches: each event is forwarded to the corresponding OFSwitch object, that executes proper algorithms to handle it (See Figure 6.7). Using this programming model, distributing the controller application becomes a problem of partitioning the OFSwitch objects among different controller nodes. The controller nodes are assumed to be placed in locations that are near the switches they are controlling (from the network perspective), in order to reduce the delay of controller-switch communication. The current FMC controller implementation takes care also of the distribution of messages destined to OFSwitch instances, that are used to develop control algorithms that involve more than one switch, i.e., algo-

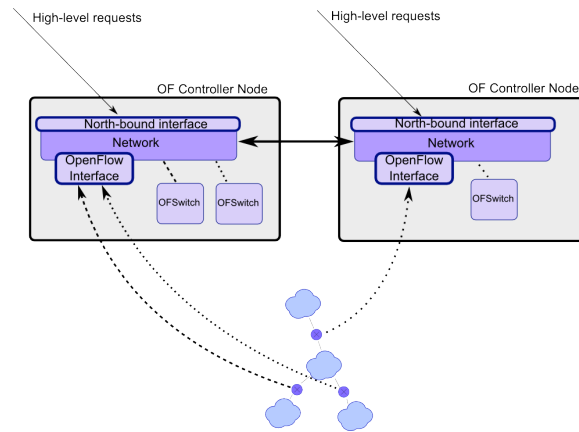


Figure 6.7: FMC controller architecture and deployment example: network OFSes are distributed among two OF Controller Nodes. OFSes are managed through OFSwitch objects. The Network object dispatches events coming from switches to the correspondent OFSwitch objects, and implements transparent communication among OFSwitch objects hosted at different controller nodes.

rithms belonging to the global level of the hierarchy. Such communications are handled in a way that is transparent to the programmer, using a “proxy” object in case the OFSwitch object is located on a different OFC node. The model provides the programmer with a clear separation of the control decisions that could be taken at the single switch level (local level), from the ones that need a broader view of the network (global level).

## Extensibility

Extensibility is provided using OO paradigm characteristics, like inheritance (See Figure 6.8.a). The OFSwitch class can be extended to provide new or enhanced functions, e.g, by overriding and extending methods. For example, it is possible to provide a LearningSwitch implementation, that resembles classical L2 switches functions, and then, extending this class, other functions can be added (e.g., the FMCSwitch class implements the FMC functions). The main issue in providing extendability through an OO model is the paradigm mismatch between OO programming and OF switches programming, since the switches programming is performed by means of FTEs. Because of this, the OO paradigm is not used to program the network itself, but to interact

with the network devices in a (hopefully) simplified way.

The final outcomes of the execution of methods from OFSwitch class and derivatives are a set of FTEs to be installed on the switch, and a set of packets to be sent by the switch. The extended classes and methods, hence, are in charge of providing a set of FTEs and network packets modified accordingly to the desired result. This process is tricky: the addition of a FTE can have unexpected effects on the behavior of the switch, that is defined by the combination of all the FTEs installed on that switch. Moreover, some packets coming from a super-class's methods may not be sent anymore in the extended function, and so on. We are actually handling a two levels programming problem:

1. high-level programming is performed by using API provided by OFSwitch classes and derivatives. Programmers are in charge of defining convenient APIs to allow the extensions of the functions they are providing in a given class;
2. low-level programming is performed by means of FTEs and packets sent through the switch. All the high-level functions are finally translated in FTEs and packets to be sent.

Our purpose is to provide extendability in any case, so, also when the developer of an OFSwitch sub-class is not providing methods to easily modify its application behavior before the FTEs and network packets are generated.

To this end, the OFSwitch class provides convenient methods to perform network events handling. For example, when a packet is forwarded from a switch to the controller, a *packet\_in* network event is generated. The event is handled by a specific method in the OFSwitch class, that implements the control logic to handle the packet, and provides (i) an ordered list of FTEs to install at switches, and (ii) an ordered list of network packets that must be sent by the selected switches. This approach allows for the extension of the OFSwitch class: a subclass that inherits from the OFSwitch can still use the methods from the superclass, to get the lists of FTEs and packets to send,

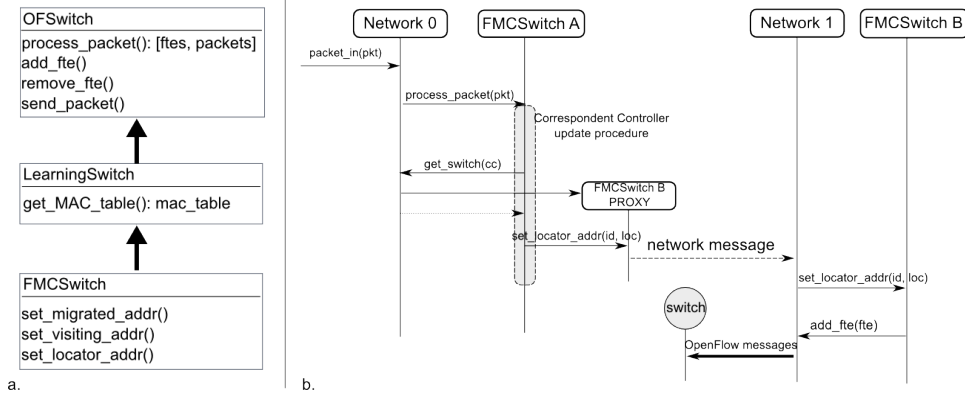


Figure 6.8: a. FMC controller object model: the basic `OFSwitch` class is extended to introduce `LearningSwitch` functions and, then, it is further extended to introduce FMC functions. b. FMC controller operations sequence diagram, when an Home Switch receives a packet from a not updated correspondent network: the `Network` object contains information about the location of the correspondent node, hence, the related CC can be updated with the *identifier/locator* mapping information.

and adjust them according to the extended control logic.

In addition to this feature, the `OFSwitch` class provides a dedicated method to install rules on the corresponding OF-switch, in order to intercept all the FTE installation requests (and network packets sending requests) before actually issuing them at the corresponding switch. Using this approach, it is possible to introduce some extended logic that, before the actual switch programming happens.

## 6.2.4 Discussion

In this section we discuss how the design we made helped us in developing the FMC distributed controller. As stated in section 6.2.3, FMC is well suited to use a hierarchical organization of the control logic, with a good separation between local and global operations. The Object Oriented model used to describe switches and networks is particularly useful at this aim. The `Network` class, working as a container for a set of `OFSwitch` objects, is a good place to implement the global control logic, while the `OFSwitch` objects, being closely related to the network device, are in charge of handling the local control logic. If necessary it could be possible also to provide more

intermediate levels in the hierarchy, but for our purposes it was not the case.

Local control logic includes also the handling of local network events, generated by switches, that are actually handled at corresponding OFSwitch objects. This approach helped in the distributed controller implementation, since all the local events are kept local, requiring no interactions among different Controller nodes. The OFSwitch class (and its sub-classes), in this context is used both as an aggregation and filtering point for local network events, that then are passed up in the hierarchy shaped as “high-level events”. The implementation of such filtering and aggregation functions, i.e., the shape of “high-level events”, is tightly coupled to the required global control logic. E.g., in figure 6.8.b the event linked to the reception, at the home network, of a packet destined to a migrated MN is handled locally by the HC and then translated into a high-level event, that is implemented through the call of the *set\_locator\_addr()* method on the (proxy) of the FMCSwitch object representing the CC. Using OO inheritance, The OFSwitch class can be extended by a sub-class that implements the required filtering and aggregation logic, then, the sub-class is associated with the switches that require the application of such logic. The final outcome is the association of different OFSwitch classes to different switches, according to the control logic we have to implement at such switches, in an elegant and easy way.

The combination of OFSwitch sub-classes that expose an interface and network events related to the global control logic, with the presence of global-level Network objects made the implementation of the north-bound interface functions straight forward. The Network object is a good place to expose the north-bound interface, while the mapping of this interface to the low level FTEs is made easy using OFSwitch sub-classes, that translate a set of high-level methods in corresponding FTEs. Hence, the north-bound interface interacts with OFSwitch sub-classes instead of having to take care of low-level FTEs, separating the development of low level FTEs programming from the development of high-level network functions.

While OO paradigm is really useful in providing separation of concerns

and scalability, it does not help in providing extensibility, that still requires a direct handling of FTEs, i.e., the function that is going to be extended must be well known by the programmer, to correctly handle the provided FTEs. Our design helped the extensibility by providing a mean to manipulate FTEs before they are installed at switches. In particular, the abilities to intercept FTEs that are going to be installed and network packets that are going to be sent are crucial to this end. Moreover, the use of FTEs caching at OF-Switch objects made the analysis of the actual switches behavior much easier, simplifying both the network functions extension and controller debugging.

### 6.2.5 Related Work

The design of an OpenFlow controller, like the design of any software application, requires the application of languages, methodologies/abstractions and tools. In this section we provide an overview of available network languages, models and abstractions as implemented by OFCs, and tools to aid OpenFlow controller development.

#### Programming languages

Frenetic [49] is a high-level language based on the functional programming paradigm, that provides the programmer with an omniscient, centralized view of the network. A run-time system, linked to the language, “translates” the high-level instructions to a set of low-level packet processing rules, and manages them interacting with network equipment. NetCore [95] is an evolution of Frenetic, that extends the high-level language and provides some improvement in the compilation algorithms and run-time system, trying to speed up the network performance.

#### Programming Frameworks

Hyperflow[142] provides a Distributed OpenFlow Controller Framework that separates the network in partitions. Each partition is assigned to a controller instance and all the instances are synchronized by means of a pub-



lish/subscribe mechanism. In this approach each controller instance runs the same control logic, while the framework distributes the network events and OpenFlow messages to the appropriate controller instance. Hence, HyperFlow provides transparent scalability of the Controller. To provide its features, Hyperflow distributes consistently the network state updates and OpenFlow messages. Onix[72] defines a Network Information Base (NIB) that is read and written by the control logic. The NIB is actually distributed using different strategies defined by the developer, that chooses the storage system type for the NIB data, according to their needs in terms of speed, consistency and reliability. All the synchronizations and operations on the physical network are performed through the NIB, so, by partitioning the NIB among several controller instances, it is possible to share the overall load and distribute responsibilities. A third approach is based on a service view of the Controller: the control logic is implemented as a collection of services that collaborate among them. Each service is able to run on a separate controller instance, hence, several instances can be used to share the workload. Clearly this approach requires a careful design of the control logic, whose implementation as a collection of services, and the way such services are distributed, dictates the actual shape of the system[135].

## Tools

To test the correctness of OpenFlow applications, NICE was proposed in [30]. NICE is a tool for automatic OpenFlow applications testing, that combines model checking and concolic execution to explore the state space of OpenFlow programs written for the NOX controller platform. In FlowChecker [3] the aim is to detect OFS misconfigurations. FlowChecker uses manually built binary decision diagrams to encode OpenFlow rules and then applies model checking in order to detect OpenFlow switches misconfigurations.

### 6.2.6 Conclusions

Software Defined Networking is a promising paradigm for future network management, and OpenFlow is emerging as a successful industry-supported SDN building block. In this paper we presented the design decisions and the experience we made in developing a distributed OpenFlow controller for supporting the Follow-Me Cloud technology. We introduced a hierarchical view of the control operations, as well as a network model based on the Object Oriented paradigm. Furthermore we introduced a first coarse-grained classification of OpenFlow controller behavior in respect to a few parameters, and explained how the OO paradigm can be exploited to support both scalability and extendability of the OpenFlow controller, taking into account the paradigm mismatch among OO model and the OpenFlow programming model.

## 6.3 Handover handling

### 6.3.1 Introduction

In this section we present the details of the FMC handover procedure. We perform an analytical analysis of the FMC handover operations compared to other mobility technologies and identify key criticalities and issues.

### 6.3.2 Related Work

Adding mobility support to TCP/IP networks is a widely addressed research problem. MobileIP (MIP), available for both IPv4 (MIPv4) [113] and IPv6 (MIPv6) [115] enables end-points mobility by introducing a network stack extension for mobility in the end-points and adding two network entities: the home agent (HA) and the foreign agent (FA). A mobile node (MN) that changes its network attachment point, in addition to its original IP address (called the *home address*), acquires a new address, the *Care-of Address* (CoA), and registers this address to the HA through a *binding update* (BU) message, that actually binds the *home address* with the CoA. The HA builds

a tunnel to the CoA, so that any packet sent by a *Correspondent Node* (CN) and destined to the *home address* is tunneled at HA and forwarded to the CoA. The FA is used in MN network change detection, moreover, the CoA can actually be the FA's address. The HA, in this case, builds a tunnel to the FA, that in turn decapsulates packets and send them to the MN. Mobile IP introduces some inefficiencies in network routing, since it realizes the so-called *triangular routing*, i.e., packets follow a suboptimal route when the CN sends them, since they have to pass through the HA, while they follow a direct route in the MN-CN direction. To handle such issue, Mobile IP may use a route optimizations scheme, that sends a BU also to CNs, so that a CN can switch the *home address* with the CoA before actually sending a packet to the MN, without involving the HA anymore. In order to reduce the connectivity downtime for MN during network changes, Fast-handover MIP (FMIP) [71] has been developed. FMIP reduces handover time by pre-configuring the involved network entities, e.g., by acquiring the CoA in advance, before the network change happens. Hierarchical MIP (HMIP) [137] further extends MIP to reduce the required signalling, by introducing a hierarchy in MIP. A new network entity, the *mobility anchor point* (MAP), manages MN mobility in a local domain. Each MAP is associated with a local domain, that comprises several networks. In HMIP, a MN is actually associated with two CoA, the Regional CoA (RCoA), that identifies the current MN's MAP and the Local CoA (LCoA), that locates the MN in the local domain. When the MN changes a network inside a local domain, it just sends a BU message to the MAP to update the LCoA. When the MN changes the local domain, it first acquires the RCoA and LCoA, then it sends a BU with the LCoA to the MAP and a BU with the RCoA to the HA. The final outcome of the hierarchy is a reduced signalling traffic when the MN moves inside a local domain. It is also possible to combine FMIP and HMIP to further optimize MIP. Proxy MIP (PMIP)[56] avoid the direct involvement of the MN in local mobility operations, i.e., to allow transparent MN migrations among networks in the same local domain. PMIP uses a network entity called *Mo-*

*bility Access Gateway* (MAG) that performs MIP interactions on behalf of the MN, hence, the MN is unaware of the CoA, that is instead acquired by the MAG. In case the MN migrates over local domain boundaries, it has to be still involved in mobility handling operations.

MIP supports end-points mobility by providing identifier/locator concepts split, i.e., the *home address* works as identifier (ID) for the MN while the CoA works as locator (LOC). Other approaches provide either end-points mobility or ID/LOC split. *Locator/Identifier Separation Protocol* [46] (LISP) provides separation among *locator* and *identifier* addresses, using an “alternate topology” to distribute ID/LOC mapping information among routers in the network. LISP does not provide mobility support natively, while it is added in its “mobility extensions” that need direct involvement of mobile devices in the ID/LOC mapping. The *Host Identity Protocol* (HIP) [96] provides a similar splitting of ID and LOC concepts, but instead of reusing the already defined IP address space, it defines a new address space for identifiers, based on cryptographic (asymmetric) keys. Also in the HIP case mobility is provided as an extension to the base architecture. *Interactive Protocol for Mobile Networks* (IPMN) [153] provides mobility support without defining identifier addresses, changing transparently MN’s address on-the-fly at nodes involved in the network communications. IPMN uses a paradigm called Interactive Transparent Networking to hook into the TCP stack of network end-points, in order to define a new TCP connection message that informs the correspondent node about address changes, and to transparently substitute IP addresses used by TCP into operating system’s network sockets.

### 6.3.3 Follow-Me Cloud

FMC services are implemented through a distributed architecture of network entities called Controllers. To manage ID/LOC mapping information and to distribute it among networks, FMC uses the architecture depicted in Figure 6.2.b. When a MN changes its network attachment point, an external entity

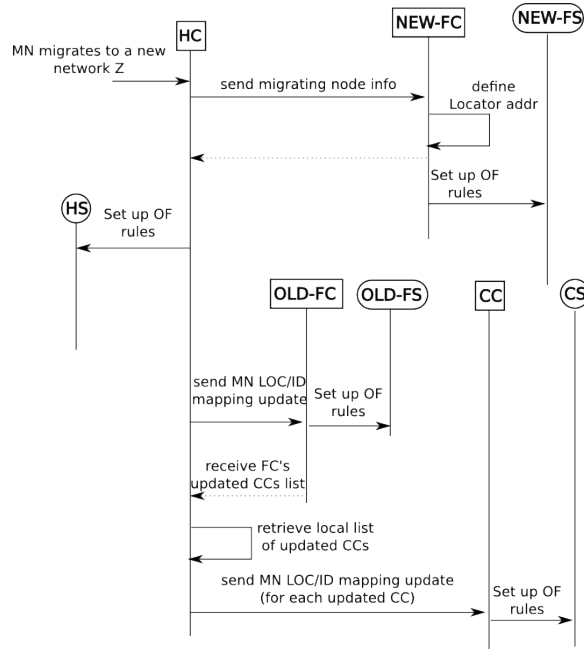


Figure 6.9: FMC Handover Procedure

detect the movement<sup>6</sup> and triggers the HC to start an handover procedure (Fig. 6.9). HC is provided with information about current MN's IP address, i.e., the ID, and on which network MN is moving to (we call such a network nFN). HC sends a message to the FC in charge of managing nFN (the nFC), in order to obtain a LOC address to create a new ID/LOC mapping. nFC sends the generated LOC address back to HC, so that both controllers can configure the OpenFlow Switch (OFS) located at respective networks. If the MN was already migrated to a FN (the oFN), the controller in charge of managing such network (the oFC) is informed by HC of the new migration. oFC answers HC sending a list of the CCs that requires updated ID/LOC mapping information as well. In the last step, HC informs CCs, that in turn set up OFSes located at their networks.

<sup>6</sup>The definition of such a service is out of the scope of this paper. It could be, e.g., a network stack's layer 2 service

Table 6.1: Delay Variables

Variable	Description	Expression	Default Val. (ms)
$T_L$	<i>Local delay</i>		1
$T_{HN-oFN}$	$HN < - > oFN$		10
$T_{HN-nFN}$	$HN < - > nFN$		15
$T_{HN-CNet}$	$HN < - > CNet$		20
$T_{nFN-CNet}$	$nFN < - > CNet$		20
$T_{HC-oFC}$	$HC < - > oFC$	$T_{HN-oFN}$	10
$T_{HC-nFC}$	$HC < - > nFC$	$T_{HN-nFN}$	15
$T_{HC-CC}$	$HC < - > CC$	$T_{HN-CNet}$	20
$T_{MN-HA}$	$MN < - > HA$	$T_L + T_{HN-nFN}$	16
$T_{MN-CN}$	$MN < - > CN$	$2T_L + T_{nFN-CNet}$	22
$T_{MN-MAP}$	$MN < - > MAP$	$2T_L$	2

### 6.3.4 Handover analysis

To analyze handover performance and compare it with other mobility technologies, we considered the case of a Mobile Node (MN) moving from a Foreign Network (called oFN) to a new FN (called nFN). Our evaluation assumes that the time to exchange messages among different entities is the characterizing factor for handover performance. It is worth to note that we are not taking into account other factors, e.g., movement detection time or IP addresses acquisition time, since we consider such times not being part of the core mobility technique, i.e., some approaches can be applied to different technologies, e.g., FMIP concepts can be easily ported to FMC. Since the mobility technologies presented in section 6.3.2 behaves differently in local and global mobility cases, we perform our analytical evaluation separating the global case from the local one. Fig. 6.10 presents a simplified view of the network that includes the entities adopted by considered technologies to provide mobility, and introduces variables names used to identify the network delays among the different entities. Network delays experienced by messages exchanged in the network, for the considered technologies, can be obtained summing the presented variables, as explained in table 6.1. Our evaluation has been performed considering 5 different scenarios. In each scenario a single variable changes its value in a given range, as shown in table 6.2, while the other variables remains fixed to the default values presented in table 6.1.

We compared FMC to MIP, HMIP and IPMN. MIP and HMIP are taken into account being standard technologies to provide mobility that do not require significant modifications to the current network architecture (e.g., LISP

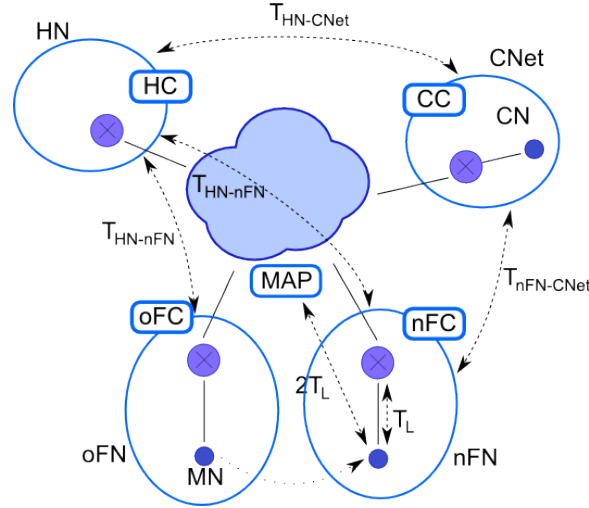


Figure 6.10: Graphical representation of delay variables presented in Table 6.1

Table 6.2: Analysis Scenarios

	Local	HN-oFN	HN-nFN	HN-CNet	nFN-CNet
Range	1-10	10-55	15-60	20-65	20-65
Scenario #	0	1	2	3	4

would require an “alternate topology”), while IPMN was selected since it is a technology that provides mobility without requiring intermediate network entities. We computed handover times summing the delays of the messages sequence required by each technology to perform the process (See table 6.3). The results of our performance comparison are presented in figure 6.11. The graphs show that technologies are not homogeneous in the way they perform in respect to changing network delays. FMC is less influenced by local delays ( $T_L$ ), since only controllers, that are placed at the edge of access networks, are involved into the handover, but it shows poor performance when the delay between HN and oFN, or the delay between HN and CNet increases. In scenario 2 it behaves like MIP and HMIP (actually it is slightly faster), while it is unaffected by the increasing of nFN-CNet delay.

For the local mobility case, we compared FMC only against HMIP, since other technologies does not provide any enhancement for this specific case. FMC architecture is flexible enough to allow a Controller to play more roles,

Table 6.3: Handover delays

Solution	Formula
FMC	$2T_{HC-nFC} + 2T_{HC-oFC} + T_{HC-CC} + T_L$
MIP	$2T_{MN-HA} + 2T_{MN-CN}$
HMIP (local)	$2T_{MN-MAP}$
HMIP (global)	$2T_{MN-MAP} + 2T_{MN-HA} + 2T_{MN-CN}$
IPMN	$3T_{MN-CN}$

Table 6.4: FMC configurations

#	Controller Joint Networks	Formula
a	HN, oFN	$2T_{HC-nFC} + T_{HC-CC} + T_L$
b	HN, nFN	$2T_{HC-oFC} + T_{HC-CC} + T_L$
c	HN, CNet	$2T_{HC-nFC} + 2T_{HC-oFC} + T_L$
d	HN, oFN, nFN	$T_{HC-CC} + T_L$
e	HN, oFN, CNet	$2T_{HC-nFC} + T_L$
f	HN, nFN, CNet	$2T_{HC-oFC} + T_L$
g	All	$T_L$

hence, we assume that, in a local mobility case, the same Controller could be in charge of managing more than one network. In FMC we will call "local domain" the set of networks controlled by the same Controller. We considered the configurations presented in table 6.4. Each configuration involves the HC, since the current FMC architecture mandates that the HC has to orchestrate the handover. Because of this, with the exception of FMC-g, all the FMC configurations require at least a message that passes the local domain boundaries, hence HMIP usually performs better, since it requires message exchanges only among MAP and MN (See figure 6.12). The sole case in which FMC can compete with HMIP is when all the involved networks are under the management of a single Controller, i.e., when all the involved end-points are in same local domain.

### 6.3.5 Discussion and Conclusions

FMC is a novel technology that, among other features not discussed in this paper, adds support to mobility and ID/LOC concepts splitting in TCP/IP



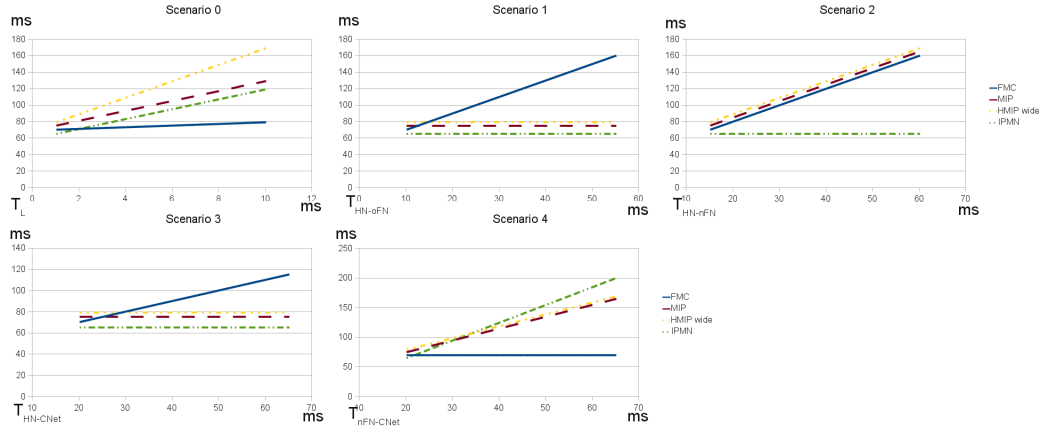


Figure 6.11: Handover delays evaluation for global mobility

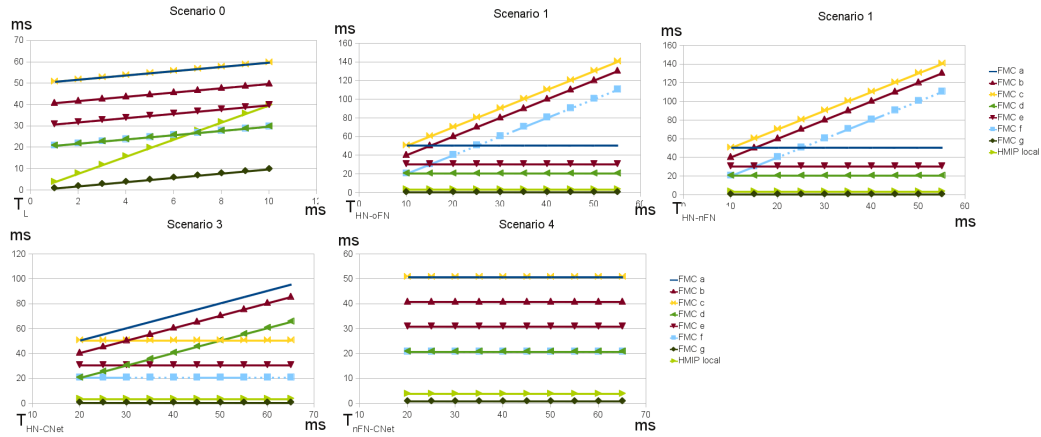


Figure 6.12: Handover delays evaluation for local mobility

networks, requiring no involvement of end-points both in the global and local cases. Other technologies we cited in this paper mandates end-points modifications to provide such a feature. Since MN is not aware of its movement, FMC requires that the HC orchestrates the movement on behalf of MN. This architectural choice provides a different behavior in handover performance in respect to MIP technologies. FMC uses the HC as the entity in charge of managing the handover, in MIP and HMIP the handover is managed through the help of the HA, while in IPMN the MN manages the handover on its own. In particular, FMC performs better than any other technology when (i) the local delay is big, since it does not need to directly exchange signalling traffic

with MN. and (ii) when MN is moving to networks that are far from CNetS and the HC to CNetS delay remains unchanged. On the contrary, FMC performance is poor when a MN is moving far from its HN, e.g., a mobile user is travelling, traversing several FNs. In this case, a technology that reduces or does not require signalling messages with the HN, i.e., IPMN, outperforms FMC. Nevertheless, handover procedure optimization strategies can be explored to improve performance, e.g., dynamically delegating some handover operations to oFC or nFC in order to reduce signalling messages delay.

An advantage of FMC is the ability to support local mobility optimizations in a straightforward manner: FMC does not require the definition of new network entities like in HMIP, since each Controller can play more than one role in the architecture. In this case, all the signalling messages exchanged among the roles played by the same Controller are avoided, speeding up the process. Anyway, HMIP still performs better since it avoids non-local signalling by using multiple IP addresses. Hence, the complexity of HMIP in managing more IP addresses is traded with better handover performance in local domains. In one case FMC shows better performance than HMIP, when all the roles are played by a single Controller. This property is useful when FMC is applied to closed or small networks, e.g., in a datacenter network to support Virtual Machines migrations.

In conclusion, FMC provides several features that other technologies provide in a separately, e.g., end-point transparency, ID/LOC splitting, local mobility optimizations, avoiding, at the same time, the use of tunneling to reduce the overhead caused by encapsulation. Moreover, the FMC handover process still shows performance that in many cases are comparable or better than the one experienced by other technologies. Nevertheless, there are rooms for optimizations and enhancements, both in the global and local mobility cases, that should be taken into account in FMC future developments.

# Conclusions

The big number of different applications supported by the Internet makes the design of a proper network architecture an hard task. The heterogeneity and the fast evolution of network-related technologies require an architecture that supports flexibility. Nevertheless, the Internet was born several decades ago, when many requirements were unforeseen: even if the success of the original design enabled the fast growth of the network, at the same time it has been unable to support evolution, making the Internet an “ossified” technology.

In this Ph.D. thesis we highlighted an evolution in the Internet communication model, from a process-oriented to a service-oriented communication model. To this end we shown how applications can be provided using a service-oriented model and how they, at the same time, require and enable the use of flexible virtualized infrastructures. Based on this use case, we solved several issues in the provisioning and management of such infrastructures, both in local and geographical contexts. Finally, using a software-defined networking approach, we implemented and evaluated an architecture for separating the network identifier and locator concepts in order to provide flexible network identifiers that can be applied both to nodes and in particular to services, decoupling routing in the network from identification of the communicating entities, and enabling the implementation of a service-oriented communication model. In particular, we provided the following contributions:

- We implemented two different applications following an “as a service” model: (i) a testbed as a service system to execute automatic networking experiments in network testbeds provided on demand; (ii) a GRID

as a Service system to create and manage either GRID resources or completely new GRID sites on demand. The two use cases show how the network, using a service model, can be used to access even complex infrastructures composed of several network devices. We designed and implemented both use cases, defining the usage model and the required abstractions.

- We defined methodologies to create and manage virtual infrastructures composed of virtual links and virtual nodes. Three different contributions were provided in this field: (i) we designed a system for providing isolated virtual links on top of a shared medium, supporting Quality of Service for each virtual link, as well as emulated link properties (e.g., delay, loss rate), in order to enable experimentation of network applications and protocols under different network conditions; (ii) we adapted the Lin-Kernighan heuristic to the solution of a *network testbed mapping* problem, we implemented it and evaluated our implementation, in order to assign virtual nodes and virtual links to resources belonging to the physical substrate; (iii) we designed a system for the fast provisioning of virtual nodes using a copy-on-write method, which takes advantage of the characteristics of the particular application, showing how sensible improvements can be obtained when the knowledge of the specific application is exploited.
- In order to be able to support mobility in a traditional IP network, we designed and implemented two different methods, one that makes use of a centralized controller entity, suitable for datacenters and localized networks, and one that uses distributed entities exploiting a model similar to the Mobile IP one, targeted at geographical deployments. We applied both models to the case of virtual infrastructures mobility.
- Flexible virtualized infrastructure raise new challenges when managing resources access and security. We provided a first study on the applicability of traditional techniques for network intrusion detection to

highly virtualized environments like cloud datacenters, evaluating different architectural approaches. Moreover, we present a solution for the management of the access to resources geographically distributed and belonging to different authorities.

- To enable an higher degree of flexibility in the network architecture, we explored software-defined networking solutions, providing a first coarse-grained mean to characterize the behavior of the control logic in a software-defined network, highlighting the effect on the scalability of the system and introducing viable architectural solutions. Moreover, we provided an algorithm and a tool to ease the programming of the network, by automatically detecting interacting control actions operated on network devices.
- Finally we designed, implemented and evaluated an architecture for the split of the network identifier and locator concepts. In addition, we provided a control plane scalability analysis, a programming methodology for the network control plane, and a comparison of the implemented technique in regard to the handover performance in case of identifier/locator mapping change (e.g., in case of a network node migration).

The provided contributions show a clear path towards a new way of handling network protocols and architectures: one of the aim of this work is to provide a practical example of a design that, using the same network protocols already deployed, is able to overload the meaning of the protocol fields to provide a new semantic and to introduce a new communication model. Chapter 6 accomplishes this purpose by using minor modifications in the current network infrastructure, in order to overload of the IP address meaning, enabling, this way, a new communication paradigm in which the IP address could be used as an immutable identifier, regardless of the position or of the entity to which it is actually bond to. For instance, an IP address, when working as an identifier, could be associated to a “service entity“, then, us-

ing the architecture described in chapter 6, a dynamic binding to a proper service implementation could be defined to support, e.g., dynamic service selection or load balancing.

The presented architecture, together with the introduced studies, is a first step towards the implementation of a service oriented networking. Future work include the assessment of the impact of such an architecture on already deployed protocols, such as DNS, but also the development of applications targeted specifically to the new communication paradigm, to highlight the benefits. Moreover, in this thesis we introduced the concept of network identifier looking at the sole IP address. We believe that extending the concept including other fields, such as the TCP/UDP port number, requires a little effort, but could provide even more benefits (e.g., it could partially solve the IPv4 addresses exhaustion). Furthermore, other effects of such an architecture on more complex network dynamics, like routing table dimensions in the core networks, are worth to be explored. For instance, we believe that the provided separation of identifiers from locators could enable little or no fragmentation of routes in routing tables, reducing both devices and their management costs.

This thesis also contributed to the definition of flexible network infrastructures (see chapters 2, 4, 5), nevertheless, the path towards a flexible network architecture is still long. In this thesis we presented the promising concept of software-defined networking (chapter 5), but we also pointed out some of the main concerns regarding its scalability. To be completely successful, SDN still requires several research efforts: moving the concept of network controller towards a concept of network operating system is, among the others, one of the most interesting future works in this context. A network operating system could enable the same fast evolution we had for general purpose computer also for networks. The creation of programming/runtime tools like the one presented in this thesis is actually pushing into this direction.

# Bibliography

- [1] Vinay Aggarwal, Anja Feldmann, and Christian Scheideler. Can ISPs and P2P systems co-operate for improved performance? *ACM SIGCOMM Computer Communications Review (CCR)*, 37(3):29–40, July 2007.
- [2] I.F. Akyildiz, X. Wang, and W. Wang. Wireless mesh networks: a survey. *Computer Networks*, 47(4):445–487, 2005.
- [3] Ehab Al-Shaer and Saeed Al-Haj. Flowchecker: configuration analysis and verification of federated openflow infrastructures. In *Proceedings of the 3rd ACM workshop on Assurable and usable security configuration*, SafeConfig '10, pages 37–44, New York, NY, USA, 2010. ACM.
- [4] E.S. Al-Shaer and H.H. Hamed. Modeling and management of firewall policies. *Network and Service Management, IEEE Transactions on*, 1(1):2 –10, april 2004.
- [5] Amazon.com Inc. Amazon.com reports ddos attacks on wikileaks. <http://aws.amazon.com/message/65348/>.
- [6] Amazon.com Inc. EC2 web site. <http://aws.amazon.com/ec2/>.
- [7] Amazon.com Inc. EC2 web site. <http://aws.amazon.com/ec2/>.
- [8] Amazon.com Inc. High Performance Computing (HPC) on AWS. <http://aws.amazon.com/hpc-applications/>.
- [9] A. Anadiotis, A. Apostolaras, D. Syrivelis, T. Korakis, L. Tassiulas, L. Rodriguez, and M. Ott. A new slicing scheme for efficient use of wireless testbeds. In *Proceedings of the 4th ACM International Workshop on Experimental Evaluation and Characterization*, WINTECH '09, pages 83–84, 2009.

- [10] A. Anadiotis, A. Apostolaras, D. Syrivelis, T. Korakis, L. Tassiulas, L. Rodriguez, I. Seskar, and M. Ott. Towards maximizing wireless testbed utilization using spectrum slicing. In *Proceedings of Trident-Com 2010*, Berlin (Germany), May 2010.
- [11] T. Anderson, L. Peterson, S. Shenker, and J. Turner. Overcoming the internet impasse through virtualization. *Computer*, 38(4):34 – 41, april 2005.
- [12] M. Andic, Dejan, Ignacio M. Llorente, and Ruben S. Montero. Opennebula: A cloud management tool. *Internet Computing, IEEE*, 15(2):11 –14, march-april 2011.
- [13] asterisk.org. Asterisk open source telephony project. <http://www.asterisk.org/>.
- [14] P. Barham, B. Dragovic, K. Fraser, S. Hand, A. Harris, T. and Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. volume 37, pages 164–177. ACM New York, NY, USA, 2003.
- [15] G.B. Barone, R. Bifulco, V. Boccia, D. Bottalico, and L. Carracciuolo. Toward a flexible, environmentally conscious, on demand high performance computing service. In *Data Compression, Communications and Processing (CCP), 2011 First International Conference on*, pages 136 –138, june 2011.
- [16] Andy Bavier, Nick Feamster, Mark Huang, Larry Peterson, and Jennifer Rexford. In vini veritas: realistic and controlled network experimentation. *SIGCOMM Comput. Commun. Rev.*, 36(4):3–14, 2006.
- [17] Andy Bavier, Nick Feamster, Mark Huang, Larry Peterson, and Jennifer Rexford. In VINI veritas: Realistic and controlled network experimentation. In *Proceedings of ACM SIGCOMM 2006*, Pisa (Italy), September 2006.
- [18] Sapan Bhatia. VSys: A Privilege Allocation Tool. Technical report, Princeton university, September 2008.
- [19] Sapan Bhatia, Murtaza Motiwala, Wolfgang Mi $\frac{1}{2}$ hlbauer, Yogesh Mundada, Vytautas Valancius, Andy Bavier, Nick Feamster, Larry Peterson, and Jennifer Rexford. Trellis: A platform for building flexible, fast virtual networks on commodity hardware. In *Proceedings of ROADS 2008*, 2008.



- [20] A. Bianco, R. Birke, L. Giraudo, and M. Palacin. Openflow switching: Data plane performance. In *Communications (ICC), 2010 IEEE International Conference on*, pages 1–5, may 2010.
- [21] R. Bifulco, R. Canonico, G. Ventre, and V. Manetti. Transparent migration of virtual infrastructures in large datacenters for cloud computing. In *Computers and Communications (ISCC), 2011 IEEE Symposium on*, pages 179–184, 28 2011-july 1 2011.
- [22] Roberto Bifulco, Marcus Brunner, Roberto Canonico, Peer Haselmeyer, and Faisal Mir. Scalability of a mobile cloud management system. In *To appear in Proc. of SIGCOMM workshop on Mobile Cloud Computing (MCC-2012)*, 2012.
- [23] Bastian Blywis, Mesut Günes, Felix Juraschek, and Jochen Schiller. Trends, advances, and challenges in testbed-based wireless mesh network research. *ACM/Springer Mobile Networks and Applications (MONET)*, 15(3):315–329, June 2010.
- [24] Alessio Botta, Roberto Canonico, Giovanni Di Stasi, Antonio Pescapè, Giorgio Ventre, and Serge Fdida. Integration of 3G Connectivity in PlanetLab Europe. *ACM/Springer Mobile Networks and Applications (MONET)*, 15(3):344–355, 2010.
- [25] Alessio Botta, Alberto Dainotti, and Antonio Pescapé. Multi-protocol and multi-platform traffic generation and measurement. INFOCOM 2007 DEMO Session, Anchorage (Alaska, USA), May 2007.
- [26] T. Braun, V. Hilt, M. Hofmann, I. Rimac, M. Steiner, and M. Varvello. Service-centric networking. In *Communications Workshops (ICC), 2011 IEEE International Conference on*, pages 1–6, june 2011.
- [27] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. Advances in network simulation. *IEEE Computer*, 33(5):59–67, May 2000.
- [28] Anton Burtsev, Prashanth Radhakrishnan, Mike Hibler, and Jay Lepreau. Transparent checkpoints of closed distributed systems in emulab. In *EuroSys '09: Proceedings of the fourth ACM european conference on Computer systems*, pages 173–186, New York, NY, USA, 2009. ACM.
- [29] Zheng Cai, Alan L. Cox, and T. S. Eugene Ng. Maestro: Balancing fairness, latency and throughput in the openflow control plane. 2011.

- [30] Marco Canini, Daniele Venzano, Peter Peresini, Dejan Kostic, and Jennifer Rexford. A nice way to test openflow applications. *EPFL Technical Report*, October 2011.
- [31] Roberto Canonico, Pasquale Di Gennaro, Vittorio Manetti, and Giorgio Ventre. Virtualization techniques in network emulation systems. In Luc Bougüé,  $\frac{1}{2}$ , Martti Forsell, Jesper Larsson Traff, Achim Streit, Wolfgang Ziegler, Michael Alexander, and Stephen Childs, editors, *EuroPar Workshops*, volume 4854 of *Lecture Notes in Computer Science*, pages 144–153. Springer, 2007.
- [32] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: taking control of the enterprise. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '07, pages 1–12, New York, NY, USA, 2007. ACM.
- [33] Ludmila Cherkasova and Rob Gardner. Measuring cpu overhead for I/O processing in the xen virtual machine monitor. pages 387–390, 2005.
- [34] Stephen Childs, Brian Coghlan, and Jason McCandless. Dynamic virtual worker nodes in a production grid. In Geyong Min, Beniamino Di Martino, Laurence Yang, Minyi Guo, and Gudula Runger, editors, *Frontiers of High Performance Computing and Networking - ISPA 2006 Workshops*, volume 4331 of *Lecture Notes in Computer Science*, pages 417–426. Springer Berlin/Heidelberg, 2006.
- [35] N. M. Mosharaf Kabir Chowdhury and Raouf Boutaba. Network virtualization: state of the art and research challenges. *Comm. Mag.*, 47(7):20–26, July 2009.
- [36] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, July 2003.
- [37] Cisco Systems. *Cisco Data Center Infrastructure 2.5 Design Guide*, March 2010.
- [38] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd Symposium*

- on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 273–286. USENIX Association, 2005.
- [39] CloudFlu. CloudFlu - HPC cloud computing for OpenFOAM (R) users. <http://sourceforge.net/projects/cloudflu/>.
- [40] Pasquale Di Gennaro, Roberto Bifulco, and Roberto Canonico. Neptune project sources homepage. <http://code.google.com/p/neptune-network-emulator/>.
- [41] Pasquale Di Gennaro, Roberto Bifulco, Roberto Canonico, and Giorgio Ventre. Neptune: Network emulation for protocol tuning and evaluation. Poster presented at the 2nd ICST International Conference on Simulation Tools and Techniques (SIMUTOOLS09), Rome, March 2009.
- [42] Marcel Dischinger, Andreas Haeberlen, Ivan Beschastnikh, Krishna P. Gummadi, and Stefan Saroiu. Satellitelab: adding heterogeneity to planetary-scale network testbeds. *SIGCOMM Computer Communication Review*, 38(4):315–326, 2008.
- [43] Norbert Egi, Adam Greenhalgh, Mark Handley, Mickael Hoerd, Laurent Mathy, and Tim Schooley. Evaluating xen for router virtualization. In *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, pages 1256–1261, 2007.
- [44] Chip Elliott. GENI: Opening up new classes of experiments in global networking. *IEEE Internet Computing*, 14(1):39–42, 2010.
- [45] David Erickson, Glen Gibb, Brandon Heller, Jad Naous, David Underhill, Guido Appenzeller, Guru Parulkar, N McKeown, and Et Al. A demonstration of virtual machine mobility in an openflow network. *Proceedings of ACM SIGCOMM Demo*, page 513, 2008.
- [46] Dino Farinacci, Vince Fuller, Dave Meyer, and Darrel Lewis. Locator/ID separation protocol (LISP). Internet-Draft draft-ietf-lisp-13.txt, IETF Secretariat, June 2011.
- [47] Nick Feamster, Lixin Gao, and Jennifer Rexford. How to lease the internet in your spare time. *SIGCOMM Comput. Commun. Rev.*, 37(1):61–64, January 2007.
- [48] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, August 2001.

- [49] Nate Foster, Rob Harrison, Michael J. Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. Frenetic: a network programming language. *SIGPLAN Not.*, 46(9):279–291, September 2011.
- [50] Anastasius Gavras, Arto Karila, Serge Fdida, Martin May, and Martin Potts. Future Internet research and experimentation: the FIRE initiative. *SIGCOMM Computer Communication Review.*, 37(3):89–92, 2007.
- [51] GENI Planning Group. GENI Design Principles. *IEEE Computer.*, 39(9):102–105, 2006.
- [52] Pasquale Di Gennaro, Roberto Bifulco, and Roberto Canonico. Link multiplexing in a xen-based network emulation system. In *Proceedings of the 2009 Workshop on Virtualization in High-Performance computing (VHPC’09)*, 2009.
- [53] Andreas Grau, Steffen Maier, Klaus Herrmann, and Kurt Rothermel. Time jails: A hybrid approach to scalable network emulation. In *PADS ’08: Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation*, pages 7–14, Washington, DC, USA, 2008. IEEE Computer Society.
- [54] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. VL2: a scalable and flexible data center network. *Communications of the ACM*, 54:95–104, March 2011.
- [55] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. Nox: towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, 2008.
- [56] S. Gundavelli, K. Leung, V. Devarapalli, K. Chowdhury, and B. Patil. Proxy Mobile IPv6. RFC 5213 (Proposed Standard), August 2008. Updated by RFC 6543.
- [57] hackingvoip.com. Sip invite flooder. <http://www.hackingvoip.com/tools/inviteflood.tar.gz>.
- [58] George C. Hadjichristofi, Avi Brender, Marco Gruteser, Rajesh Mahindra, and Ivan Seskar. A wired-wireless testbed architecture for network layer experimentation based on ORBIT and VINI. In *Proceedings*

- of ACM WINTECH 2007*, pages 83–90, Montréal, Québec (Canada), September 2007.
- [59] Stephen Hemminger. Network emulation with netem. <http://linux-net.osdl.org/index.php/Netem>. In *Linux Conf Au*, April 2005.
- [60] D. Herrscher, A. Leonhardi, and K. Rothermel. On node virtualization for scalable network emulation. In *Proceedings of the 2005 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS '05)*, July 2005.
- [61] D. Herrscher and K. Rothermel. A dynamic network scenario emulation tool. In *Proceedings of the 11th International Conference on Computer Communications and Networks (ICCCN 2002)*, October 2002.
- [62] Mike Hibler, Robert Ricci, Leigh Stoller, Jonathon Duerig, Shashi Guruprasad, Tim Stack, Kirk Webb, and Jay Lepreau. Large-scale virtualization in the emulab network testbed. In *ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 113–128, Berkeley, CA, USA, 2008. USENIX Association.
- [63] IEEE. Vlan tagging. IEEE 802.1Q .
- [64] Eucalyptus Systems Inc. Eucalyptus web site. <http://www.eucalyptus.com/>.
- [65] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, pages 1–12, New York, NY, USA, 2009. ACM.
- [66] Michael Jarschel, Simon Oechsner, Daniel Schlosser, Rastin Pries, Sebastian Goll, and Phuoc Tran-Gia. Modeling and performance evaluation of an openflow architecture. In *Proceedings of the 23rd International Teletraffic Congress, ITC '11*, pages 1–7. ITCP, 2011.
- [67] Juniper Networks. *Data Center LAN Connectivity Design Guide*, 2009.
- [68] Y. Kanada and T. Tarui. A Network-Paging Based Method for Wide-Area Live-Migration of VMs. In *Proceedings of the 25th International Conference on Information Networking (ICOIN 2011)*, January 2011.

- [69] Y. Kanaumi, S. Saito, and E. Kawai. Toward large-scale programmable networks: Lessons learned through the operation and management of a wide-area openflow-based network. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 330–333, oct. 2010.
- [70] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(1):291–307, 1970.
- [71] R. Koodli. Mobile IPv6 Fast Handovers. RFC 5268 (Proposed Standard), June 2008. Obsoleted by RFC 5568.
- [72] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. Onix: a distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI’10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.
- [73] Teemu Koponen, Scott Shenker, Hari Balakrishnan, Nick Feamster, Igor Ganichev, Ali Ghodsi, P. Brighten Godfrey, Nick McKeown, Guru Parulkar, Barath Raghavan, Jennifer Rexford, Somaya Arianfar, and Dmitriy Kuptsov. Architecting for innovation. *SIGCOMM Comput. Commun. Rev.*, 41(3):24–36.
- [74] UNINA Comics Lab. Distributed internet traffic generator. <http://www.grid.unina.it/software/ITG/>.
- [75] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets ’10, pages 19:1–19:6, New York, NY, USA, 2010. ACM.
- [76] E. Laure and et al. Programming the Grid with gLite. *Computational Methods in Science and Technology*, 12(1):33–45, 2006.
- [77] J. Ledlie, P. Gardner, and M. Seltzer. Network Coordinates in the Wild. In *Proceedings of NSDI 2007*, Cambridge (MA, USA), April 2007.
- [78] linux kvm.org. KVM web site. <http://www.linux-kvm.org>.

- [79] Y. Liu, H. Zhang, W. Gong, and D. Towsley. On the Interaction Between Overlay Routing and Underlay Routing. In *Proceedings of IEEE INFOCOM 2005*, pages 2543–2553, Miami (Florida, USA), March 2005.
- [80] C. Loomis, M. Airaj, M. Bégin, E. Floros, S. Kenny, and D. O’Callaghan. StratusLab Cloud Distribution. In D. Petcu and J.L. Vázquez-Poletti, editors, *European Research Activities in Cloud Computing*, pages 260–282. Cambridge Scholars Publishing, 2012.
- [81] L. Rizzo M. Carbone, G. Cecchetti. Integrated dummynet and planetlab. <http://www.onelab.eu/images/PDFs/Deliverables/d4e.2.pdf>. OneLab Project FP6-2004-IST-4 Public Deliverable D4E.2.
- [82] Thomas Magedanz and Sebastian Wahle. Control framework design for future internet testbeds. *e & i Elektrotechnik und Informationstechnik*, 126:274–279, 2009.
- [83] R. Mahindra, G. Bhanage, G. Hadjichristofi, S. Ganu, P. Kamat, I. Seskar, and D. Raychaudhuri. Integration of heterogeneous networking testbeds. In *Proceedings of TridentCom 2008*, Innsbruck (Austria), March 2008.
- [84] Steffen Maier, Daniel Herrscher, and Kurt Rothermel. Experiences with node virtualization for scalable network emulation. *Comput. Commun.*, 30(5):943–956, 2007.
- [85] V. Manetti, R. Canonico, G. Ventre, and I. Stavrakakis. System-level virtualization and mobile ip to support service mobility. In *Proceedings of the 2009 International Conference on Parallel Processing Workshops, ICPPW ’09*, pages 243–248, 2009.
- [86] Vittorio Manetti, Roberto Canonico, Giorgio Ventre, and Ioannis Stavrakakis. System-Level Virtualization and Mobile IP to Support Service Mobility. In *Proceedings of the 2009 International Conference on Parallel Processing Workshops, ICPP’09*, pages 243–248. IEEE Computer Society, September 2009.
- [87] Vittorio Manetti, Roberto Canonico, Giorgio Ventre, and Ioannis Stavrakakis. System-level virtualization and mobile ip to support service mobility. In *Proceedings of the International Workshop on Design, Optimization and Management of Heterogeneous Networked Systems (DOM-HetNetS’09)*, September 2009.

- [88] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communications Review*, 38:69–74, March 2008.
- [89] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.
- [90] P. Mell and T. Grance. The NIST definition of Cloud Computing.
- [91] M.S. Memon, Z. Nagy, E. Yen, and O. Koeroo. Virtualization and Cloud Computing Task Force Report V.0.7, 2012.
- [92] Aravind Menon, Alan L. Cox, and Willy Zwaenepoel. Optimizing network virtualization in xen. pages 15–28, 2006.
- [93] Aravind Menon, Jose R. Santos, Yoshio Turner, (john) G. Janakiraman, and Willy Zwaenepoel. Diagnosing performance overheads in the xen virtual machine environment. In *VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, pages 13–23. ACM Press, 2005.
- [94] Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, Andrew R. Curtis, and Sujata Banerjee. Devoflow: cost-effective flow management for high performance enterprise networks. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets '10, pages 1:1–1:6, New York, NY, USA, 2010. ACM.
- [95] Christopher Monsanto, Nate Foster, Rob Harrison, and David Walker. A compiler and run-time system for network programming languages. *SIGPLAN Not.*, 47(1):217–230, January 2012.
- [96] R. Moskowitz and P. Nikander. Host Identity Protocol (HIP) Architecture. RFC 4423 (Informational), May 2006.
- [97] National Institute of Standards and Technology. NIST Definition of Cloud Computing, 2009.
- [98] Erik Nordstrom, David Shue, Prem Gopalan, Rob Kiefer, Matvey Arye, Steven Ko, Jennifer Rexford, and Michael J. Freedman. Serval: An End-Host Stack for Service-Centric Networking. Technical report, October 2011.



- [99] D. Nurmi, R. Wolski, C. Grzegorzcyk, S. Obertelli, G. and Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus open-source cloud-computing system. In *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09)*, pages 124–131, May 2009.
- [100] University of Utah and Princeton University Proposal. Statement of Work: Exploring Federation of Testbeds with Diverse Models. Technical report, 2008.
- [101] openflow.org. Openflow - <http://www.openflow.org/>.
- [102] openflow.org. *The OpenFlow Switch Consortium* - <http://www.openflow.org>.
- [103] OpenFOAM-Foundation. OpenFOAM. <http://www.openfoam.com/>.
- [104] opennetworking.org. Open networking foundation. <https://www.opennetworking.org/>.
- [105] opennetworking.org. Openflow specification 1.1.0 - <http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf>.
- [106] opennetworking.org. Sdn white paper. <https://www.opennetworking.org/images/stories/downloads/openflow/wp-sdn-newnorm.pdf>.
- [107] orbit lab.org. Orbit lab. <http://www.orbit-lab.org/>.
- [108] Maximilian Ott, Ivan Seskar, Robert Siraccusa, and Manpreet Singh. ORBIT Testbed Software Architecture: Supporting Experiments as a Service. In *Proceedings of TridentCom 2005*, pages 136–145, Trento (Italy), February.
- [109] Andrea Passarella. Review: A survey on content-centric technologies for the current internet: Cdn and p2p solutions. *Comput. Commun.*, 35(1):1–32, January 2012.
- [110] Larry Paterson and Timothy Roscoe. The Design Principles of Planet-Lab. *ACM SIGOPS Operating Systems Review*, 40(1):11–16, January 2006.
- [111] C. Perkins. IP Mobility Support for IPv4. RFC 3220 (Proposed Standard), jan 2002. Obsoleted by RFC 3344.
- [112] C. Perkins. RFC 3220 - IP Mobility Support for IPv4, 2002.

- [113] C. Perkins. IP Mobility Support for IPv4, Revised. RFC 5944 (Proposed Standard), November 2010.
- [114] C. Perkins et al. IP mobility support, 1996.
- [115] C. Perkins, D. Johnson, and J. Arkko. Mobility Support in IPv6. RFC 6275 (Proposed Standard), July 2011.
- [116] Larry Peterson, Steve Muir, Timothy Roscoe, and Aaron Klingaman. PlanetLab Architecture: An Overview. Technical Report PDN-06-031, PlanetLab Consortium, May 2006.
- [117] Larry Peterson, Vivek Pai, Neil Spring, and Andy Bavier. Using PlanetLab for Network Research: Myths, Realities, and Best Practices. Technical Report PDN-05-028, PlanetLab Consortium, June 2005.
- [118] PlanetLab. PlanetLab Europe. <http://www.planet-lab.eu/>.
- [119] J. Postel. Internet Protocol. RFC 791 (Standard), sep 1981. Updated by RFC 1349.
- [120] J. Postel. Transmission Control Protocol. RFC 793 (Standard), sep 1981. Updated by RFCs 1122, 3168, 6093, 6528.
- [121] Princeton. VNET+ subsystem of PlanetLab . <http://www.cs.princeton.edu/sapanb/vnet/>.
- [122] Himabindu Pucha, Y. Charlie Hu, and Z. Morley Mao. On the impact of research network based testbeds on wide-area experiments. In *Proceedings of ACM IMC '06*, Rio de Janeiro (Brazil), December 2006.
- [123] Thierry Rakotoarivelo, Maximilian Ott, Guillaume Jourjon, and Ivan Seskar. OMF: A control and management framework for networking testbeds. *ACM SIGOPS Operating Systems Review*, 43(4):54–59, 2009.
- [124] D. Raychaudhuri, M. Ott, and I. Secker. ORBIT Radio Grid Tested for Evaluation of Next-Generation Wireless Network Protocols. In *Proceedings of TridentCom 2005*, pages 308–309, Trento (Italy), February 2005.
- [125] RedHat and et others. LibVirt virtualization API. <http://www.libvirt.org>.
- [126] Robert Ricci, Chris Alfeld, and Jay Lepreau. A solver for the network testbed mapping problem. *SIGCOMM Comput. Commun. Rev.*, 33(2):65–81, 2003.

- [127] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*, pages 199–212, New York, NY, USA, 2009. ACM.
- [128] Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *SIGCOMM Comput. Commun. Rev.*, 27(1):31–41, 1997.
- [129] Martin Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on Systems Administration (LISA-99), Seattle, WA, USA*, pages 229–238. USENIX, November 1999.
- [130] P. Saint-Andre. RFC 3921, Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence, 2004.
- [131] D. Salomoni, A. Italiano, and A. Ronchieri. WNoDeS, a tool for integrated Grid and Cloud access and computing farm virtualization. In *Proceedings of the International Conference on Computing in High Energy and Nuclear Physics (CHEP 2010)*, pages 18–35, 2011.
- [132] Kento Sato, Hitoshi Sato, and Satoshi Matsuoka. A Model-Based Algorithm for Optimizing I/O Intensive Applications in Clouds Using VM-Based Migration. In *Proceedings of IEEE/ACM CCGRID '09*, pages 466–471, May 2009.
- [133] Lucius Annaeus Seneca. Seneca ad Lucilium epistulae morales, epistula 71.
- [134] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru Parulkar. Can the production network be the testbed? In *Proceedings of the 9th USENIX conference on Operating systems design and implementation, OSDI'10*, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.
- [135] Hideyuki Shimonishi, Shuji Ishii, Lei Sun, and Yoshihiko Kanaumi. Architecture, implementation, and experiments of programmable network using openflow. *IEICE Transactions*, 94-B(10):2715–2722, 2011.
- [136] Ezra Silvera, Gilad Sharaby, Dean Lorenz, and Inbar Shapira. IP mobility to support live migration of Virtual Machines across subnets. In

- Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, pages 13:1–13:10. ACM, 2009.
- [137] H. Soliman, C. Castelluccia, K. ElMalki, and L. Bellier. Hierarchical Mobile IPv6 (HMIPv6) Mobility Management. RFC 5380 (Proposed Standard), October 2008.
- [138] Stephen Soltesz, Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. *ACM SIGOPS Operating Systems Review*, 41(3):275–287, 2007.
- [139] Marc Krochmal Stuart Cheshire and Kiren Sekar. Nat port mapping protocol (nat-pmp). Technical report, April 2008.
- [140] Brian Coghlan Stuart Kenny. Towards a grid-wide intrusion detection system. In *Proceedings of the European Grid Conference (EGC2005)*, pp. 275–284, Amsterdam, The Netherlands, February 2005.
- [141] Arsalan Tavakoli, Martin Casado, Teemu Koponen, and Scott Shenker. Applying nox to the datacenter. In *Proc. of workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.
- [142] Amin Tootoonchian and Yashar Ganjali. Hyperflow: a distributed control plane for openflow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, INM/WREN’10, pages 3–3, Berkeley, CA, USA, 2010. USENIX Association.
- [143] Francesco Tusa, Maurizio Paone, Massimo Villari, and Antonio Puliafito. Clever: A cloud cross-computing platform leveraging grid resources. In *UCC*, pages 390–396. IEEE Computer Society, 2011.
- [144] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Assoc, 2002. Dept. of Comput. Sci., Duke Univ., Durham, NC, USA.
- [145] Kleber Vieira, Alexandre Schulter, Carlos Westphall, and Carla Westphall. Intrusion detection techniques in grid and cloud computing environment. *IT Professional*, 99(PrePrints), 2009.

- [146] VMWare Inc. VMWare vMotion. <http://www.vmware.com/products/vmotion>.
- [147] voiptechchat.com.
- [148] voiptechchat.com. Amazon ec2 sip brute force attacks on rise. <http://www.voiptechchat.com/voip/457/amazon-ec2-sip-brute-force-attacks-on-rise/>.
- [149] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation. In *Proceedings of the 1st International Conference on Cloud Computing (CloudCom '09)*, pages 254–265. Springer-Verlag, 2009.
- [150] wired.com. Wired - going with the flow: Google's secret switch to the next wave of networking.
- [151] xensource.com. Xen wiki. <http://wiki.xensource.com/xenwiki/XenNetworking>.
- [152] H. Xie, R. Yang, A. Krishnamurthy, Y. Liu, and A. Silberschatz. P4P: Provider Portal for Applications. *ACM SIGCOMM Computer Communications Review (CCR)*, 38(4):351–362, October 2008.
- [153] Raid Y. Zaghal, Sandeep Davu, and Javed I. Khan. An interactive transparent protocol for connection oriented mobility-performance analysis with voice traffic. In *Proceedings of the Third International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks, WIOPT '05*, pages 219–228, Washington, DC, USA, 2005. IEEE Computer Society.